TOWARDS EFFICIENT GRAPH NEURAL NETWORKS FOR LARGE-SCALE RECOMMENDER SYSTEMS

JIAHAO ZHANG

MPhil

The Hong Kong Polytechnic University

2024

The Hong Kong Polytechnic University

Department of Computing

Towards Efficient Graph Neural Networks for Large-scale Recommender Systems

Jiahao Zhang

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Philosophy

April 2024

CERTIFICATE OF ORIGINALITY

Signature: ____________________

Name of Student: Jiahao Zhang

Abstract

In the era of information explosion, recommender systems are vital tools for delivering personalized recommendations for users by forecasting their future behaviours based on historical user-item interactions. To model these interaction behaviours, Graph Neural Networks (GNNs) have remarkably boosted the prediction performance of recommender systems due to their strong expressive power of capturing high-order information in user-item interactions through multi-layer embedding propagations. Nonetheless, classic Matrix Factorization (MF) and Deep Neural Network (DNN) approaches still play an important role in real-world, large-scale recommender systems due to their scalability advantages. Despite the existence of acceleration solutions, it remains an open question that whether GNN-based recommender systems can scale as efficiently as classic MF and DNN methods. In this thesis, we systematically reviewed the existing GNN-based recommendation models and investigated the inefficiency of these prior works. In response to this limitation, we propose a Linear-Time GNN (LTGNN) to scale up GNN-based recommender systems to achieve comparable scalability as the classic and efficient Matrix Factorization approaches while maintaining the powerful expressiveness for superior prediction accuracy. Detailed theoretical analysis are conducted to illustrate the expressive capability and training efficiency of the proposed LTGNN. We also present extensive empirical experiments and ablation studies to validate and understand the effectiveness and scalability of the proposed algorithm.

Publications Arising from the Thesis

1. Jiahao Zhang, Rui Xue, Wenqi Fan, Xin Xu, Qing Li, Jian Pei, Xiaorui Liu, "Linear-Time Graph Neural Networks for Scalable Recommendations", Accepted by *The Web Conference (WWW)* (2024).

Acknowledgments

I am deeply grateful to my supervisors, Prof. Wenqi Fan and Prof. Qing Li, for their profound expertise, understanding, and patience, which have significantly enriched my graduate experience. Their willingness to allow me the intellectual freedom to carve my own research path, coupled with their insightful guidance, has been essential to the progress and achievements of this work. I extend my heartfelt thanks to Prof. Xiaorui Liu for his rigorous guidance on my research projects and for nurturing my foundation in machine learning. His high standards and our shared passion have greatly shaped my academic journey, and for that, I am eternally appreciative.

My academic journey would not have been the same without the invaluable contributions of my collaborators, Mr. Rui Xue, Mr. Lin Wang, and Mr. Shijie Wang. Their assistance and thoughtful suggestions have been crucial to my research. I am equally thankful to my colleagues and friends, who have engaged in inspiring discussions, provided unwavering support in both my personal and academic life, and continually motivated me to chase my aspirations.

The support of my family and my partner (the best HCI researcher in the world :-)) has been the bedrock of my perseverance. Their unconditional love and encouragement have been my greatest source of strength, and it is with deep gratitude that I acknowledge their role in my achievements.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In an era of information explosion, recommender systems have become pivotal in enhancing the digital journey across various online platforms, owing to their exceptional capability in delivering personalized item suggestions. For instance, recommender systems have revolutionized users' experience in e-commerce [84, 90, 128] and streaming services [7, 20], while also profoundly influencing user engagement in social media platforms, such as Facebook [51] and LinkedIn [32]. Specifically, the success of these systems lies behind effectively predicting the users' incoming interactions with items by capturing their preferences from historical behaviors [41]. At the heart of this predictive power, collaborative filtering (CF) serves as a prevailing technique, which leverages the patterns across similar users and items to predict the users' preferences.

In the landscape of CF models, matrix factorization (MF) models stand as a general and foundational paradigm to capture the patterns in user-item interaction matrices. These models represent users and items in a low-dimensional embedding space and recover missing interactions in the user-item matrix through embedding similarities. Following the advent of MF models, a remarkable stream of literature has made great efforts to improve the expressive capability of these user and item embeddings by transforming them into more powerful latent representations. As discussed in many previous studies [102, 103, 67], this stream of literature can be categorized into two primary branches, based on their modeling ability of user-item interaction graphs. First, most early approaches in collaborative filtering focus on the *local connectivity* of users and items, such as item similarity models [57, 43] and deep neural networks (DNNs) [44, 117]. Second, due to the intrinsic limitation of modeling high-order connectivity in early CF models, recent years have witnessed a rising interest in graph neural networks (GNNs) in recommendations. To be specific, these GNN-based CF models extend beyond local interactions by iteratively aggregating embeddings along local neighborhood structures within the interaction graph [102, 41, 28]. This design enables them to encode both *local and long-range collaborative signals* into user and item representations, resulting in superior performance in modeling complex user-item interactions.

Despite the promising potential of GNNs in modeling high-order information in interaction graphs, the widespread adoption of GNN-based CF models in industrial-level applications remains limited due to scalability limitations [37, 119]. Classic CF models, such as MF and DNNs, continue to play major roles in real-world applications because of their computational advantages in large-scale industrial recommender systems [26, 19]. In particular, the computational advantage of classic CF models is inherent to their *linear* time complexity w.r.t. the number of user-item interactions in the interaction matrix. This is more efficient than the computation complexities of training GNN-based CF models, which are are *exponential* to the number of propagation layers or *quadratic* to the number of edges (as will be discussed in Section 2.2.5).

While numerous efforts have continued to accelerate the training of GNN-based recommender systems, including two main strategies focusing on neighbor sampling [119, 63] and design simplification [110, 41], none of them can achieve the linear complexity for GNN-based solutions, leading to inferior efficiency in comparison with conventional CF methods such as MF and DNNs. However, the problem size (i.e., the numbers of nodes and edges in the interaction graph) in web-scale recommender systems can easily reach a billion scale [99, 52], which necessitates a scalable GNN design with nearly linear or sub-linear complexity with respect to the problem size. Otherwise, GNN-based CF models could be infeasible in real-world and large-scale settings due to the unaffordable computational cost [94]. Thus, there is still an open question in academia and industry: *Whether GNN-based recommendation models can scale linearly as the classic MF and DNN methods, while exhibiting long-range modeling abilities and strong prediction performance.*

In this thesis, our primary research objective revolves around *1) preserving the strong expressive capabilities inherent in GNNs* while simultaneously *2) achieving a linear computation complexity* that is comparable to traditional CF models like MF and DNNs. However, it is highly non-trivial to pursue such a scalable GNN design, since the expressive power of high-order collaborative signals lies behind the number of recursive aggregations (i.e., GNN layers). Moreover, the embedding aggregation over a large number of neighbors is highly costly. To achieve a linear computation complexity, we propose a novel implicit graph modeling for recommendations with the *single-layer propagation* model design and an *efficient variance-reduced neighbor sampling* algorithm. Our contributions can be summarized as follows:

- We provide a critical complexity analysis and comparison of widely used collaboration filtering approaches, and we reveal their performance and efficiency bottlenecks.

- We propose a novel GNN-based model for large-scale collaborative filtering in recommendations, namely LTGNN (Linear Time Graph Neural Networks), which only incorporates *one propagation layer* while preserving the capability of capturing long-range collaborative signals.

- To handle large-scale user-item interaction graphs, we design an efficient and improved *variance-reduced neighbor sampling* strategy for LTGNN to significantly reduce the neighbor size in embedding aggregations. The random error caused by neighbor sampling is efficiently tackled by our improved variance reduction technique.

- We provide a detailed model analysis of LTGNN's long-range expressive capability, random error control, and desirable time complexity.
- We conduct extensive comparison experiments and ablation studies on three real-world recommendation datasets, including a large-scale dataset with millions of users and items. The experiment results demonstrate that our proposed LTGNN significantly reduces the training time of GNN-based CF models while preserving the recommendation performance on par with previous GNN models. We also perform detailed time complexity analyses to show our superior efficiency.

The subsequent sections of this thesis are organized as follows: Chapter 2 provides a foundational overview, beginning with the concepts and notations used in this study in Section 2.1. It then delves into a detailed survey of collaborative filtering in Section 2.2, followed by a survey of the scalability challenges inherent in graph neural networks in Section 2.3. Chapter 3 outlines the technical details and an in-depth analysis of the proposed method. Chapter 4 depicts the experiment results of our method. Finally, Chapter 5 presents the concluding remarks of this thesis.

Chapter 2

Background

In this chapter, we begin by introducing essential concepts and notations used throughout this thesis. Next, we review existing works in collaborative filtering (CF) and scalable graph neural networks (GNNs). Additionally, we analyze the time complexity of mainstream collaborative filtering models, which stands as a motivation behind our proposed approach.

2.1 Preliminaries

Collaborative Filtering (CF)

The essence of personalized recommendations lies in using historical user-item interactions to predict the most likely interactions in unobserved user-item pairs. Given user set $\mathcal{U} = \{u_1, u_2, \cdots, u_n\}$ and item set $\mathcal{I} = \{i_1, i_2, \cdots, i_m\}$, we denote all the possible user-item interactions as the Cartesian product of user and item sets $\mathcal{R} = \mathcal{U} \times \mathcal{I}$. The observed user-item interactions (i.e., training data) are represented as set $\mathcal{R}^+ \subset \mathcal{R}$, while the missing user-item interactions to be ranked by recommender systems are represented as $\mathcal{R}^-$, where $\mathcal{R}^+ \cup \mathcal{R}^- = \mathcal{R}$.

Table 2.1: Frequently used notations in this paper.

Notations	**Definitions**
$\mathbb{R}^d$	d-dimensional Euclidean space
a, $\boldsymbol{x}$, $\boldsymbol{X}$	Scalar, (column) vector, matrix
$\boldsymbol{X}^T, \boldsymbol{x}^T$	Matrix/vector transpose
$\lvert \cdot \rvert$	The cardinality of an arbitrary set
$\mathcal{R}^+$	Observed user-item interactions in recommendations
$\mathcal{R}^-$	Unobserved user-item interactions in recommendations
$\mathcal{G}$	The user-item interaction graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to represent $\mathcal{R}^+$
$\mathcal{V}$	The set of nodes in a graph
$\mathcal{E}$	The set of edges in a graph
v	A node $v \in \mathcal{V}$
$\mathcal{B}$	A batch of nodes $\mathcal{B} \subset \mathcal{V}$
$\boldsymbol{A}, \boldsymbol{D}$	The adjacency matrix and diagonal degree matrix of a graph
$\tilde{\boldsymbol{A}}$	The normalized adjacency matrix $\tilde{\boldsymbol{A}} = (\boldsymbol{D} + \boldsymbol{I})^{-\frac{1}{2}}(\boldsymbol{A} + \boldsymbol{I})(\boldsymbol{D} + \boldsymbol{I})^{-\frac{1}{2}}$
$\mathcal{N}(v)$	The set of neighbors of node v, including the node itself
$(\boldsymbol{E}_l^k)_{\mathcal{B}}$	Embedding matrix of node set $\mathcal{B}$ at training iteration k and GNN layer l
$(\boldsymbol{e}_l^k)_v$	Embedding vector of node v at training iteration k and GNN layer l
$\hat{\mathcal{N}}(v)$	A set with D nodes randomly sampled from $\mathcal{N}(v)$
$\hat{\boldsymbol{A}}$	The normalized adjacency matrix $\tilde{\boldsymbol{A}}$ with random neighbor sampling
$\hat{\boldsymbol{X}}, \hat{\boldsymbol{x}}$	Embedding matrices and vectors computed with random aggregation
$\bar{\boldsymbol{X}}, \bar{\boldsymbol{x}}$	Embedding matrices and vectors acquired from memory spaces

Collaborative filtering (CF) models represent users and items using trainable embeddings. Specifically, we construct the embedding matrix $\boldsymbol{E} = [\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n, \ \ \boldsymbol{e}_{n+1}, \ldots, \boldsymbol{e}_{n+m}]^T \in \mathbb{R}^{(n+m)\times d}$. The first n rows correspond to d-dimensional user embeddings, while the subsequent $n+1$ to $n+m$ rows represent d-dimensional item embeddings. The prediction of future user-item interactions involves transforming these embeddings into expressive latent representations and computing the similarity between user and item representations.

Graph-based Collaborative Filtering

To effectively leverage the high-order user-item connectivities in the observed user-item interaction $\mathcal{R}^+$, graph-based CF models represent the training data $\mathcal{R}^+$ as a bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The node set $\mathcal{V}$ includes n user nodes $\{v_1, \cdots, v_n\}$ and m item nodes $\{v_{n+1}, \cdots, v_{n+m}\}$, corresponding to users and items in $\mathcal{U}$ and $\mathcal{I}$. The edge set $\mathcal{E} = \{e_1, \cdots, e_{|\mathcal{E}|}\}$ consists of undirected edges connecting user nodes and item nodes, which is determined by the interactions in $\mathcal{R}^+$. Notably, the number of undirected edges $|\mathcal{E}|$ equals to the number of observed user-item interactions $|\mathcal{R}^+|$ in the training data (i.e., $|\mathcal{E}| = |\mathcal{R}^+|$).

For computational convenience, graph-based CF models denote the graph structure of $\mathcal{G}$ as the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{(n+m)\times(n+m)}$, where $\boldsymbol{A}_{i,j} = 1$ if there exists an interaction or edge between node v_i and node v_j; otherwise $\boldsymbol{A}_{i,j} = 0$. We also define the diagonal degree matrix based on the adjacency matrices as $\boldsymbol{D}$, where $\boldsymbol{D}_{i,i} = \sum_{j=1}^{n+m} \boldsymbol{A}_{i,j}$. The normalized adjacency matrix with self-loops is defined as $\tilde{\boldsymbol{A}} = (\boldsymbol{D} + \boldsymbol{I})^{-\frac{1}{2}}(\boldsymbol{A} + \boldsymbol{I})(\boldsymbol{D} + \boldsymbol{I})^{-\frac{1}{2}}$. Besides, we use $\mathcal{N}(v)$ to denote the set of neighboring nodes of a node v, including v itself.

Notations for Collaborative Filtering Models

Collaborative filtering (CF) models often employ deep architectures that compute multiple intermediate representations at each layer. In some cases, it also becomes necessary to describe the training process of CF models explicitly, including the index of training iterations. To address these two points, we introduce a slightly different notation system in this thesis, which uses $(\boldsymbol{E}_l^k)_\mathcal{B}$ or $(\boldsymbol{e}_l^k)_v$ to represent an embedding matrix or a single embedding vector. The inner subscript k denotes the index of training iterations, while another inner subscript l corresponds to the index of model layers. The outer subscript $(\cdot)_\mathcal{B}$ or $(\cdot)_v$ specifies whether the embedding refers to a batch of nodes $\mathcal{B}$ or an individual node v. In addition, to distinguish the input and output embeddings from the intermediate results, we denote the original trainable user and item embeddings at the input of CF models as $\boldsymbol{E}_{in}$ and $\boldsymbol{e}_{in}$, while the model's output embeddings are denoted as $\boldsymbol{E}_{out}$ and $\boldsymbol{e}_{out}$ for clarity.

In some graph-based CF models, intermediate embeddings involve stochastic computations with random neighbor sampling. We show the process of these stochastic computations with random neighbor sets $\hat{\mathcal{N}}(v)$ and random adjacency matrix $\hat{\mathcal{A}}$. Specifically, a random neighbor set $\hat{\mathcal{N}}(v)$ selects D random neighbors (without repetition) from the neighborhood $\mathcal{N}(v)$ of the target node v, while the normalized adjacency matrix obtained through random neighbor sampling is represented as $\hat{\mathcal{A}}$. Different from the intermediate embeddings computed without stochasticity, any matrices or vectors computed using random aggregations follow a similar notation as $\hat{\boldsymbol{X}}$ and $\hat{\boldsymbol{x}}$. For specific approaches that leverage stored embeddings from previous training iterations to mitigate stochastic effects, we denote the memory-based matrices and vectors similar to $\bar{\boldsymbol{X}}$ and $\bar{\boldsymbol{x}}$.

All the frequently used notations in this paper are summarized in Table. 2.1 for a quick reference.

2.2 Collaborative Filtering (CF)

2.2.1 Neighborhood-based Methods

Collaborative Filtering (CF) stands as one of the most prominent paradigms in modern recommender systems, aiming to understand users' preferences [29, 27]. The foundational concept of CF traces back to early email filtering systems [34], where individuals collaborated to enhance email filtering by recording their reactions to documents they encountered. This idea has significantly influenced the design of recommender systems, where users with similar historical behaviors are more likely to share similar preferences toward items [93].

During the initial stages of CF, recommender systems relied on neighborhood-based methods. These early practices represent each user by using a vector containing their ratings for every item in the system, thereby building user neighborhoods of similar users based on similarity measures, such as Pearson correlation or cosine similarity [80, 87]. Therefore, predictions for unrated items can be made by aggregating the average ratings from similar users.

However, neighborhood-based methods face limitations when dealing with large-scale and sparse user-item interaction data. Users often have few shared interacted items, and computing nearest neighbors becomes computationally expensive due to the large number of users and items [82]. Moreover, these methods fail to capture the latent associations between items, as different entries in the user-item rating vector may correspond to items sharing similar latent factors [82, 59]. To address these challenges, matrix factorization (MF) models emerged as a solution.

2.2.2 Matrix Factorization (MF)

Genearl Form

Matrix factorization (MF) has become a fundamental technique in collaborative filtering (CF) following its success in the Netflix Prize competition [7]. The essence of MF lies in decomposing the user-item interaction data into learnable user and item embeddings. This process effectively reduces the dimensionality of user-item interaction or rating vectors and represents users and items with compact latent factors The formulation of MF models is typically expressed as:

$$\begin{aligned} \min_{\boldsymbol{U},\boldsymbol{V}} \quad & \mathcal{L}(\boldsymbol{R}, \boldsymbol{U}^T\boldsymbol{V}) + r(\boldsymbol{U},\boldsymbol{V}), \\ \text{s.t.} \quad & \boldsymbol{U} \in \mathcal{C}_{\boldsymbol{U}}, \boldsymbol{V} \in \mathcal{C}_{\boldsymbol{V}} \end{aligned} \tag{2.1}$$

where the user-item interaction matrix $\boldsymbol{R} \in \mathbb{R}^{n\times m}$ is decomposed into trainable user embeddings $\boldsymbol{U} \in \mathbb{R}^{n\times d}$ and item embeddings $\boldsymbol{V} \in \mathbb{R}^{m\times d}$ under constraints $\mathcal{C}_{\boldsymbol{U}}$ and $\mathcal{C}_{\boldsymbol{V}}$. Here, the loss function $\mathcal{L}(\cdot,\cdot)$ measures the approximation error, and the second term $r(\cdot,\cdot)$ serves as a regularization term to prevent over-fitting.

Representative Models

Drawing inspiration from Latent Semantic Analysis (LSA) used in information retrieval [21], Sarwar et al. introduced the first matrix factorization (MF) model [82], applying a rank-reduced Singular Value Decomposition (SVD) to collaborative filtering (CF). The SVD model employs Mean Squared Error (MSE) as the loss function $\mathcal{L}$ to measure the discrepancy between the interaction matrix and its reconstructed counterpart $\boldsymbol{U}^T\boldsymbol{\Sigma}\boldsymbol{V}$, $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthonormal matrices, and $\boldsymbol{\Sigma}$ is a diagonal matrix containing the d-largest singular values of $\boldsymbol{R}$. This early model did not explicitly include a regularization term.

Subsequently, to address the issue of negative values in user and item embeddings, which reduced interpretability, Non-negative Matrix Factorization (NMF) techniques [61, 60] were developed. NMF models impose non-negativity constraints on $\boldsymbol{U}$ and $\boldsymbol{V}$, ensuring all embeddings are non-negative, which enhances the interpretability of the latent factors. The loss function $\mathcal{L}$ can be simple MSE objective [60], or divergence measures that match $\boldsymbol{R}$ and $\boldsymbol{U}^T\boldsymbol{V}$ from a probabilistic perspective [60, 17, 56]. While these models are more natural to be understood from the viewpoint of humans, they present optimization challenges and typically converge to local optima through alternating optimization techniques [105, 97].

Bias correction represents another significant strand of research in MF models, particularly for rating prediction tasks where biases are prevalent. For example, baseline ratings (i.e., average ratings) across different datasets may vary significantly. Some users may consistently rate items higher, while others may be more stringent in their ratings. To address this problem, Koren et al. adjust the global and user/item bias by subtracting the average rating from individual ratings during preprocessing [6]. Building upon this, the NSVD model introduced learnable parameters for each user and item to encapsulate their biases [75, 57]. Furthermore, recognizing that biases can fluctuate over time, Koren suggested a model that incorporates time t to parameterize user and item biases, thus capturing the temporal dynamics of bias. [58].

Despite the success of MF models in the Netflix Prize competition, they still encounter difficulties in generating expressive embeddings for cold-start users with a small number of ratings or when dealing with datasets that only provide implicit user feedback, which can be inherently noisy. To mitigate these challenges, item-similarity models were introduced. These models predict a user's preference for an item not solely based on the similarity between user and item embeddings but also considering the user's history of interactions [48, 72]. The standard approach for predicting user u's preference for item i involves calculating the inner product $\hat{y}_{u,i} = \boldsymbol{e}_u^T \boldsymbol{e}_i$, while item-similarity models extend this by incorporating the weighted sum of the embeddings of items which the user has interacted with:

$$\hat{y}_{u,i} = \boldsymbol{e}_i^T \left(\boldsymbol{e}_u + \sum_{(u,j)\in\mathcal{R}^+} c_{u,j} \boldsymbol{e}_j \right), \tag{2.2}$$

where $c_{u,j}$ denotes the confidence level of the interaction (u, j), and the specific formulation of this confidence measure can vary across different models (e.g., $c_{u,j} = \frac{1}{|\mathcal{N}(u)|}$ in PMF [72]). This predictive model utilizes a richer user representation that is particularly beneficial for cold-start users and in the context of noisy datasets, resulting in a broad impact on recommendation quality. Moreover, this concept seamlessly integrates with bias correction techniques, leading to the development of widely adopted MF variants such as FISM [53] and SVD++ [57].

The regularization term $r(\boldsymbol{U}, \boldsymbol{V})$ plays a crucial role in mitigating over-fitting and endowing the learned user and item embeddings with desirable characteristics. The most common and straightforward approach is L2-normalization, defined as $r(\boldsymbol{U}, \boldsymbol{V}) =$

$\frac{1}{2}\lambda_1\|\boldsymbol{U}\|_F^2+\frac{1}{2}\lambda_2\|\boldsymbol{V}\|_F^2$, where λ_1 and λ_2 are hyperparameters that control the strength of regularization [72, 57, 53]. Besides, sparse coding techniques are also employed to conserve space and computational resources while enhancing the interpretability and robustness of the representations [105, 46]. Notable methods for achieving sparse representations include Non-Negative Sparse Coding (NNSC) [46], which applies L1-regularization to item embeddings and L2-regularization to user embeddings, thereby emphasizing the most significant latent factors beneath user preferences. Another method, NMF with Sparseness Constraints (NMFSC) [47], introduces a novel sparsity measure that integrates both L2 and L1 regularization, which offers a more gradual and explainable sparsity constraint.

Probabilistic Interpretation

Adopting a probabilistic viewpoint offers insightful ideas for conceptualizing and innovating matrix factorization (MF) models. Probabilistic Matrix Factorization (PMF) [72], for instance, employs zero-mean spherical Gaussian priors for user and item latent factors and optimizes the posterior distribution to estimate model parameters. This approach inherently yields a vanilla matrix factorization model with L2-regularization, providing a novel perspective to comprehend the effect of regularization in MF models. Nonetheless, PMF's assumption that all latent factors are drawn from a singular Gaussian distribution fails to accommodate different variances among different latent factors. To overcome this, Bayesian Probabilistic Matrix Factorization (BPMF) [81] introduces a Gaussian-Wishart prior for user and item embeddings, allowing for an automatic adjustment to the diverse variances present in latent factors, with model inference facilitated by Gibbs sampling. Furthermore, another pivotal contribution to the probabilistic landscape of collaborative filtering is the Bayesian Personalized Ranking (BPR) [78] loss function, which optimizes a pairwise ranking objective for implicit user feedback and has emerged as one of the most widely adopted loss functions in the domain.

Limitations

Matrix factorization (MF) has established itself as a simple yet effective baseline in collaborative filtering and modern recommender systems. Its computational efficiency is particularly beneficial in large-scale applications. Nonetheless, the modeling capabilities of MF are inherently constrained. First, the manually chosen user-item similarity (e.g., inner product or cosine similarity) degrades the model's capacity to encapsulate more intricate similarity metrics. Moreover, the scope of MF in capturing long-range dependencies is limited, as it primarily explores the local connections in user-item interactions. In response to these limitations, more advanced collaborative filtering models like Deep Neural Networks (DNNs) and Graph Neural Networks (GNNs) have evolved from MF's core concepts, providing enhanced tools for more comprehensive interaction modeling.

2.2.3 Deep Neural Networks (DNNs)

The rise of deep learning in recent years has revolutionized fields ranging from computer vision [39, 38, 127] to natural language processing [98, 22, 10] and speech recognition [73, 2, 77]. The success of deep learning lies behind the strong modeling ability of Deep Neural Networks (DNNs), which employ multiple differentiable layers to extract non-linear and complex patterns from data. The successful application of DNNs has inspired the development of DNN-based collaborative filtering (CF) models, which build upon and enhance the foundational principles of matrix factorization (MF) with these modernized modeling techniques, pushing a new frontier of recommendation performance in this field.

General Form

Regardless of the model design and application scenario, Collaborative filtering (CF) models universally describe user-item interactions through a standard formulation [101, 112, 113]:

$$\hat{y}_{u,i} = f_{\Theta}(u, i), \tag{2.3}$$

where f denotes the model predicting the interaction score $\hat{u}_{u,i}$ for user u and item i, and Θ represents the model's trainable parameters. For instance, by setting Θ as $\Theta = \{\boldsymbol{U}, \boldsymbol{V}\}$, the vanilla matrix factorization model can be recovered with f comprising an embedding layer and an inner product operation:

$$\begin{aligned} \textbf{Embedding}: \quad \boldsymbol{e}_u &= \text{Embedding}(\boldsymbol{U}, u), \\ \boldsymbol{e}_i &= \text{Embedding}(\boldsymbol{V}, i), \\ \textbf{Interaction}: \quad \hat{y}_{u,i} &= \boldsymbol{e}_u^T \boldsymbol{e}_i. \end{aligned} \tag{2.4}$$

Similar to their MF predecessors, DNN-based CF models can also be conceptualized with a dual-component framework, including **1) representation learning** and **2) interaction modeling** [102, 112]. Specifically, the former is tasked with creating rich representations for the user index u and item index i, while the latter harnesses the power of neural networks to utilize the interactions between these user and item representations.

DNNs for Representation Learning

Representation learning is a fundamental aspect of DNN-based collaborative filtering, focusing on converting raw data into latent representations that can be effectively utilized for prediction. A common approach to deriving user and item representations is to directly adapt the matrix factorization (MF) framework, as shown in Eq. (2.4), which involves encoding the user index u and item index i with trainable embeddings [44]. An enhancement to this method includes incorporating user and item attributes, typically represented as multi-hot vectors, into the embeddings. For example, Neural Social Collaborative Ranking (NSCR) [101] assigns learnable embeddings to each user-item attribute (e.g., binary labels indicating “art lover” or “luxury” on travel websites) and calculates the latent representation by summing the element-wise products of user/item representations and attribute embeddings.

Additionally, another significant line of research in DNN-based representation learning involves utilizing neural networks to enrich user/item representations with historical user-item interactions, sharing similar intuitions with the item-similarity models. DeepMF [117], for instance, transforms the multi-hot user-item interaction vector (i.e., a row or a column in the user-item interaction matrix) into latent representations using multi-layer perceptrons (MLPs), while models like ACF [14], NAIS [43], and NNCF [3] employ attention mechanisms to selectively aggregate historical interactions, thereby capturing the most relevant parts in the history. This approach has further inspired research into autoencoder-based collaborative filtering [86, 114, 64, 69, 88], which leverages sophisticated generative models such as autoencoders (AEs) and variational autoencoders (VAEs) to attain superior user and item representations, outperforming those based solely on multi-hot vector transformation or attention mechanisms.

DNNs for Interaction Modeling

In DNN-based collaborative filtering (CF), after establishing meaningful representations, the focus shifts to interaction modeling. This stage aims to explore the complex and often non-linear interactions between users and items, leveraging the promising predictive power of neural networks. A straightforward practice is to employ the inner product layer in MF models, as seen in DeepMF [117] and ACF [14]. However, this approach may not fully capture the influences between latent factors, thus necessitating the use of DNNs.

Early DNN-based CF models address this by concatenating user and item representations and employing MLPs to predict a hidden state $\boldsymbol{h} = \mathrm{MLP}([\boldsymbol{e}_u \mid\mid \boldsymbol{e}_i])$ for user-item interactions [101, 44, 3]. Thus, the interaction score $\hat{y}_{u,i}$ can be estimated by linear combinations of the elements in the hidden state $\boldsymbol{h}$. However, empirical studies reveal that DNNs can be computationally expensive (i.e., requiring a large number of hidden state dimensions) to replicate the inner product between user and item representation [8, 79], which can be pivotal in capturing low-rank structures within the user-item interaction matrix [8]. To mitigate this issue, models like NeuMF [44] and NNCF [3] integrate the element-wise product of user and item representations into the interaction modeling process is beneficial, as in $\boldsymbol{h} = \mathrm{MLP}\left([\boldsymbol{e}_u \mid\mid \boldsymbol{e}_i \mid\mid (\boldsymbol{e}_u \odot \boldsymbol{e}_i)]\right)$, fostering both expressiveness and computational efficiency in DNN-based interaction modeling.

Beyond MF-based and MLP-based interaction modeling, alternative approaches such as CNNs and AEs are also viable. For instance, ONCF [42] offers an innovative approach by modeling interactions through the outer product of user-item representations and 2-D convolutional neural networks (CNNs). Meanwhile, autoencoder (AE)-based CF models [86, 114], on the other hand, adopt an encoder-decoder architecture to fill in the gaps of the user-item interaction matrix, enabling the capture of more complex and higher-order item interactions within user histories.

Limitations

Deep Neural Networks (DNNs) have emerged as pivotal tools to extend the traditional principles of collaborative filtering in an era of deep learning. Demonstrated to be both effective and efficient, these models have shown great promise in large-scale and industrial settings [20, 26]. Yet, the full potential of their modeling capabilities has not been unleashed. Like their matrix factorization (MF) counterparts, DNNs often struggle to capture high-order collaborative signals between users and items [102, 41].

This limitation can result in representations that do not fully reflect the complex interplay of user-item interactions, suggesting that there is still room for advancing these models to achieve their maximum potential.

2.2.4 Graph-based Collaborative Filtering

The integration of user interaction histories has become a common practice to enhance the latent representations within collaborative filtering (CF) models, including item-similarity MF models [57, 53], history-attention DNN models [3, 14, 43], and auto-encoder models [86, 64]. To understand this common practice from the lens of user-item interaction graphs, it is clear that a user's interaction history can be viewed as their first-order connectivity [102, 112]. This insight naturally leads to an extension of MF and DNN-based CF models to include the exploration of higher-order connections within the user-item interaction graph. Fortunately, Graph Neural Networks (GNNs), known for their effectiveness in various graph mining tasks such as node classification [55], link prediction [125], community detection [120], and subgraph classification [1], are well-suited for learning these high-order connectivities. In light of this, recent research has successfully integrated GNNs into CF-based recommender systems. These GNN-based models excel at capturing long-range dependencies through information propagation mechanisms on the user-item graph, offering a promising direction for advancing the representation learning process in CF models.

Graph Neural Networks

The development of GNN-based collaborative filtering (CF) commenced before the widespread adoption of DNN-based CF models, with the Graph Convolution Matrix Completion (GC-MC) [95] model being one of the earliest instances of such an approach. GC-MC pioneered the application of a GNN auto-encoder framework, which utilizes graph convolutional networks (GCNs) to encode user-item bipartite graphs and decode user-item interactions as a multi-class classification problem. Although GC-MC provided the groundwork for GNN-based recommender systems, its scalability to web-scale systems was still limited. Addressing this, Ying et al. developed the PinSAGE framework [119], which adapted the key idea of GraphSAGE [37] into the recommendation domain. By employing random sampling to reduce neighbor aggregation costs and incorporating techniques like importance pooling and curriculum training, PinSAGE achieved notable success on Pinterest, a content discovery platform with data at a billion-scale. Another landmark in GNN-based CF is the Neural Graph Collaborative Filtering (NGCF) model [102], which was the first to formulate the concept of collaborative signals, modeling them as high-order connectivities through message-passing mechanisms. This model revealed and strengthened the strong connections between collaborative filtering and graph machine learning. Subsequently, LightGCN [41] demonstrated the redundancy of non-linear activation functions and feature transformations in GNN-based CF models, advocating for a simplified design of such models. Based on this insightful idea, LightGCN achieved notable performance gains while maintaining a model of elegant simplicity, thereby establishing itself as a prevalent GNN backbone within the domain. These models have built a strong foundation for representation learning in graph-based CF models, creating space for further developments of other components in CF models, such as interaction modeling and data-centric approaches.

Graph Contrastive Learning

Following the establishment of strong GNN backbones for collaborative filtering, the research focus has expanded to other facets of the collaborative filtering (CF) framework, notably the loss function and training data. Graph contrastive learning (GCL) has emerged as a significant trend, formulating auxiliary loss functions through contrastive learning to delve deeper into the user-item interaction graph's structure. For example, Self-supervised Graph Learning (SGL) [111] integrates traditional recommendation objectives with an additional contrastive loss that minimizes discrepancies between two distinct graph views created via data augmentations. SimGCL [123] and XSimGCL [122] simplify this process by replacing graph augmentations with more efficient components, such as noise injection or layer reduction, while preserving the quality of recommendations. Moreover, Neighborhood-enriched Contrastive Learning (NCL) [66] aligns the representations of structurally adjacent nodes with their semantic counterparts, offering a fresh insight into loss function design tailored to GCL. Additionally, Robust Graph Contrastive Learning (RGCL) introduces decision boundary-aware adversarial perturbations to enhance both the robustness and accuracy of recommendations [92], while CDR [36] extends conventional graph contrastive learning on bipartite user-item graphs to more expressive tripartite graphs. These approaches provide extra supervisory signals that bring more distinctive node representations, thereby enhancing recommendation performance independent of the underlying GNN architecture.

Data-centric Methods

Recent advancements in graph-based CF also encompass improvements in negative sampling techniques and data-centric approaches. Enhanced negative sampling methods [50, 116] improve the negative sampling process in prevalent recommendation loss functions (e.g., BPR [78], BCE [16, 44]), aiming to yield more informative negative samples by leveraging the structural data of the user-item interaction graph. Besides, data-centric approaches [104, 40] concentrate on refining CF training data to boost recommendation quality or computational efficiency. For instance, the Collaboration-aware graph neural network (CAGCN) [104] evaluates the collaborative impact between node pairs in the user-item interaction graph, adjusting link weights to enrich the graph structure with recommendation-quality-related signals, while Simplified Graph Collaborative Filtering (SGCF) [40] compresses the interaction graph into a condensed form that can be trained with greater efficiency while sustaining recommendation performance. Another notable direction in data-centric methods involves constructing virtual nodes that represent broader sets of users or items, which accelerates the message-passing process on the modified graphs [11, 68]. Similar to GCL techniques, these methods revise the recommendation graph and also serve as general plugins for GNN-based collaborative filtering, remaining orthogonal to the choice of the GNN backbones.

Limitations

While GNN backbones and their auxiliary components have shown proficiency in capturing user preferences, scaling them to large-scale recommendation systems presents challenges, primarily due to the neighborhood explosion problem inherent to GNNs [37, 13]. Current solutions to this issue generally fall into two categories: design simplification [110, 41, 15] and neighbor sampling [119, 63]. Design simplification, inspired by LightGCN [41], aims to streamline GNN-based CF models by removing non-essential elements. However, these adjustments typically yield only constant-level time complexity improvements, which are insufficient for substantial acceleration. On the other hand, neighbor sampling, as implemented in PinSAGE [119], attempts to reduce the cost of embedding propagation by randomly selecting a limited subset of neighbors. This approach, while reducing computational overhead, can compromise the quality of recommendations due to the introduction of significant random errors. These limitations in existing graph-based CF models indicate that the scalability of GNN-based recommender systems remains an open question.

2.2.5 Time Complexity of Representative CF Models

Mini-batch Training in Recommendations

To analyze the time complexity of representative CF models, we first show the mini-batch training process with the typical pairwise loss function. This loss function samples two items for each user u: a positive item i (with observed interaction $(u, i) \in \mathcal{R}^+$) and a negative item j (with unobserved interaction $(u, j) \in \mathcal{R}^-$), thereby encouraging higher similarity between user representations and positive item representations compared to negative items. For instance, the most widely adopted Bayesian Personalized Ranking (BPR) [78] loss serves as an example:

$$\mathcal{L}_{BPR} = \sum_{(u,i,j)\in\mathcal{O}} -\ln\sigma(\hat{y}_{u,i} - \hat{y}_{u,j}), \tag{2.5}$$

where $\mathcal{O} = \{(u, i, j)|(u, i) \in \mathcal{R}^+, (u, j) \in \mathcal{R}^-\}$ denotes the pairwise training data, and $\hat{y}_{(\cdot,\cdot)}$ denotes the predicted similarity scores between a user and an item. The sets $\mathcal{R}^+$ and $\mathcal{R}^-$ correspond to observed and unobserved user-item interactions.

In practice, using the entire training data $\mathcal{O}$ is impractical due to the large number of user-item interactions [41, 102]. Instead, a common approach is to leverage $\mathcal{O}$ in a mini-batch setting, which involves splitting the original training data $\mathcal{O}$ into multiple components by randomly partitioning the observed user-item interactions in $\mathcal{R}^+$. Formally, these components are denoted as set $\mathbf{\Omega} = \{\mathcal{O}_{(u_1,i_1)}, \mathcal{O}_{(u_2,i_2)}, \cdots, \mathcal{O}_{(u_{|\mathcal{R}^+|}, i_{|\mathcal{R}^+|})}\}$, where $\mathcal{O}_{(u_r,i_r)} = \{(u_r, i_r, j)|(u_r, j) \in \mathcal{R}^-\}$ contains the corresponding negative samples for a specific interaction (u_r, i_r). In each training iteration, we sample B interactions from $\mathcal{R}^+$, which is denoted as $\hat{\mathcal{R}}^+$, satisfying $|\hat{\mathcal{R}}^+| = B$. Afterward, we create the training data for $\hat{\mathcal{R}}^+$ by merging the corresponding components in $\mathbf{\Omega}$, which can be denoted as $\hat{\mathcal{O}}(\hat{\mathcal{R}}^+) = \bigcup_{(u,i)\in\hat{\mathcal{R}}^+} \mathcal{O}_{(u,i)}$. Thus, the mini-batch training objective can be formalized as follows:

$$\hat{\mathcal{L}}_{BPR}(\hat{\mathcal{R}}^+) = \sum_{(u,i,j)\in\hat{\mathcal{O}}(\hat{\mathcal{R}}^+)} -\ln\sigma(\hat{y}_{u,i} - \hat{y}_{u,j}). \tag{2.6}$$

In each training epoch, we iterate over all user-item interactions in $\mathcal{R}^+$, so the mini-batch training objective $\hat{\mathcal{L}}_{BPR}$ needs to be evaluated for $|\mathcal{R}^+|/B$ times (i.e., $|\mathcal{E}|/B$ times). These understandings of the training process of CF models help reveal the computation complexities of mainstream CF models and further guide the design of our approach.

Analysis on Representative CF Models

In this subsection, we will briefly review MF and its DNN variant NCF, as well as two representative GNN-based CF models, including LightGCN [41] and PinSAGE [119], and discuss their computation complexity.

LightGCN. Inspired by the graph convolution operator in GCN [55] and SGC [110], LightGCN [41] iteratively propagates the user embedding $(\boldsymbol{e}_l)_u$ and item embedding $(\boldsymbol{e}_l)_i$ as follows:

$$(\boldsymbol{e}_{l+1})_u = \frac{1}{\sqrt{|\mathcal{N}(u)|}} \sum_{i \in \mathcal{N}(u)} \frac{1}{\sqrt{|\mathcal{N}(i)|}} (\boldsymbol{e}_l)_i, \tag{2.7}$$

$$(\boldsymbol{e}_{l+1})_i = \frac{1}{\sqrt{|\mathcal{N}(i)|}} \sum_{i \in \mathcal{N}(i)} \frac{1}{\sqrt{|\mathcal{N}(u)|}} (\boldsymbol{e}_l)_u. \tag{2.8}$$

The embedding propagation of LightGCN can be re-written in matrix form as follows:

$$\boldsymbol{E}_{l+1} = \tilde{\boldsymbol{A}} \boldsymbol{E}_l, \quad \forall l = 0, \ldots, L-1 \tag{2.9}$$

$$\boldsymbol{Y} = \frac{1}{L+1} \sum_{l=0}^{L} \boldsymbol{E}_l, \tag{2.10}$$

where L denotes the number of GNN layers, and $\boldsymbol{Y}$ denotes the model output of LightGCN with layer-wise combination. As LightGCN computes full embedding propagation in Eq. (2.9) for L times to capture L-hop neighborhood information, the computation complexity of LightGCN in one training iteration is $\mathcal{O}(L|\mathcal{E}|d)$ with the support of sparse matrix multiplications. Thus, the computation complexity for one training epoch is $\mathcal{O}(\frac{|\mathcal{E}|}{B} \cdot L|\mathcal{E}|d) = \mathcal{O}(\frac{1}{B} L|\mathcal{E}|^2 d)$, where $\frac{|\mathcal{E}|}{B}$ is the number of training iterations in one epoch.

PinSAGE. The embedding propagation in LightGCN aggregates all the neighbors for a user or an item, which is less compatible with Web-scale item-to-item recommender systems. Another important embedding propagation rule in GNN-based recommendation is proposed in PinSAGE:

$$(\boldsymbol{n}_{l+1})_u = \text{Aggregate}(\{\text{ReLU}(\boldsymbol{Q} \cdot (\boldsymbol{e}_l)_v + \boldsymbol{q}) \mid v \in \hat{\mathcal{N}}(u)\}), \tag{2.11}$$

$$(\boldsymbol{e}_{l+1})_u = \text{Normalize}(\boldsymbol{W} \cdot \text{Concat}[(\boldsymbol{e}_l)_u; (\boldsymbol{n}_{l+1})_u] + \boldsymbol{w}), \tag{2.12}$$

where $\boldsymbol{Q}, \boldsymbol{q}, \boldsymbol{W}, \boldsymbol{w}$ are trainable parameters, and $\hat{\mathcal{N}}(u)$ denotes the randomly sampled neighbors for node u. If PinSAGE constantly samples D random neighbors for each node at each layer, and the sampled B edges have n_B target nodes without repetition, the computation complexity in each training iteration is $\mathcal{O}(n_B D^L d^2)$ as discussed in previous studies [115]. Thus, the time complexity in the entire training epoch is $\mathcal{O}(\frac{|\mathcal{E}|}{B} \cdot n_B D^L d^2) = \mathcal{O}(|\mathcal{E}| D^L d^2)$, as n_B and B shares the same order. Moreover, the neighbor sampling in PinSAGE incurs large approximation errors that impact the prediction performance.

MF and NCF. Matrix factorization and its neural variant NCF [44] are simple but strong baselines for recommendations at scale. Given learnable user embedding $\boldsymbol{p}_u$ and item embedding $\boldsymbol{q}_i$, MF models their interaction directly by inner product as $\hat{y}_{u,i} = \boldsymbol{p}_u^T \boldsymbol{q}_i$, while NCF models the interaction by deep neural networks as follows:

$$\boldsymbol{e}_L = \phi(\boldsymbol{W}_L(\cdots \phi(\boldsymbol{W}_2 \phi(\boldsymbol{W}_1 \begin{bmatrix} \boldsymbol{p}_u \\ \boldsymbol{q}_i \end{bmatrix} + \boldsymbol{b}_1) + \boldsymbol{b}_2) \cdots) + \boldsymbol{b}_L), \tag{2.13}$$

$$\hat{y}_{u,i} = \sigma(\boldsymbol{h}^T \boldsymbol{e}_L), \tag{2.14}$$

where $\boldsymbol{W}$, $\boldsymbol{b}$ and $\boldsymbol{h}$ are trainable parameters, and ϕ is a non-linear activation function. In each training iteration, the computation complexity for MF and NCF is $\mathcal{O}(Bd)$ and $\mathcal{O}(BLd^2)$, which stands for the complexity of dot products and MLPs, respectively. Thus, the time complexity in each training epoch for MF and NCF is $\mathcal{O}(|\mathcal{E}|d)$ and $\mathcal{O}(|\mathcal{E}|Ld^2)$.

Inefficiency of GNNs. In comparison with conventional MF models, GNNs' inefficiency lies behind their non-linear complexity with respect to the number of edges $|\mathcal{E}|$ or layers L. For example, the time complexity for LightGCN is $\mathcal{O}(\frac{1}{B}L|\mathcal{E}|^2 d)$, which grows quadratically with $|\mathcal{E}|$, and PinSAGE has a complexity of $\mathcal{O}(|\mathcal{E}|D^L d^2)$, which grows exponentially with L. In this paper, we pursue a *linear-time* design for GNNs, which means the time complexity of our proposed model is expected to be $\mathcal{O}(C|\mathcal{E}|d)$, where C is a small constant (e.g., $C = Ld$ for NCF).

2.3 Scalability of Graph Neural Networks

Graph neural networks (GNNs) have risen as a essential technology in learning and mining from graph data, with profound impact on downstream applications such as recommender systems [119, 28, 41], protein design [83, 106], and knowledge-based systems [85, 91, 96]. The core mechanism behind GNNs' success is the message-passing paradigm, which empowers nodes to integrate semantic and structural information through recursive neighbor aggregations. Formally, the message-passing framework [33, 119, 30] can be formulated as follows:

$$(\boldsymbol{n}_{l+1})_u = \text{Aggregate}(\{\text{MSG}(u \leftarrow v) \mid v \in \mathcal{N}(u)\}), \tag{2.15}$$

$$(\boldsymbol{e}_{l+1})_u = \text{Update}\left((\boldsymbol{e}_l)_u, (\boldsymbol{n}_{l+1})_u\right), \tag{2.16}$$

where $\text{MSG}(u \leftarrow v)$ denotes a message function that produces a vector by combining the features of nodes u and v. This formulation reveals a critical challenge inherent in GNNs: the number of nodes involved in computations for a target node can grow exponentially with the depth of the GNN, due to recursive aggregation from preceding layers. This 'neighborhood explosion' [124] issue leads to significant computational and memory bottlenecks, limiting the deployment of GNNs in large-scale scenarios.

To address this problem, recent years have witnessed a rising research interest on the scalability techniques of GNNs. These methods aim to preserve the expressive power of GNNs while enhancing their efficiency on large-scale graph datasets, with a particular focus on **1) sampling methods**, **2) memory-based methods**, and **3) pre-computing & post-computing methods**.

Sampling-based Methods

Sampling-based methods reduce the computational and memory demands of graph neural networks (GNNs) by employing a mini-batch training paradigm. This approach strategically samples a constrained subset of nodes into computation, thereby diminishing the computational intensity in the training phase. The earliest practice in this line of research is found in node-wise sampling techniques [37, 12, 18], which randomly select a small number of neighbors for each target node for aggregation, rather than aggregating across the entire neighborhood. However, the exponential growth in the number of neighbors required for aggregation still persists, despite a reduction in the growth rate. To address this problem, the layer-wise sampling strategy [13, 130, 4] was introduced, which innovatively samples a fixed number of nodes at each GNN layer, rather than for each target node. This adjustment brings a linear correlation between the number of nodes for aggregation and the number of layers, which is highly desirable for computational efficiency. Besides, another vital solution in this domain is the subgraph-based sampling strategy [124, 89], which constructs stochastic subgraphs that depicts the essential characteristics of the original full graph, facilitating full-batch training on these distilled subgraphs. These methods have shown promising advances in improving the training efficiency of GNNs. However, these sampling-based methods inevitably omit a large number of neighbors for aggregation, which may sacrifice the prediction performance due to their large random errors.

Memory-based Methods

Memory-based methods [30, 121, 118] address the limitations of sampling-based approaches by utilizing historical feature representations to approximate the full aggregation outputs. For example, GNNAutoScale [30] stores historical embeddings of nodes at each layer within a non-blocking memory module. This method only updates the embeddings of the target and one-hop nodes in the aggregation computations, while the the embeddings of the out-of-batch nodes are approximated using their historical values. Subsequently, GraphFM [121] builds upon GNNAutoScale by introducing feature momentum to minimize the error introduced by reusing historical embeddings. Despite these advancements, these methods still have to compute full aggregations on a mixture of in-batch and out-of-batch nodes, which may still impede achieving a desirable linear computational complexity.

Pre-computing & Post-computing Methods

Furthermore, the decoupled design of node feature encoding and propagation in various GNN models [31] have inspired the development of pre-computing and post-computing methods. These techniques separate feature transformation from aggregation, allowing for the capture of long-range dependencies within graphs without performing feature aggregation during training. Specifically, pre-computing methods calculate the feature aggregation results prior to training [110, 9, 126], whereas post-computing methods train a feature transformation model first and then apply unsupervised algorithms, such as label propagation, for node label prediction [49, 129]. Although these methods have simplified the training process, they may compromise the benefits of end-to-end training and still exhibit notable approximation errors.

Limitations

Scalability remains a critical frontier in the evolution of Graph Neural Networks (GNNs), with various solutions attempting to address this challenge. Yet, as highlighted in our discussion, these solutions face two main limitations: they do not achieve linear complexity, and they are still insufficient to balance the trade-off between expressiveness and scalability. These issues also present significant obstacles in developing GNN-based collaborative filtering (CF) models that can operate effectively in large-scale recommender systems. These challenges consistently motivates our research in this thesis, which aims to create an innovative GNN framework for recommendations at scale, trying to surpass both specialized GNN-based CF models and general scalable GNNs.

Chapter 3

The Proposed Method

The scalability issue of GNN-based recommendation models inspires us to pursue a more efficient algorithm design with linear computation complexities. However, to reduce the computation complexity while preserving GNNs' long-range modeling ability, we need to overcome **two main challenges**:

- **Layer and expressiveness**: Increasing the number of layers L in GNNs can capture long-range dependencies, but the complexity (e.g., $\mathcal{O}(|\mathcal{E}|D^L d^2)$ in PinSAGE) can hardly be linear when L is large.

- **Neighbor aggregation and random error**: The number of neighbors D aggregated for each target node in a GNN layer substantially affects both computation cost and approximation error. Aggregating all neighbors (e.g., LightGCN) is costly, while aggregation with random sampling incurs large errors (e.g., PinSAGE).

To address these challenges, we propose implicit graph modeling in Section 3.1 to reduce the layer number L significantly, and a variance-reduced neighbor sampling strategy in Section 3.2 to lower the neighbor aggregation cost for high-degree nodes. We carefully handle the numerical and random errors of our designs, and ensure the strong expressiveness of our proposed LTGNN.

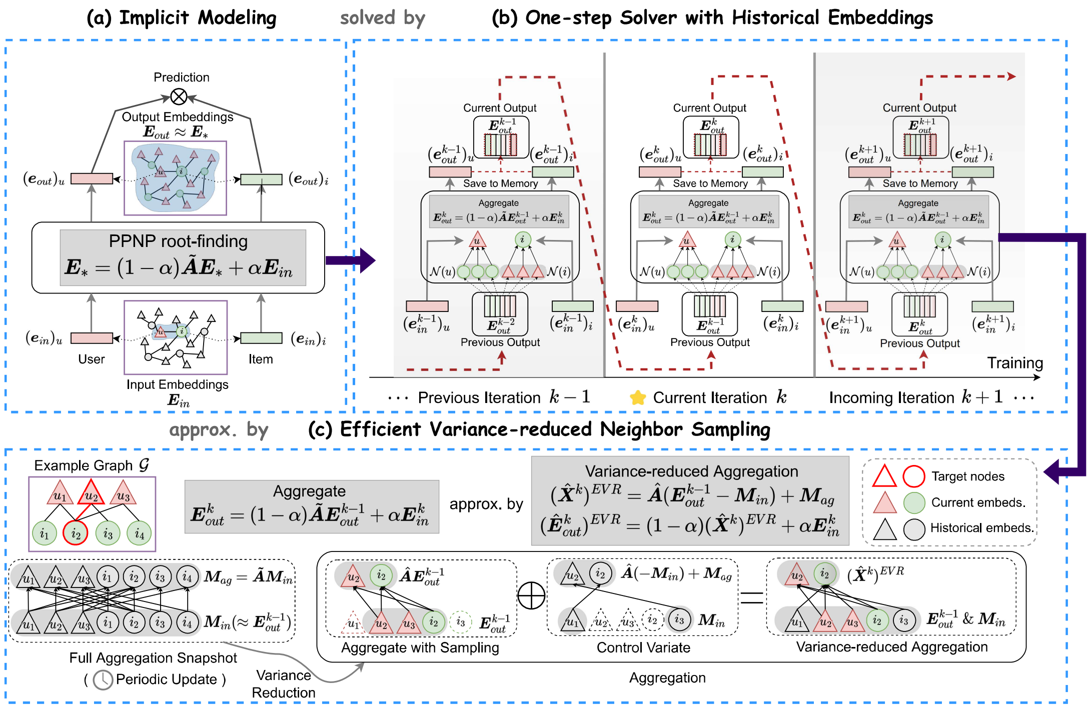
(a) Implicit Modeling
solved by
(b) One-step Solver with Historical Embeddings
Prediction
Output Embeddings
$E_{out} \approx E_*$
$(e_{out})_u$
$(e_{out})_i$
PPNP root-finding
$E_* = (1-\alpha)\tilde{A}E_* + \alpha E_{in}$
$(e_{in})_u$
$(e_{in})_i$
User
Item
Input Embeddings
E_{in}
Current Output
Save to Memory
Aggregate
$E^k_{out} = (1-\alpha)\tilde{A}E^{k-1}_{out} + \alpha E^k_{in}$
$\mathcal{N}(u)$
$\mathcal{N}(i)$
Previous Output
Training
··· Previous Iteration $k-1$
Current Iteration k
Incoming Iteration $k+1$ ···
approx. by
(c) Efficient Variance-reduced Neighbor Sampling
Example Graph $\mathcal{G}$
Aggregate
$E^k_{out} = (1-\alpha)\tilde{A}E^{k-1}_{out} + \alpha E^k_{in}$
approx. by
Variance-reduced Aggregation
$(\hat{X}^k)^{EVR} = \hat{A}(E^{k-1}_{out} - M_{in}) + M_{ag}$
$(\hat{E}^k_{out})^{EVR} = (1-\alpha)(\hat{X}^k)^{EVR} + \alpha E^k_{in}$
Target nodes
Current embeds.
Historical embeds.
$M_{ag} = \tilde{A}M_{in}$
$M_{in}(\approx E^{k-1}_{out})$
Full Aggregation Snapshot
(Periodic Update)
Variance Reduction
$\hat{A}E^{k-1}_{out}$
E^{k-1}_{out}
Aggregate with Sampling
$\hat{A}(-M_{in}) + M_{ag}$
M_{in}
Control Variate
$(\hat{X}^k)^{EVR}$
E^{k-1}_{out} & M_{in}
Variance-reduced Aggregation
Aggregation

Figure 3.1: An illustration of our model architecture. (a) The forward process of our model aims to solve the PPNP fixed-point equation, which expresses an equilibrium state of the embedding propagations, and can be used to capture long-range relations between any pair of nodes regardless of their distance. (b) The PPNP fixed-point equation is solved with a single forward propagation layer, which leverages the historical output embeddings in previous training iterations. (c) The process of efficient variance-reduced neighbor sampling in LTGNN.

3.1 Implicit Modeling for Recommendations

Personalized PageRank [74] is a classic approach for the measurement of the proximity between nodes in a graph. It is adopted by a popular GNN model, PPNP (Personalized Propagation of Neural Predictions) [31], to propagate node embeddings according to the personalized PageRank matrix:

$$E^k_{PPNP} = \alpha\Big(I - (1-\alpha)\tilde{A}\Big)^{-1} E^k_{in}, \tag{3.1}$$

where α is the teleport factor and $\boldsymbol{E}_{in}^k$ is the input node embedding. Due to the infeasible cost of matrix inversion, APPNP approximates this by L propagation layers:

$$\boldsymbol{E}_0^k = \boldsymbol{E}_{in}^k, \tag{3.2}$$

$$\boldsymbol{E}_{l+1}^k = (1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_l^k + \alpha\boldsymbol{E}_{in}^k, \ \forall l = 0, \dots, L-1 \tag{3.3}$$

such that it can capture the L-hop high-order information in the graph without suffering from over-smoothing due to the teleport term $\alpha\boldsymbol{E}_{in}^k$. LightGCN exhibits a close relation with APPNP although the embedding from different propagation layers is averaged with a different weight (see the analysis in Section 3.2 of [41]). However, like most GNNs, both APPNP and LightGCN suffer from scalability issues due to the multi-layer recursive feature aggregations, which greatly limit their applications in large-scale recommendations.

Motivated by the previous success of Implicit Deep Learning [25] and Implicit GNNs [35, 62, 118], we propose implicit modeling for graph-based recommendations (as shown in Fig. 3.1(a)), which directly computes the fixed point $\boldsymbol{E}_*^k$ of the embedding propagation in Eq. (3.3). Particularly, the output of our implicit model can be formalized as the solution of a linear system:

$$\boldsymbol{E}_{out}^k = \text{RootFind}(\boldsymbol{E}_*^k), \quad \text{s.t.} \quad \boldsymbol{E}_*^k = (1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_*^k + \alpha\boldsymbol{E}_{in}^k, \tag{3.4}$$

where the relation between output embedding $\boldsymbol{E}_{out}^k$ and input embedding $\boldsymbol{E}_{in}^k$ is implicitly defined by a root-finding process of the fixed-point equation. Formulating graph-based recommendations implicitly with a fixed-point equation has two advantages: 1) The fixed-point $\boldsymbol{E}_*^k$ models the equilibrium state of embedding propagations, which is equivalent to the extreme state under an infinite number of propagations, effectively capturing the long-range node dependencies in graphs. 2) This implicit modeling provides flexibility for the GNN design, as we can use any equation solver to acquire $\boldsymbol{E}_*^k$ instead of stacking multiple GNN layers.

Specifically, to pave the way to linear-time computation, we propose to solve this fixed-point equation by a single forward propagation layer, as shown in Fig. 3.1(b):

$$\boldsymbol{E}_{out}^{k} = (1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_{out}^{k-1} + \alpha\boldsymbol{E}_{in}^{k}, \tag{3.5}$$

where $\boldsymbol{E}_{out}^{k-1}$ is the historical output embeddings at previous training iteration $k-1$ and serves as a better initialization for the fixed-point solver. This single-layer design is ultra-efficient compared with multi-layer embedding propagations but still captures multi-hop neighbor information through information accumulation across training iterations.

The backward propagation of implicit models is independent of the forward computation [25, 35, 118]. Given the gradient from the output embedding layer $\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{out}^{k}}$, the gradient of $\boldsymbol{E}_{in}^{k}$ can be computed based on the fixed-point equation in Eq. (3.4):

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}} = \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{out}^{k}} \Big(\boldsymbol{I} - (1-\alpha)\tilde{\boldsymbol{A}}\Big)^{-1}. \tag{3.6}$$

Due to the prohibitively high dimensionality of the adjacency matrix, computing its inverse is infeasible. Therefore, we propose to approximate this gradient by a single backward propagation layer:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}} = (1-\alpha)\tilde{\boldsymbol{A}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}} + \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{out}^{k}}, \tag{3.7}$$

where $\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}}$ is the historical gradient of input embedding from iteration $k-1$ and serves as a better initialization for the fixed-point solver. In summary, the forward and backward computation of our single-layer GNN are formulated as:

$$\textbf{Forward}: \quad \boldsymbol{E}_{out}^{k} = (1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_{out}^{k-1} + \alpha\boldsymbol{E}_{in}^{k}, \tag{3.8}$$

$$\textbf{Backward}: \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}} = (1-\alpha)\tilde{\boldsymbol{A}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}} + \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{out}^{k}}, \tag{3.9}$$

where the historical computations $\boldsymbol{E}_{out}^{k-1}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}}$ can be obtained by maintaining the model outputs and gradients at the end of each training iteration. The numerical error of LTGNN in comparison with APPNP and PPNP will be shown in Section 3.4.2.

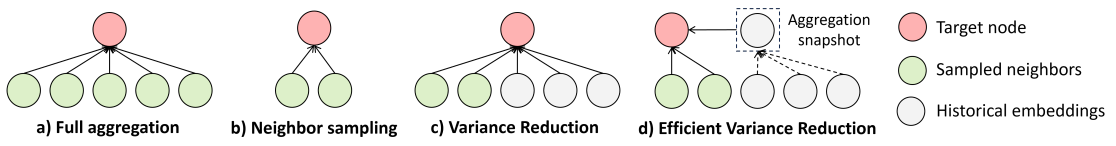

Figure 3.2: A comparison between full aggregation, neighbor sampling, vanilla variance reduction, and the proposed efficient variance reduction.

3.2 Efficient Variance-Reduced Neighbor Sampling

The previous section presents an implicit modeling and single-layer design for GNNs, which significantly reduces GNNs' computation complexity. For example, PinSAGE has a complexity of $\mathcal{O}(|\mathcal{E}|Dd^2)$ given there is only one layer ($L = 1$), which can be linear if D is a small constant. Unfortunately, D affects the expressiveness of GNNs in recommendations and cannot be easily lowered.

As explained in Section 2.2.5, the mini-batch training process samples user-item interactions (i.e., links in the user-item graph) in each iteration, which means that nodes with higher degrees are more likely to be sampled. These nodes also need more neighbors to compute their output embeddings accurately. Therefore, a small D will introduce large approximation errors and degrade the performance. This is consistent with previous studies on the impact of neighbor sampling in large-scale OGB benchmarks [24]. Some methods, such as VR-GCN [12] and MVS-GNN [18], use variance-reduction (VR) techniques to reduce the random error in neighbor sampling. However, we will show that these methods still require the full aggregation of historical embeddings, which maintains an undesirable complexity. To address this issue, we will propose an efficient VR neighbor sampling approach that achieves linear complexity while controlling the random error.

Classic Variance-reduced Neighbor Aggregation. Recent research has investigated variance reduction on GNNs, such as VR-GCN and MVS-GNN [12, 18]:

$$(\hat{\boldsymbol{X}}^k)^{VR} = \hat{\boldsymbol{A}}(\boldsymbol{E}_{in}^k - \overline{\boldsymbol{E}}_{in}^k) + \tilde{\boldsymbol{A}}\overline{\boldsymbol{E}}_{in}^k \quad (\approx \tilde{\boldsymbol{A}}\boldsymbol{E}_{in}^k), \tag{3.10}$$

where $\hat{\boldsymbol{A}}$ is an unbiased estimator of $\tilde{\boldsymbol{A}}$, $\hat{\boldsymbol{A}}_{u,v} = \frac{|\mathcal{N}(u)|}{D}\tilde{\boldsymbol{A}}_{u,v}$ if node v is sampled as a neighbor of target node u, otherwise $\hat{\boldsymbol{A}}_{u,v} = 0$. $\overline{\boldsymbol{E}}_{in}^{k}$ is the historical embeddings for approximating $\boldsymbol{E}_{in}^{k}$. However, as illustrated in Figure. 3.2(c), such approaches need to perform full neighbor aggregations on the historical embedding by computing $\tilde{\boldsymbol{A}}(\overline{\boldsymbol{E}}_{in}^{k})$. Importantly, this computation has to be performed in each mini-batch iteration, leading to the quadratic computation complexity $\mathcal{O}(\frac{|\mathcal{E}|^2 d}{B})$ for the whole training epoch. Therefore, they seriously sacrifice the computational efficiency of neighbor sampling in large-scale recommender systems. Besides, it is noteworthy that other GNN acceleration approaches based on historical embeddings [30, 121] also suffer from the full aggregation problem.

Efficient Variance-reduced Neighbor Sampling. To further reduce the quadratic computation complexity, we propose to compute the historical embedding aggregation periodically instead of computing them in every training iteration. Specifically, we allocate two memory variables $\boldsymbol{M}_{in}$ and $\boldsymbol{M}_{ag}$ to store the historical input embeddings and fully aggregated embeddings, where $\boldsymbol{M}_{ag} = \tilde{\boldsymbol{A}}\boldsymbol{M}_{in}$. The input memory variable $\boldsymbol{M}_{in}$ is updated periodically at the end of each training epoch, and the aggregated embedding $\boldsymbol{M}_{ag}$ are updated based on the renewed inputs. We name it as Efficient Variance Reduction (EVR), which can be formulated as:

$$(\hat{\boldsymbol{X}}^{k})^{EVR} = \hat{\boldsymbol{A}}(\boldsymbol{E}_{out}^{k-1} - \boldsymbol{M}_{in}) + \boldsymbol{M}_{ag} \qquad (\approx \tilde{\boldsymbol{A}}\boldsymbol{E}_{out}^{k-1}), \tag{3.11}$$

$$(\hat{\boldsymbol{E}}_{out}^{k})^{EVR} = (1-\alpha)(\hat{\boldsymbol{X}}^{k})^{EVR} + \alpha\boldsymbol{E}_{in}^{k} \qquad (\approx \boldsymbol{E}_{out}^{k}), \tag{3.12}$$

where the first term in Eq. (3.8) is approximated by $(\hat{\boldsymbol{X}}^{k})^{EVR}$, and the second term remains unchanged. An illustration of this sampling algorithm is shown in Fig. 3.1(c), and a comparison between the proposed EVR method and other sampling methods is shown in 3.2. Compared to other approaches, this innovative neighbor aggregation technique is highly desirable as it mimics full aggregation by considering all neighbors of the target nodes. Despite this, it introduces minimal computational overhead (i.e., almost as efficient as vanilla neighbor sampling as shown in Fig. 3.2(b)) by precomputing the full aggregation of historical embeddings into snapshot $\boldsymbol{M}_{ag}$, avoiding repeated full aggregation.

This variance reduction method can also be adapted to backward computations. Symmetrically, the backward computation in Eq. (3.9) can be computed with our proposed EVR as:

$$(\hat{\boldsymbol{G}}^k)^{EVR} = \hat{\boldsymbol{A}}(\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}} - \boldsymbol{M}'_{in}) + \boldsymbol{M}'_{ag} \qquad (\approx \tilde{\boldsymbol{A}}\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}}), \tag{3.13}$$

$$(\widehat{\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}}})^{EVR} = (1-\alpha)(\hat{\boldsymbol{G}}^k)^{EVR} + \alpha\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{out}^{k}} \qquad (\approx \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}}), \tag{3.14}$$

where $\boldsymbol{M}'_{in}$ stores the historical input gradients and $\boldsymbol{M}'_{ag} = \tilde{\boldsymbol{A}}\boldsymbol{M}'_{in}$ maintains the fully aggregated gradients. The random error of this variance-reduction design is detailed in Section 3.4.3.

3.3 Linear Time Graph Neural Networks

In this section, we present the Linear Time Graph Neural Network (LTGNN), which integrates implicit modeling and efficient variance-reduced neighbor sampling into a scalable and end-to-end GNN framework. While the main idea of these two modules has been established in previous sections, here we concentrate on the practical aspects of integrating these methodologies. Algorithm 1 outlines the step-by-step training procedure. The subsequent subsections will elaborate on the components critical to LTGNN's functionality, including embedding initialization, model training, model inference, and the details of neighbor aggregation.

```
Algorithm 1: The training process of LTGNN
Input: User-item interactions $\mathcal{R}$; BPR batch size $B$; Epochs $E$
Output: Optimized user-item embeddings $\boldsymbol{E}_{in}^{final}$
1  Initialize training iteration count $k \leftarrow 0$;
2  Initialize the user-item embeddings $\boldsymbol{E}_{in}^{0}$;
3  for $epoch = 1 \dots E$ do
4      Compute variance reduction memory
       $[\boldsymbol{M}_{in}; \boldsymbol{M}'_{in}] \leftarrow [\boldsymbol{E}_{out}^{k-1}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}}], [\boldsymbol{M}_{ag}; \boldsymbol{M}'_{ag}] \leftarrow \tilde{\boldsymbol{A}}[\boldsymbol{M}_{in}; \boldsymbol{M}'_{in}]$;
       $\triangleright \mathcal{O}(|\mathcal{E}|d)$
5      for sample $B$ interactions $\hat{\mathcal{R}}^{+}$ from $\mathcal{R}^{+}$ do        $\triangleright \frac{|\mathcal{E}|}{B}$ batches
6          Obtain training data $\hat{\mathcal{O}} = \{(u,i,j) | (u,i) \in \hat{\mathcal{R}}^{+}, (u,j) \in \mathcal{R}^{-}\}$;
7          Obtain the set of target nodes $\mathcal{B} = \bigcup_{(u,i,j)\in\hat{\mathcal{O}}} \{u,i,j\}$;
8          Sample the random neighbors for each target node in $\mathcal{B}$ to obtain
           $\hat{\boldsymbol{A}}$;
9          Obtain the forward output $\boldsymbol{E}_{out}^{k}$ according to Eq. (3.11) and
           Eq. (3.12);                                              $\triangleright \mathcal{O}(n_B D d)$
10         Compute the loss function $\mathcal{L}_{BPR}$ in Eq. (2.6) and the gradients at
           the output layer $\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{out}^{k}}$;                                 $\triangleright \mathcal{O}(Bd)$
11         Compute the implicit gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}}$ according to Eq. (3.13) and
           Eq. (3.14) ;                                              $\triangleright \mathcal{O}(n_B D d)$
12         Update the embedding table $\boldsymbol{E}_{in}$ with an arbitrary optimizer
           $\boldsymbol{E}_{in}^{k+1} \leftarrow \mathrm{UPDATE}(\boldsymbol{E}_{in}^{k}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}})$;                       $\triangleright \mathcal{O}(n_B d)$
13         Save $\boldsymbol{E}_{out}^{k}$ and $\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k}}$ to memory for the next training iteration;
14         $k \leftarrow k + 1$ ;                        $\triangleright$The for-loop costs $\mathcal{O}(|\mathcal{E}|Dd)$
15 return the optimized embeddings $\boldsymbol{E}_{in}^{final} = \boldsymbol{E}_{in}^{k}$.
```

Embedding Initialization (line 2). In the realm of collaborative filtering, typically only user-item interactions are available [44, 41], which contrasts with content-based recommendation systems that also utilize additional content features [108, 107]. Thus, we adopt the widely used strategy of assigning a unique trainable embedding to each user and item, as shown in line 2 in Algorithm 1:

$$\boldsymbol{E}_{in}^0 = [\ \underbrace{\boldsymbol{e}_1^0, \cdots, \boldsymbol{e}_n^0}_{\text{user embeddings}}, \underbrace{\boldsymbol{e}_{n+1}^0, \cdots, \boldsymbol{e}_{n+m}^0}_{\text{item embeddings}}\ ]^T \in \mathbb{R}^{(n+m)\times d}, \qquad (\textbf{Option I})$$

where $\boldsymbol{e}_i^0$ represents the embedding vector for the i-th entity (i.e., a user or an item), initialized by sampling from a Gaussian distribution. This approach is inherently adaptable to scenarios where additional content information is available for users and items. For instance, consider the encoded content feature vectors:

$$\boldsymbol{C} = [\ \underbrace{\boldsymbol{c}_1, \cdots, \boldsymbol{c}_n}_{\text{user features}}, \underbrace{\boldsymbol{c}_{n+1}, \cdots, \boldsymbol{c}_{n+m}}_{\text{item features}}\ ]^T,$$

which may be derived from pre-trained models [23, 109] or categorical attributes [26, 19] such as gender or age, or graph embeddings [65]. In such enriched contexts, the initial embeddings can be generated through a feature transformation function $f(\cdot;\boldsymbol{\Theta})$, parameterized by $\boldsymbol{\Theta}$:

$$\boldsymbol{E}_{in}^0 = [\underbrace{f(\boldsymbol{c}_1;\boldsymbol{\Theta}^0), \cdots, f(\boldsymbol{c}_n;\boldsymbol{\Theta}^0)}_{\text{user embeddings}}, \underbrace{f(\boldsymbol{c}_{n+1};\boldsymbol{\Theta}^0), \cdots, f(\boldsymbol{c}_{n+m};\boldsymbol{\Theta}^0)}_{\text{item embeddings}}]^T \in \mathbb{R}^{(n+m)\times d},$$

$$(\textbf{Option II})$$

where $\boldsymbol{\Theta}^0$ denotes the initial parameter set at the commencement of training, which evolves throughout the learning process. This flexible framework enables the LTGNN model to fit into diverse recommendation scenarios, effectively accommodating the complexities of user and item side information.

Model Training (line 3 - 14). The model training phase, as specified from lines 3 to 14, initiates with the initialized iteration counter k and the embeddings $\boldsymbol{E}_{in}^0$. The process repeats for E epochs, with each epoch starting by setting up the variance-reduction memory (line 4). This involves computing an aggregation snapshot using the previous iteration's embeddings $\boldsymbol{E}_{out}^{k-1}$ and their corresponding gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{E}_{in}^{k-1}}$. Specifically, this snapshot is initialized to zero for the first iteration.

Within each iteration, mini-batch interactions $\hat{\mathcal{R}}^+$ are sampled from the set of observed interactions $\mathcal{R}^+$. Each mini-batch undergoes negative sampling to form the training set $\hat{\mathcal{O}}$ (line 6), followed by neighbor sampling for nodes involved in GNN aggregation (lines 7 to 8). Subsequent to sampling, the forward and backward propagation steps of LTGNN are computed (lines 9 to 11), and model parameters are updated using an arbitrary optimizer (line 12). Notably, this model update strategy is adaptable to both collaborative filtering and content-based recommendation systems, as the next set of input embeddings can be derived by directly updating the trainable embeddings $\boldsymbol{E}_{in}^k$ or by updating the feature transformation model $f(\cdot; \boldsymbol{\Theta}^k)$. At the end of each iteration, the outputs from the forward and backward passes are preserved (line 13), enabling us to reuse these historical computations in the next training iteration

Model Inference (after line 15). After the model training process described in Algorithm 1, we have several options to predict user preference with the optimized input embeddings $\boldsymbol{E}_{in}^{final}$. The simplest solution is to directly infer the output embeddings with Eqs. (3.11) - (3.12), and then predict the future user-item interaction with an inner product. However, the training process may introduce small errors due to reusing the historical computations, which could cause $\boldsymbol{E}_{out}^{final}$ to deviate from the exact PPNP fixed-point $\boldsymbol{E}_{PPNP}^{final}$ w.r.t. $\boldsymbol{E}_{in}^{final}$. Therefore, a possible improvement is to compute $\boldsymbol{E}_{PPNP}^{final}$ with APPNP layers in Eqs. (3.2) - (3.3). This strategic choice can reduce the error due to reusing the previous computations, despite slightly compromising the behavior consistency between training and inference.

We find that both inference choices can accurately predict the user preference, and choosing either of them does not have a significant impact on the recommendation performance. In practice, we compute $\boldsymbol{E}_{PPNP}^{final}$ with a 3-layer APPNP which takes $\boldsymbol{E}_{in}^{final}$ as the input.

Neighbor Sampling and Aggregation. In this thesis, we have adopted $\hat{\boldsymbol{A}}$ as a simplified notation to represent the adjacency matrix with random neighbor sampling.

For each target node in $\mathcal{B}$, which consists of all the users, positive items, and negative items, we sample D random neighbors without repetition (including the node itself). We denote the set of random neighbors for a target node u as $\hat{\mathcal{N}}(u)$, and its original neighbor set as $\mathcal{N}(u)$. We define the elements of the random adjacency matrix $\hat{\boldsymbol{A}}$ as follows:

$$\hat{\boldsymbol{A}}_{u,v} = \begin{cases} \frac{|\mathcal{N}(u)|}{D}\tilde{\boldsymbol{A}}_{u,v}, & \text{if } u \in \mathcal{B} \text{ and } v \in \hat{\mathcal{N}}(u) \\ 0, & \text{otherwise} \end{cases}. \tag{3.15}$$

Thus, the matrix form embedding propagation $\hat{\boldsymbol{X}} = \hat{\boldsymbol{A}}\boldsymbol{E}$ is equivalent to the node-wise random aggregation:

$$\hat{\boldsymbol{x}}_u = \sum_{v \in \hat{\mathcal{N}}(u)} \frac{|\mathcal{N}(u)|}{D}\tilde{\boldsymbol{A}}_{u,v}\boldsymbol{e}_v, \qquad \forall u \in \mathcal{B}. \tag{3.16}$$

It is clear that $\hat{\boldsymbol{x}}_u$ is an unbiased estimator of exact aggregation $\boldsymbol{x}_u = \sum_{v\in\mathcal{N}(u)} \boldsymbol{A}_{u,v}\boldsymbol{e}_v$ for any node $u \in \mathcal{B}$, since

$$\begin{aligned}
\mathbb{E}[\hat{\boldsymbol{x}}_u] &= \frac{|\mathcal{N}(u)|}{D}\mathbb{E}\left[\sum_{v\in\mathcal{N}(u)} \tilde{\boldsymbol{A}}_{u,v}\boldsymbol{e}_v\mathbb{I}(v \mid u)\right] \\
&= \frac{|\mathcal{N}(u)|}{D}\sum_{v\in\mathcal{N}(u)} \tilde{\boldsymbol{A}}_{u,v}\boldsymbol{e}_v\mathbb{E}\left[\mathbb{I}(v \mid u)\right] \\
&= \frac{|\mathcal{N}(u)|}{D}\sum_{v\in\mathcal{N}(u)} \tilde{\boldsymbol{A}}_{u,v}\boldsymbol{e}_v\frac{\mathcal{C}^{D-1}_{|\mathcal{N}(u)|-1}}{\mathcal{C}^{D}_{|\mathcal{N}(u)|}} \\
&= \sum_{v\in\mathcal{N}(u)} \tilde{\boldsymbol{A}}_{u,v}\boldsymbol{e}_v = \boldsymbol{x}_u,
\end{aligned}$$

where $\mathbb{I}(v \mid u)$ is an indicator function that equals to 1 when v is sampled for target node u and 0 otherwise. This unbiasedness holds for both forward (line 9) and backward (line 11) computations in Algorithm 1, and is in line with previous discussions in VR-GCN [12].

3.4 Model Analysis

3.4.1 Time Complexity

Complexity Analysis. In each training epoch, the efficiency bottleneck lies in the forward and backward computations in line 9 and line 11. According to Eq. (3.15), there are $n_B D$ edges in the random adjacency matrix $\hat{\boldsymbol{A}}$, so the variance-reduced neighbor aggregation has complexity $\mathcal{O}(n_B D d)$ with the help of sparse matrix multiplications. Thus, the overall complexity of our method is $\mathcal{O}(\frac{|\mathcal{E}|}{B} \cdot n_B D d) = \mathcal{O}(|\mathcal{E}| D d)$, since the inner training loop in lines 5- 14 repeats $\frac{|\mathcal{E}|}{B}$ times, and B and n_B have the same order. Given that D is a small constant, the complexity becomes linear to the number of edges $|\mathcal{E}|$, demonstrating high scalability potential.

It is also noteworthy that our variance-reduced aggregation strategy does not affect the linear complexity of LTGNN. Particularly, the variance-reduced aggregations in lines 9 and 11 have the same complexity as vanilla neighbor sampling methods in PinSAGE, while the extra computational overhead of computing the variance reduction memory is $\mathcal{O}(|\mathcal{E}| d)$, which is as efficient as the simplest MF model (line 4). This means our proposed variance reduction approach reduces the random error without sacrificing the training efficiency.

Table 3.1: Complexity Comparisons.

Models	**Computation Complexity**
LightGCN	$\mathcal{O}(\frac{1}{B} L |\mathcal{E}|^2 d)$
PinSAGE	$\mathcal{O}(|\mathcal{E}| D^L d^2)$
MF	$\mathcal{O}(|\mathcal{E}| d)$
NCF	$\mathcal{O}(L |\mathcal{E}| d^2)$
LTGNN	$\mathcal{O}(|\mathcal{E}| D d)$

Comparison with Mainstream CF Models. The computational complexity of the proposed LTGNN model, as compared to mainstream CF models, is summarized in Table 3.1 (detailed in previous Section 2.2.5). Unlike GNN-based CF models such as LightGCN and PinSAGE, LTGNN's complexity does not scale quadratically or exponentially with the number of edges $|\mathcal{E}|$ or the depth of layers L, positioning it as a more scalable alternative. Moreover, LTGNN achieves a linear complexity with respect to $|\mathcal{E}|$, akin to simpler models like MF and NCF, while better capturing the complex long-range dependencies within graphs. This shows that LTGNN takes the strong expressive capability from GNN-based models, while achieving similar linear complexity to MF-based models, showing a perfect trade-off between scalability and expressiveness, which is crucial for practical large-scale recommendation systems.

Discussion on More Recent Graph-based CF Models. Besides, LTGNN not only enhances the balance between expressiveness and efficiency compared to mainstream CF models, but it also demonstrates computational benefits over the latest graph-based CF models focused on scalability. For instance, UltraGCN [71] replaces the embedding propagations in GNN-based CF models with an explicit constraint loss:

$$\max \sum_{i \in \mathcal{N}(u)} \beta_{u,i} \boldsymbol{e}_u^T \boldsymbol{e}_i, \quad u \in \mathcal{U},$$

where $\beta_{u,i}$ is a fixed weight derived from a GCN with infinite depth. This constraint loss effectively transforms GNN-based models into a type of weighted MF, maintaining linear complexity akin to MF and NCF. However, UltraGCN primarily leverages first-order connections and overlooks the higher-order interactions within the user-item interaction graph. In contrast, LTGNN addresses this limitation through its implicit GNN layers, which are capable of capturing these complex interactions. Furthermore, LTGNN's contributions, which focus on backbone-level innovations without specific assumptions about training objectives, can seamlessly integrate UltraGCN's constraint loss to enhance performance. This adaptability indicates that LTGNN is complementary to UltraGCN, rather than being in direct competition.

Additionally, SVD-GCN [76], another recent advancement in GNN-based CF models, tackles the scalability issue by decomposing the adjacency matrix $\tilde{\boldsymbol{A}}$ into its K-largest singular components using truncated SVD:

$$\tilde{\boldsymbol{A}} \approx \boldsymbol{P}_{(K)} \text{diag}\left(\boldsymbol{\sigma}_{\boldsymbol{(K)}}\right) \boldsymbol{P}_{(K)}^{T},$$

and then simplifying the embedding propagation to a linear transformation based on these components:

$$\boldsymbol{E}_{out} = \boldsymbol{P}_{(K)} \text{diag}\left(e^{\boldsymbol{\beta}^T \sigma_{(K)}}\right) \boldsymbol{W},$$

where $\boldsymbol{\beta}$ is a hyperparameter that modulates the influence of different connectivity ranges, and $\boldsymbol{W}$ is a learnable weight matrix. This design simplifies the propagated embeddings $\text{Poly}(\tilde{\boldsymbol{A}})\boldsymbol{E}_{in}$ ($\text{Poly}(\cdot)$ represents a polynomial combination) with a fixed embedding from SVD decomposition, and only trains a learnable linear transformation $\boldsymbol{W}$. As shown in the original paper, the training complexity of SVD-GCN is $\mathcal{O}(|\mathcal{E}|Kd)$, achieving linear complexity. However, SVD-GCN often requires a substantial number of singular values and vectors, with K typically ranging from 60 to 90, even for small-scale datasets. This requirement is less efficient than the neighbor count D in LTGNN's complexity, which maintains performance even when D is reduced to between 5 and 15, as demonstrated in Section 4.4. Moreover, SVD-GCN is primarily designed for collaborative filtering scenarios, where users and items are represented by a set of learnable embeddings $\boldsymbol{E}_{in}$. In contrast, LTGNN can incorporate user and item side information, offering greater flexibility for complex, real-world recommendation systems, as discussed in the previous Section 3.3.

3.4.2 Numerical Error

Relation to APPNP

As shown in Section 3.1, the forward propagation in LTGNN is an approximation of the Personalized PageRank (PPNP) fixed point, enabling the model to capture dependencies between nodes across varying distances within the interaction graph. To quantitatively demonstrate LTGNN's capacity for long-range interaction modeling, we first show its numerical error in comparison to a k-layer APPNP, which is a well-known approximation of PPNP, in Theorem 1. Additionally, we also introduce a important lemma which is useful in the proof of Theorem 1, presented as follows:

Lemma 1. *Given matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and arbitrary matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, we have*

$$\sigma_{min}(\boldsymbol{A})||\boldsymbol{X}||_F \leq ||\boldsymbol{A}\boldsymbol{X}||_F \leq \sigma_{max}(\boldsymbol{A})||\boldsymbol{X}||_F,$$

where $\sigma_{min}(\boldsymbol{A})$ and $\sigma_{min}(\boldsymbol{A})$ are minimum and maximum singular values of matrix $\boldsymbol{A}$.

Proof. Let $\boldsymbol{x}_i$ denote the i-th column vector of matrix $\boldsymbol{X}$, we have

$$||\boldsymbol{A}\boldsymbol{X}||_F^2 = tr(\boldsymbol{X}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X}) = \sum_{i=1}^{d} \boldsymbol{x}_i^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}_i,$$

Considering that $\boldsymbol{A}^T\boldsymbol{A}$ is a symmetric matrix, we can derive both the upper bound of $||\boldsymbol{A}\boldsymbol{X}||_F^2$ based on the Rayleigh's Theorem [45] as follows:

$$||\boldsymbol{A}\boldsymbol{X}||_F^2 = \sum_{i=1}^{d} \boldsymbol{x}_i^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}_i \leq \lambda_{max}(\boldsymbol{A}^T\boldsymbol{A}) \sum_{i=1}^{d} \boldsymbol{x}_i^T\boldsymbol{x}_i = \lambda_{max}(\boldsymbol{A}^T\boldsymbol{A})||\boldsymbol{X}||_F^2,$$

where $\lambda_{max}(\cdot)$ denotes the maximum eigenvalue of a matrix.

Similarly, we can also derive the lower bound based on Rayleigh's Theorem:

$$||\boldsymbol{A}\boldsymbol{X}||_F^2 = \sum_{i=1}^{d} \boldsymbol{x}_i^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}_i \geq \lambda_{min}(\boldsymbol{A}^T\boldsymbol{A}) \sum_{i=1}^{d} \boldsymbol{x}_i^T\boldsymbol{x}_i = \lambda_{min}(\boldsymbol{A}^T\boldsymbol{A})||\boldsymbol{X}||_F^2,$$

where $\lambda_{max}(\cdot)$ denotes the minimum eigenvalue of a matrix.

Thus, we have $\lambda_{min}(\boldsymbol{A}^T\boldsymbol{A})||\boldsymbol{X}||_F^2 \leq ||\boldsymbol{A}\boldsymbol{X}||_F^2 \leq \lambda_{max}(\boldsymbol{A}^T\boldsymbol{A})||\boldsymbol{X}||_F^2$. Note that $\sqrt{\lambda(\boldsymbol{A}^T\boldsymbol{A})}$ is the singular value of matrix $\boldsymbol{A}$, we can finish the proof of the lemma by applying the square root on both sides of the inequalities:

$$\lambda_{min}(\boldsymbol{A}^T\boldsymbol{A})||\boldsymbol{X}||_F^2 \leq ||\boldsymbol{A}\boldsymbol{X}||_F^2 \leq \lambda_{max}(\boldsymbol{A}^T\boldsymbol{A})||\boldsymbol{X}||_F^2$$

$$\Rightarrow \sqrt{\lambda_{min}(\boldsymbol{A}^T\boldsymbol{A})}||\boldsymbol{X}||_F \leq ||\boldsymbol{AX}||_F \leq \sqrt{\lambda_{max}(\boldsymbol{A}^T\boldsymbol{A})}||\boldsymbol{X}||_F$$
$$\Rightarrow \sigma_{min}(\boldsymbol{A})||\boldsymbol{X}||_F \leq ||\boldsymbol{AX}||_F \leq \sigma_{max}(\boldsymbol{A})||\boldsymbol{X}||_F,$$

where $\sigma_{min}(\boldsymbol{A})$ and $\sigma_{min}(\boldsymbol{A})$ are minimum and maximum singular values of matrix $\boldsymbol{A}$. □

Theorem 1. *At an arbitrary training iteration k, if the historical embeddings do not run stale too quickly, i.e., $\|\boldsymbol{E}_{in}^{j} - \boldsymbol{E}_{in}^{j-1}\|_F \leq \Delta$ $(j \leq k)$, then the error between the output of LTGNN $\boldsymbol{E}_{out}^k$ and a k-layer APPNP $\boldsymbol{E}_k^k$ is bounded by*

$$\|\boldsymbol{E}_{out}^k - \boldsymbol{E}_k^k\|_F \leq \left[(k-1)(1-\alpha)q^k + \frac{q}{(1-q)^2}\right]\Delta,$$

Proof. Based on the APPNP update rule in Eq. (3.3), the output of a k layer APPNP can be expanded as:

$$\begin{aligned}\boldsymbol{E}_k^k &= (1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_{k-1}^k + \alpha\boldsymbol{E}_{in}^k \\ &= (1-\alpha)^2\tilde{\boldsymbol{A}}^2\boldsymbol{E}_{k-2}^k + \alpha(1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_{in}^k + \alpha\boldsymbol{E}_{in}^k \\ &= \cdots\cdots \\ &= (1-\alpha)^k\tilde{\boldsymbol{A}}^k\boldsymbol{E}_{in}^k + \alpha\sum_{l=0}^{k-1}(1-\alpha)^l\tilde{\boldsymbol{A}}^l\boldsymbol{E}_{in}^k.\end{aligned} \tag{3.17}$$

Meanwhile, our proposed LTGNN can also be reformulated into a similar form as Eq. 3.17, where the input embedding $\boldsymbol{E}_{in}^k$ is replaced with input embeddings from previous training iterations. According to the update rule in Eq. (3.8), the output of LTGNN can be represented as:

$$\begin{aligned}\boldsymbol{E}_{out}^k &= (1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_{out}^{k-1} + \alpha\boldsymbol{E}_{in}^k \\ &= (1-\alpha)^2\tilde{\boldsymbol{A}}^2\boldsymbol{E}_{out}^{k-2} + \alpha(1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E}_{in}^{k-1} + \alpha\boldsymbol{E}_{in}^k \\ &= \cdots\cdots \\ &= (1-\alpha)^k\tilde{\boldsymbol{A}}^k\boldsymbol{E}_{in}^1 + \alpha\sum_{l=0}^{k-1}(1-\alpha)^l\tilde{\boldsymbol{A}}^l\boldsymbol{E}_{in}^{k-l}.\end{aligned} \tag{3.18}$$

It is clear that the approximation error between LTGNN and APPNP lies in the staleness of the input embedding matrix, as the computation of APPNP in Eq. 3.17 only relies on the input embedding at the current training iteration (i.e., $\boldsymbol{E}_{in}^k$), while the computation of LTGNN in Eq. 3.18 leverages the historical input embeddings from $\boldsymbol{E}_{in}^1$ to $\boldsymbol{E}_{in}^k$.

Based on Lemma 1, we get the error upper bound between LTGNN and APPNP:

$$\begin{aligned}
\|\boldsymbol{E}_{out}^k - \boldsymbol{E}_k^k\|_F &= \|(1-\alpha)^k \tilde{\boldsymbol{A}}^k (\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^1) + \alpha \sum_{l=0}^{k-1} (1-\alpha)^l \tilde{\boldsymbol{A}}^l (\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^{k-l})\|_F \\
&\le \|(1-\alpha)^k \tilde{\boldsymbol{A}}^k (\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^1)\|_F + \sum_{l=0}^{k-1} \|\alpha (1-\alpha)^l \tilde{\boldsymbol{A}}^l (\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^{k-l})\|_F \\
&= (1-\alpha)^k \|\tilde{\boldsymbol{A}}^k (\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^1)\|_F + \alpha \sum_{l=0}^{k-1} (1-\alpha)^l \|\tilde{\boldsymbol{A}}^l (\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^{k-l})\|_F \\
&\le (1-\alpha)^k \sigma_{max}^k(\tilde{\boldsymbol{A}}) \|\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^1\|_F + \alpha \sum_{l=0}^{k-1} (1-\alpha)^l \sigma_{max}^l(\tilde{\boldsymbol{A}}) \|\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^{k-l}\|_F
\end{aligned} \tag{3.19}$$

As assumption $\|\boldsymbol{E}_{in}^j - \boldsymbol{E}_{in}^{j-1}\|_F \le \Delta$ holds for all $j \le k$, we have $\|\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^1\|_F \le (k-1)\Delta$ and $\|\boldsymbol{E}_{in}^k - \boldsymbol{E}_{in}^{k-l}\|_F \le l\Delta$ due to the triangle inequality. Thus, inequality Eq. 3.19 can be transformed into:

$$\|\boldsymbol{E}_{out}^k - \boldsymbol{E}_k^k\|_F \le (k-1)(1-\alpha)^k \sigma_{max}^k(\tilde{\boldsymbol{A}})\Delta + \alpha \sum_{l=0}^{k-1} (1-\alpha)^l \sigma_{max}^l l \Delta \tag{3.20}$$

Simplifying the arithmetico-geometric series in the second term and reorganizing the formula, we have the following results:

$$\begin{aligned}
\|\boldsymbol{E}_{out}^k - \boldsymbol{E}_k^k\|_F &\le \left[(k-1)(1-\alpha)q^k + \frac{q(1-q^{k-1})}{(1-q)^2}\right]\Delta \\
&\le \left[(k-1)(1-\alpha)q^k + \frac{q}{(1-q)^2}\right]\Delta.
\end{aligned} \tag{3.21}$$

where $q = (1-\alpha)\sigma_{max}(\tilde{\boldsymbol{A}})$. The proof is finished. □

Theorem 1 illustrates that if the trainable embeddings do not go stale too quickly during the training process, the approximation error between LTGNN and a k-layer

APPNP will diminish exponentially with respect to the number of training iterations k. This indicates that as training progresses, LTGNN rapidly converges towards the output of APPNP. Furthermore, while the second term of the upper bound contains some constant error $\frac{q}{(1-a)^2}\Delta$, its impact remains minimal provided that q is not too close to 1, ensuring that this factor does not significantly affect the overall performance.

Convergence to PPNP

Building upon the established results regarding the approximation error between LT-GNN and APPNP, this part of the analysis demonstrates the approximation error between LTGNN and the exact PPNP. This comparison is pivotal as it directly reflects the long-range expressive capabilities of LTGNN. To show this approximation error, we first show the error bound between APPNP and PPNP in Lemma 2, and then show the convergence of LTGNN to PPNP in Corollary 1.

Lemma 2. *The iterations of APPNP converge to PPNP with linear convergence and rate $\mathcal{O}(q^k)$, where $q = (1-\alpha)\sigma_{max}(\tilde{\boldsymbol{A}})$.*

Proof. To prove Lemma 2, we first show the limit of APPNP propagations is PPNP, and then give the convergence rate. For the limit of APPNP with infinite iterations, we have

$$
\begin{aligned}
\boldsymbol{E}_*^k &= \lim_{k\to\infty} \boldsymbol{E}_k^k \\
&= \lim_{k\to\infty} \left[(1-\alpha)^k \tilde{\boldsymbol{A}}^k \boldsymbol{E}_{in}^k + \alpha \sum_{l=0}^{k-1} (1-\alpha)^l \tilde{\boldsymbol{A}}^l \boldsymbol{E}_{in}^k \right] \\
&= \lim_{k\to\infty} \left[\alpha \sum_{l=0}^{k-1} (1-\alpha)^l \tilde{\boldsymbol{A}}^l \boldsymbol{E}_{in}^k \right] \\
&= \alpha \lim_{k\to\infty} \left[\sum_{l=0}^{k-1} (1-\alpha)^l \tilde{\boldsymbol{A}}^l \right] \boldsymbol{E}_{in}^k \\
&= \alpha(\boldsymbol{I} - (1-\alpha)\tilde{\boldsymbol{A}})^{-1} \boldsymbol{E}_{in}^k = \boldsymbol{E}_{PPNP}^k,
\end{aligned}
$$

where the last equality holds because of the Neumann Series.

Given the APPNP propagation step $\phi : \boldsymbol{E} \rightarrow (1-\alpha)\tilde{\boldsymbol{A}}\boldsymbol{E} + \alpha\boldsymbol{E}_{in}$, it is obvious that ϕ is contraction mapping, as $\|\phi(\boldsymbol{E}_1) - \phi(\boldsymbol{E}_2)\|_F = \|(1-\alpha)\tilde{\boldsymbol{A}}(\boldsymbol{E}_1 - \boldsymbol{E}_2)\|_F \leq (1-\alpha)\sigma_{max}(\tilde{\boldsymbol{A}})\|\boldsymbol{E}_1 - \boldsymbol{E}_2\|_F$ and $0 < (1-\alpha)\sigma_{max}(\tilde{\boldsymbol{A}}) < 1$. Thus, we can conclude that $\{\boldsymbol{E}_k^k\}$ is a Cauchy sequence and $\boldsymbol{E}_k^k$ converges to fixed point $\boldsymbol{E}_*^k$ with rate $O([(1-\alpha)\sigma_{max}(\tilde{\boldsymbol{A}})]^k)$ due to the Banach Fixed Point Theorem [5], which shows the linear convergence. □

Corollary 1. *At an arbitrary training iteration k, if the historical embeddings do not run stale too quickly, i.e., $\|\boldsymbol{E}_{in}^j - \boldsymbol{E}_{in}^{j-1}\|_F \leq \Delta$ $(j \leq k)$, then the error between the output of LTGNN $\boldsymbol{E}_{out}^k$ and PPNP $\boldsymbol{E}_{PPNP}^k$ is bounded by*

$$\|\boldsymbol{E}_{out}^k - \boldsymbol{E}_{PPNP}^k\|_F \leq q^k \left[(k-1)(1-\alpha)\Delta + \|\boldsymbol{E}_{in}^k - \boldsymbol{E}_{PPNP}^k\|_F\right] + \frac{q}{(1-q)^2}\Delta,$$

where $q = (1-\alpha)\sigma_{max}(\tilde{\boldsymbol{A}})$. $\sigma_{max}(\tilde{\boldsymbol{A}})$ denotes the largest eigenvalue of normalized adjacency matrix $\tilde{\boldsymbol{A}}$.

Proof. Based on Lemma 2, it is clear that the iterations of APPNP converge to PPNP with the following error bound:

$$\|\boldsymbol{E}_k^k - \boldsymbol{E}_{PPNP}^k\|_F \leq q^k\|\boldsymbol{E}_{in}^k - \boldsymbol{E}_{PPNP}^k\|_F. \tag{3.22}$$

Meanwhile, Theorem 1 shows the convergence of error bound between LTGNN and k-layer APPNP:

$$\|\boldsymbol{E}_{out}^k - \boldsymbol{E}_k^k\|_F \leq \left[(k-1)(1-\alpha)q^k + \frac{q}{(1-q)^2}\right]\Delta. \tag{3.23}$$

Therefore, we can apply the triangle inequality and directly get the error bound between LTGNN and exact PPNP:

$$\|\boldsymbol{E}_{out}^k - \boldsymbol{E}_{PPNP}^k\|_F \leq q^k \left[(k-1)(1-\alpha)\Delta + \|\boldsymbol{E}_{in}^k - \boldsymbol{E}_{PPNP}^k\|_F\right] + \frac{q}{(1-q)^2}\Delta, \tag{3.24}$$

which finishes the proof. □

Corollary 1 shows the approximation error between LTGNN and exact PPNP decreases exponentially during the training process, given the embeddings are not running too stale in this process. The constant error term $\frac{q}{(1-q)^2}$ is not too large if hyperparameter α is not too close to 1. This theoretical result is also in line with our numerical experiments detailed in Section 4.4 and Figure 4.3.

3.4.3 Random Error

As shown in Section 3.2, LTGNN implements an efficient variance-reduced (EVR) neighbor aggregation method in both its forward and backward propagations. This method mitigates sampling variance through the utilization of a full aggregation snapshot derived from historical embeddings. The results in this subsection quantitatively demonstrate how this approach effectively reduces sampling variance, which is directly correlated with the random error inherent in neighbor aggregations.

In line with previous methodologies discussed in VR-GCN [12], we commence by presenting the sampling variance associated with the standard neighbor sampling estimator. The following theorem illustrates this concept:

Theorem 2. *For any target node $u \in \mathcal{B}$, if $\hat{\mathcal{N}}(u)$ contains D random neighbors sampled from $\mathcal{N}(u)$ without repetition, then the variance of the vanilla neighbor sampling estimator $\hat{\boldsymbol{x}}_u = \sum_{v \in \hat{\mathcal{N}}(u)} \hat{\boldsymbol{A}}_{u,v} \boldsymbol{e}_v$ is*

$$Var\left(\hat{\boldsymbol{x}}_u\right) = \frac{|\mathcal{N}(u)| - D}{2D|\mathcal{N}(u)|\left(|\mathcal{N}(u)| - 1\right)} \sum_{v_1 \in \mathcal{N}(u)} \sum_{v_2 \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{v_1,v_2} \left(\boldsymbol{e}_{v_1} - \boldsymbol{e}_{v_2}\right)^2 .$$

Proof. Let $X_1, X_2, \cdots, X_N$ be random variables. A widely used proposition is as follows:

$$Var(\sum_{i=1}^{N} X_i) = \sum_{i=1}^{N} \sum_{j=1}^{N} Cov(X_i, X_j).$$

Let $\mathbb{I}(v \mid u)$ be an indicator function that equals to 1 when v is sampled for target node u and 0 otherwise. For simplicity, we also represent the degree of a node u

(including self-loops) as $d_u = |\mathcal{N}(u)|$. Based on the proposition given above, we have:

$$\begin{aligned}
Var(\hat{\boldsymbol{x}}_u) &= Var\left(\sum_{v \in \hat{\mathcal{N}}(u)} \hat{\boldsymbol{A}}_{u,v}\boldsymbol{e}_v\right) \\
&= Var\left(\frac{d_u}{D}\sum_{v \in \hat{\mathcal{N}}(u)} \frac{1}{\sqrt{d_u}\sqrt{d_v}}\boldsymbol{e}_v\right) \\
&= \frac{d_u}{D^2} Var\left(\sum_{v \in \mathcal{N}(u)} \frac{1}{\sqrt{d_v}}\boldsymbol{e}_v \mathbb{I}(v \mid u)\right) \\
&= \frac{d_u}{D^2}\sum_{v_1 \in \mathcal{N}(u)}\sum_{v_2 \in \mathcal{N}(u)} Cov\left(\frac{1}{\sqrt{d_{v_1}}}\boldsymbol{e}_{v_1}\mathbb{I}(v_1 \mid u), \frac{1}{\sqrt{d_{v_2}}}\boldsymbol{e}_{v_2}\mathbb{I}(v_2 \mid u)\right) \\
&= \frac{d_u}{D^2}\sum_{v_1 \in \mathcal{N}(u)}\sum_{v_2 \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{v_1,v_2}\boldsymbol{e}_{v_1}\boldsymbol{e}_{v_2}\left(\mathbb{E}\left[\mathbb{I}(v_1 \mid u)\mathbb{I}(v_2 \mid u)\right] - \mathbb{E}\left[\mathbb{I}(v_1 \mid u)\right]\mathbb{E}\left[\mathbb{I}(v_2 \mid u)\right]\right)
\end{aligned}$$

To continue the proof, we must tackle the challenge of evaluating the expectations within the summation. The product of expectations in the second term is readily simplified, as $\mathbb{E}\left[\mathbb{I}(v_1 \mid u)\right]\mathbb{E}\left[\mathbb{I}(v_2 \mid u)\right] = \frac{D^2}{d_u^2}$. The first term, however, requires careful consideration under two distinct scenarios concerning v_1 and v_2. For the case where $v_1 = v_2$, it simplifies to $\mathbb{E}\left[\mathbb{I}(v_1 \mid u)\mathbb{I}(v_2 \mid u)\right] = \mathbb{E}[\mathbb{I}(v_1 \mid u)^2] = \frac{D}{d_u}$. Differently, when $v_1 \neq v_2$, the expectation is expressed as $\mathbb{E}\left[\mathbb{I}(v_1 \mid u)\mathbb{I}(v_2 \mid u)\right] = \mathbb{E}[\mathbb{I}(v_1, v_2 \mid u)] = C_{d_u-2}^{D-2}/C_{d_u}^{D} = \frac{D(D-1)}{d_u(d_u-1)}$, where $\mathbb{I}(v_1, v_2 \mid u)$ indicates that both v_1 and v_2 are sampled from $\mathcal{N}(u)$. Therefore, we decompose the original double summation into two distinct cases, addressing $v_1 = v_2$ and $v_1 \neq v_2$ separately, and then proceed to calculate the variance accordingly:

$$\begin{aligned}
Var(\hat{\boldsymbol{x}}_u) &= \frac{d_u}{D^2}\sum_{v_1 \in \mathcal{N}(u)}\sum_{v_2 \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{v_1,v_2}\boldsymbol{e}_{v_1}\boldsymbol{e}_{v_2}\left(\mathbb{E}\left[\mathbb{I}(v_1 \mid u)\mathbb{I}(v_2 \mid u)\right] - \mathbb{E}\left[\mathbb{I}(v_1 \mid u)\right]\mathbb{E}\left[\mathbb{I}(v_2 \mid u)\right]\right) \\
&= \frac{d_u}{D^2}\sum_{v \in \mathcal{N}(u)}\left(\frac{\boldsymbol{e}_v}{\sqrt{d_v}}\right)^2\left(\frac{D}{d_u} - \frac{D^2}{d_u^2}\right) + \frac{d_u}{D^2}\sum_{\substack{v_1,v_2 \in \mathcal{N}(u)\\ v_1 \neq v_2}} \tilde{\boldsymbol{A}}_{v_1,v_2}\boldsymbol{e}_{v_1}\boldsymbol{e}_{v_2}\left[\frac{D(D-1)}{d_u(d_u-1)} - \frac{D^2}{d_u^2}\right] \\
&= \frac{d_u - D}{Dd_u(d_u-1)}\left[(d_u-1)\sum_{v \in \mathcal{N}(u)}\left(\frac{\boldsymbol{e}_v}{\sqrt{d_v}}\right)^2 - \sum_{\substack{v_1,v_2 \in \mathcal{N}(u)\\ v_1 \neq v_2}} \frac{\boldsymbol{e}_{v_1}\boldsymbol{e}_{v_2}}{\sqrt{d_{v_1}}\sqrt{d_{v_2}}}\right]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{d_u - D}{2Dd_u(d_u-1)} \sum_{v_1,v_2 \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{v_1,v_2}(\boldsymbol{e}_{v_1} - \boldsymbol{e}_{v_2})^2 \\
&= \frac{|\mathcal{N}(u)| - D}{2D|\mathcal{N}(u)|\left(|\mathcal{N}(u)|-1\right)} \sum_{v_1 \in \mathcal{N}(u)} \sum_{v_2 \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{v_1,v_2}\left(\boldsymbol{e}_{v_1} - \boldsymbol{e}_{v_2}\right)^2 .
\end{aligned}
$$

□

In Theorem 2, we identify that the sampling variance (i.e., random error) of conventional random neighbor aggregation is determined by the squared differences between each pair of embeddings in the neighborhood of a given target node u. To further derive the variance of the Enhanced Variance Reduction (EVR) estimator proposed in LTGNN, we first reformulate the matrix-form EVR aggregations, as specified in Eqs. (3.11) to (3.14), into a node-centric vector representation:

$$
\hat{\boldsymbol{x}}_u^{EVR} = \sum_{v \in \hat{\mathcal{N}}(u)} \hat{\boldsymbol{A}}_{u,v}(\boldsymbol{e}_v - \boldsymbol{m}_v) + \sum_{v \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{u,v}\boldsymbol{m}_v,
$$

where $\boldsymbol{m}_v$ denotes the stored historical embedding of node v stored in memory $\boldsymbol{M}_{in}$ or $\boldsymbol{M}'_{in}$, and the second term correspond to the complete aggregation for computing the full aggregation snapshot $\boldsymbol{M}_{ag} = \tilde{\boldsymbol{A}}\boldsymbol{M}_{in}$ and $\boldsymbol{M}'_{ag} = \tilde{\boldsymbol{A}}\boldsymbol{M}'_{in}$. The stochasticity is confined to the first term, indicating that the variance of the EVR estimator can be directly inferred from Theorem 2 by substituting the embedding vectors $\boldsymbol{e}_{v_1}$ and $\boldsymbol{e}_{v_2}$ with the embedding differences $\delta\boldsymbol{e}_{v_1} = \boldsymbol{e}_{v_1} - \boldsymbol{m}_{v_1}$ and $\delta\boldsymbol{e}_{v_2} = \boldsymbol{e}_{v_2} - \boldsymbol{m}_{v_2}$.

Remark 1 (Variance Reduction)**.** *The sampling variance of the proposed EVR method is*

$$
Var\left(\hat{\boldsymbol{x}}_u\right) = \frac{|\mathcal{N}(u)| - D}{2D|\mathcal{N}(u)|\left(|\mathcal{N}(u)|-1\right)} \sum_{v_1 \in \mathcal{N}(u)} \sum_{v_2 \in \mathcal{N}(u)} \tilde{\boldsymbol{A}}_{v_1,v_2}\left(\delta\boldsymbol{e}_{v_1} - \delta\boldsymbol{e}_{v_2}\right)^2 .
$$

As training progresses, the differences $\delta\boldsymbol{e}_{v_1}$ and $\delta\boldsymbol{e}_{v_2}$ between the current and historical embeddings decrease, leading to reduced sampling variance.

To validate this variance reduction effect, we examine the difference in magnitudes between $\boldsymbol{e}$ and $\delta\boldsymbol{e}$, which correlates with the initial and reduced variance levels. We

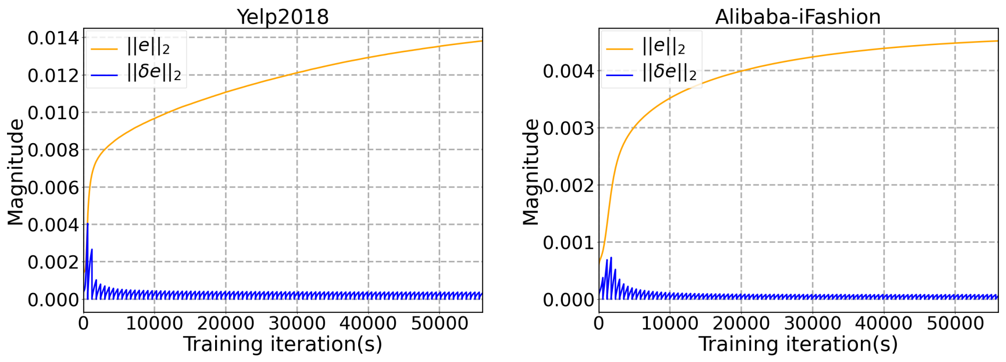

Figure 3.3: Comparison of vector magnitudes between $\boldsymbol{e}$ and $\delta\boldsymbol{e}$.

apply our model to two different datasets, and the average magnitudes of the two vectors across all nodes are shown in Fig. 3.3. The results indicate that the magnitude of the embedding difference $\delta\boldsymbol{e}$ is significantly smaller than that of the original embedding $\boldsymbol{e}$. Notably, this substantial gap is achieved within 5 training epochs (approximately 5000 iterations). The periodic fluctuations in $\|\delta\boldsymbol{e}\|_2$ are due to the periodic updates of the variance reduction memory variables $\boldsymbol{M}_{in}$ and $\boldsymbol{M}_{ag}$.

Chapter 4

Experiments

In this section, we will verify the effectiveness and efficiency of the proposed LT-GNN framework with comprehensive experiments. Specifically, we aim to answer the following research questions:

- **RQ1**: Can LTGNN achieve promising prediction performance on large-scale recommendation datasets? (Section 4.2)
- **RQ2**: Can LGTNN handle large user-item interaction graphs more efficiently than existing GNN approaches? (Section 4.3)
- **RQ3**: How does the effectiveness of the proposed LTGNN vary when we ablate different parts of the design? (Section 4.4)

4.1 Experimental Settings

We first introduce the datasets, baselines, evaluation metrics, and hyperparameter settings as follows.

Datasets. We evaluate the proposed LTGNN and baselines on two medium-scale

Table 4.1: Dataset statistics.

Dataset	# Users	# Items	# Interactions	Avg. Degree
Yelp2018	31, 668	38, 048	1, 561, 406	46
Alibaba-iFashion	300, 000	81, 614	1, 607, 813	10
Amazon-Large	872, 557	453, 553	15, 236, 325	23

datasets, including *Yelp2018* and *Alibaba-iFashion*, and one large-scale dataset *Amazon-Large*. Yelp2018 dataset is released by the baselines NGCF [102] and LightGCN [41], and the Alibaba-iFashion dataset can be found in the GitHub repository[1]. For the large-scale setting, we construct the large-scale dataset, Amazon-Large, based on the rating files from the Amazon Review Data website[2]. Specifically, we select the three largest subsets (i.e., Books, Clothing Shoes and Jewelry, Home and Kitchen) from the entire Amazon dataset, and then keep the interactions from users who are shared by all the three subsets (7.9% of all the users). The rating scale is from 1 to 5, and we transform the explicit ratings into implicit interactions by only keeping the interactions with ratings bigger than 4. To ensure the quality of our Amazon-Large dataset, we follow a widely used 10-core setting [44, 102, 100] and remove the users and items with interactions less than 10. The statistical summary of the datasets can be found in Table 4.1, where "Avg. Degree" represents the average node degree in each dataset.

Baselines. The main focus of this paper is to enhance the scalability of GNN-based collaborative filtering methods. Therefore, we compare our method with the most widely used GNN backbone in recommendations, *LightGCN* [41] and its scalable variants that employ typical GNN scalability techniques, including GraphSAGE [37], VR-GCN [12] and GAS [30]. The corresponding variants of LightGCN are denoted as *LightGCN-NS* (for Neighbor Sampling), *LightGCN-VR*, and *LightGCN-GAS*.

[1]`https://github.com/wenyuer/POG`

[2]`https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/`

To demonstrate the effectiveness of our method, we also compare it with various representative recommendation models, including MF [57], NCF [44], GC-MC [95], PinSAGE [119], NGCF [102], and DGCF [41]. Moreover, since we are designing an efficient collaborative filtering backbone that is independent of the loss function, our method is orthogonal to SSL-based methods [111, 123] and negative sampling algorithms [70, 50]. We will explore the combination of our method and these orthogonal designs in future work. We also incorporate one of the latest accuracy-driven GNN backbones, GTN [27], to show the strong recommendation performance of our proposed LTGNN.

Evaluation and Parameter Settings. We implement the proposed LTGNN using PyTorch and PyG libraries. We strictly follow the settings of NGCF [102] and LightGCN [41] to implement our method and the scalable LightGCN variants for a fair comparison. All the methods use an embedding size of 64, a BPR batch size of 2048, and a negative sample size of 1. For the proposed LTGNN, we tune the learning rate from $\{5\text{e-}4, 1\text{e-}3, 1.5\text{e-}3, 2\text{e-}3\}$ and the weight decay from $\{1\text{e-}4, 2\text{e-}4, 3\text{e-}4\}$. We employ an Adam [54] optimizer to minimize the objective function. For the coefficient α in PPNP, we perform a grid search over the hyperparameter in the range of [0.3, 0.7] with a step size of 0.05. To ensure the scalability of our model, the number of propagation layers L is fixed to 1 by default, and the number of sampled neighbors D is searched in $\{5, 10, 15, 20\}$. For the GNN-based baselines, we follow their official implementations and suggested settings in their papers. For the LightGCN variants with scalability techniques, the number of layers L is set based on the best choice of LightGCN, and we search other hyperparameters in the same range as LTGNN and report the best results.

In this paper, we adopt two widely used evaluation metrics in recommendations: Recall@K and Normalized Discounted Cumulative Gain (NDCG@K) [102, 41]. We set the K=20 by default, and we report the average result for all test users. All the experiments are repeated five times with different random seeds, and we report the average results.

4.2 Recommendation Performance

Table 4.2: The comparison of overall prediction performance.

Dataset	**Yelp2018**		**Alibaba-iFashion**		**Amazon-Large**	
Method	**Recall@20**	**NDCG@20**	**Recall@20**	**NDCG@20**	**Recall@20**	**NDCG@20**
MF	0.0436	0.0353	0.05784	0.02676	0.02752	0.01534
NCF	0.0450	0.0364	0.06027	0.02810	0.02785	0.01807
GC-MC	0.0462	0.0379	0.07738	0.03395	OOM	OOM
PinSAGE	0.04951	0.04049	0.07053	0.03186	0.02809	0.01973
NGCF	0.0581	0.0475	0.07979	0.03570	OOM	OOM
DGCF	0.0640	0.0522	0.08445	0.03967	OOM	OOM
LightGCN (L=3)	<u>0.06347</u>	<u>0.05238</u>	0.08793	0.04096	**0.0331**	<u>0.02283</u>
LightGCN-NS (L=3)	0.06256	0.05140	<u>0.08804</u>	<u>0.04147</u>	0.02835	0.02035
LightGCN-VR (L=3)	0.06245	0.05141	0.08814	0.04082	0.02903	0.02093
LightGCN-GAS (L=3)	0.06337	0.05207	0.08169	0.03869	0.02886	0.02085
LTGNN (L=1)	**0.06393**	**0.05245**	**0.09335**	**0.04387**	<u>0.02942</u>	**0.02585**

In this section, we mainly examine the recommendation performance of our proposed LTGNN, with a particular focus on comparing LTGNN with the most widely adopted GNN backbone LightGCN. We use out-of-memory (OOM) to indicate the methods that cannot run on the dataset due to memory limitations. The recommendation performance summarized in Table 4.2 provides the following observations:

- Our proposed LTGNN achieves comparable or better results on all three datasets compared to the strongest baselines. In particular, LTGNN outperforms all the baselines on Yelp and Alibaba-iFashion. The only exception is that the Recall@20 of LightGCN (L=3) outperforms LTGNN (L=1) on the Amazon-Large dataset. However, our NDCG@20 outperforms LightGCN (L=3), and LTGNN (L=1) is much more efficient compared with LightGCN (L=3), as LTGNN only uses one embedding propagation layer and very few randomly sampled neighbors.

- The scalable variants of LightGCN improve the scalability of LightGCN by sacrificing its recommendation performance in most cases. For instance, the results for LightGCN-VR, LightGCN-NS, and LightGCN-GAS are much worse than LightGCN with full embedding propagation on Amazon-Large. In contrast, the proposed LTGNN has better efficiency than these variants and preserves the recommendation performance.
- The performance of GNN-based methods like NGCF and LightGCN consistently outperforms earlier methods like MF and NCF. However, GNNs without scalability techniques can hardly be run large-scale datasets. For instance, GC-MC, NGCF, and DGCF significantly outperform MF, but they are reported as OOM on the Amazon-Large dataset. This suggests the necessity of pursuing scalable GNNs for improving the recommendation performance in large-scale industry scenarios.

Table 4.3: Recommendation performance comparison results with extra baseline GTN. Results marked with (*) are obtained in GTN's official settings.

Dataset	**Yelp2018**		**Alibaba-iFashion**		**Amazon-Large**	
Method	**Recall@20**	**NDCG@20**	**Recall@20**	**NDCG@20**	**Recall@20**	**NDCG@20**
GTN	0.0679*	0.0554*	0.0994*	0.0474*	OOM	OOM
LTGNN	**0.0681***	**0.0562***	**0.1079***	**0.0510***	**0.0294**	**0.0259**

Table 4.4: Running time comparison results with extra baseline GTN. Results marked with (*) are obtained in GTN's official settings.

Dataset	**Yelp2018**	**Alibaba-iFashion**	**Amazon-Large**
Method	**Running Time**	**Running Time**	**Running Time**
GTN	97.8s*	99.6s*	OOM
LTGNN	**24.5s***	**22.4s***	**705.9s**

Extra Comparison with GTN. Despite numerous works being orthogonal to our contributions and not tackling the scalability issue, as highlighted in Section 4.1, we remain receptive to comparing our method with more recent, accuracy-focused baselines in the field. We compare our proposed LTGNN with GTN [27], one of the latest GNN backbones for recommendations, and the results on recommendation performance are presented in Table. 4.3 and Table. 4.4.

To ensure a fair comparison, we closely followed the official settings of GTN [3], which differ from our evaluation settings detailed in Section 4.1 in embedding size and BPR batch size. We mark the results that are obtained in GTN's official settings with (*). As GTN is not designed for large-scale recommendations, GTN ran out of memory on the large-scale Amazon-Large dataset, which can be efficiently handled by our LTGNN. From the experiment result, we find that our proposed LTGNN shows advantages in both recommendation performance and scalability, even in comparison with one of the latest accuracy-driven GNN backbones for recommendations.

- Owing to the enhanced long-range expressiveness of our implicit modeling approach, the recommendation performance of LTGNN is on par with recent state-of-the-art GNN backbones such as GTN, even with only one propagation layer, as illustrated in Table. 4.3.

- Regarding the efficiency, LTGNN consistently demonstrates markedly superior training efficiency compared to GTN, as shown in Table. 4.4. For instance, GTN faces challenges, encountering out-of-memory issues when applied to large-scale datasets such as Amazon-Large. These findings unmistakably underscore the scalability advantage inherent in our model design, enabling it to operate seamlessly on large-scale recommendation datasets.

[3] `https://github.com/wenqifan03/GTN-SIGIR2022`

Table 4.5: The comparison of running time on three datasets.

Dataset		**Yelp2018**	**Alibaba-iFashion**	**Amazon-Large**
Method	**# Layer**	**Runnning Time**	**Runnning Time**	**Running Time**
LightGCN	$L = 3$	52.83s	51.4s	2999.35s
LightGCN-NS		46.09s	51.70s	4291.37s
LightGCN-VR		53.15s	59.79s	4849.59s
LightGCN-GAS		23.22s	26.576s	932.03s
LightGCN	$L = 2$	30.92s	30.78s	2061.75s
LightGCN-NS		37.17s	26.89s	1305.25s
LightGCN-VR		38.77s	30.33s	1545.34s
LightGCN-GAS		22.92s	25.04s	903.78s
LightGCN	$L = 1$	16.95s	18.02s	1117.51s
LightGCN-NS		13.90s	12.74s	684.84s
LightGCN-VR		15.52s	13.92s	870.82s
LightGCN-GAS		14.53s	13.35s	729.22s
MF	-	4.31s	4.60s	127.24s
LTGNN	$L = 1$	14.72s	13.68s	705.91s

4.3 Efficiency Analysis

To verify the scalability of LTGNN, we provide efficiency analysis in comparison with MF, LightGCN, and scalable variants of LightGCN with different layers on all three datasets: Yelp2018, Alibaba-iFasion, and Amazon-Large. From the running time shown in Table 4.5, we draw the following conclusions:

- Our proposed single-layer LTGNN achieves comparable running time with one-layer LightGCN with sampling, and outperforms the original LightGCN. This is consistent with our complexity analysis in Section 3.2. Moreover, LTGNN is faster than one-layer LightGCN with variance reduction, owing to our improved and efficient variance reduction (EVR) techniques. Although LTGNN is not substantially more efficient than some of the one-layer GNNs, it has much better recommendation performance, as shown in Figure 4.1.

- LTGNN demonstrates significantly improved computational efficiency compared to baseline models with more than one layer. When combined with the results from Table 4.2, it becomes evident that LTGNN can maintain high performance while achieving a substantial enhancement in computational efficiency.

- While the running time of LTGNN is a few times longer than that of Matrix Factorization (MF) due to their constant factor difference in the complexity analysis (Table. 3.1), it's important to note that LTGNN already exhibits a nice and *similar scaling behavior as MF.* This supports the better scalability of LTGNN in large-scale recommendations compared with other GNN-based methods.

- An interesting observation is that on large-scale datasets, full-graph LightGCN surpasses LightGCN with neighbor sampling on efficiency. This is mainly because of the high CPU overhead of random sampling, which limits the utilization of GPUs.

To further understand the efficiency of each component in LTGNN, we also conducted a fine-grained efficiency analysis on our proposed model.

Detailed Efficiency Analysis. As discussed in Section 3.2, LTGNN has linear time complexity with respect to the number of edges $|\mathcal{E}|$ in the user-item interaction graph, which is comparable to MF. However, as shown in Table. 4.5, the running time of LTGNN is worse than MF on the large-scale Amazon-Large dataset, which seems to contradict our complexity analysis. To understand this phenomenon and demonstrate LTGNN's scalability advantage, we perform a comprehensive efficiency analysis of LTGNN and MF, which identifies the main source of overhead and suggests potential enhancements.

From the results presented in Table. 4.6, we have the following observations:

Table 4.6: Detailed efficiency comparison of LTGNN with MF.

Model	**Stage**	**Yelp2018** $\lvert\mathcal{E}\rvert = 1.56m$	**Alibaba-iFashion** $\lvert\mathcal{E}\rvert = 1.61m$	**Amazon-Large** $\lvert\mathcal{E}\rvert = 15.24m$
MF	Neg. Samp.	0.12s	0.21s	1.23s
	Training	<u>4.19s</u>	<u>4.39s</u>	<u>126.01s</u>
	Total	4.31s	4.60s	127.24s
LTGNN	Neg. Samp.	0.12s	0.21s	1.23s
	Neigh. Samp.	7.98s	6.48s	512.55s
	Training	<u>6.62s</u>	<u>6.99s</u>	<u>192.13s</u>
	Memory Access	<0.005s	<0.005s	<0.005s
	Total	14.72s	13.68s	705.91s

- For both MF and LTGNN, the computation cost of negative sampling is negligible, which has a minimal impact on the total running time.
- For LTGNN, excluding the neighbor sampling time, the model training time is linear in the number of edges $|\mathcal{E}|$, which is similar to MF. This indicates LTGNN's high scalability in large-scale and industrial recommendation scenarios.
- Besides, the computational overhead incurred by maintaining and updating the memory spaces for variance reduction is insignificant, which validates the high efficiency of our proposed EVR approach.
- For LTGNN, the main source of extra computational overhead is the neighbor sampling process for the target nodes, which accounts for more than 50% of the total running time, preventing LTGNN from achieving a perfect linear scaling behavior. This is an implementation issue that can be improved by better engineering solutions.

To further enhance the efficiency of LTGNN, we can adopt some common engineering techniques. For example, we can follow the importance sampling approach in PinSAGE [119], which assigns a fixed neighborhood for each node in the interaction graph, avoiding the expensive random sampling process. We plan to leave this as a future work.

4.4 Ablation Study

In this section, we provide extensive ablation studies to evaluate the effectiveness of different parts in our proposed framework. We also conduct experiments on the effect of hyperparameter α and different variance-reduction designs.

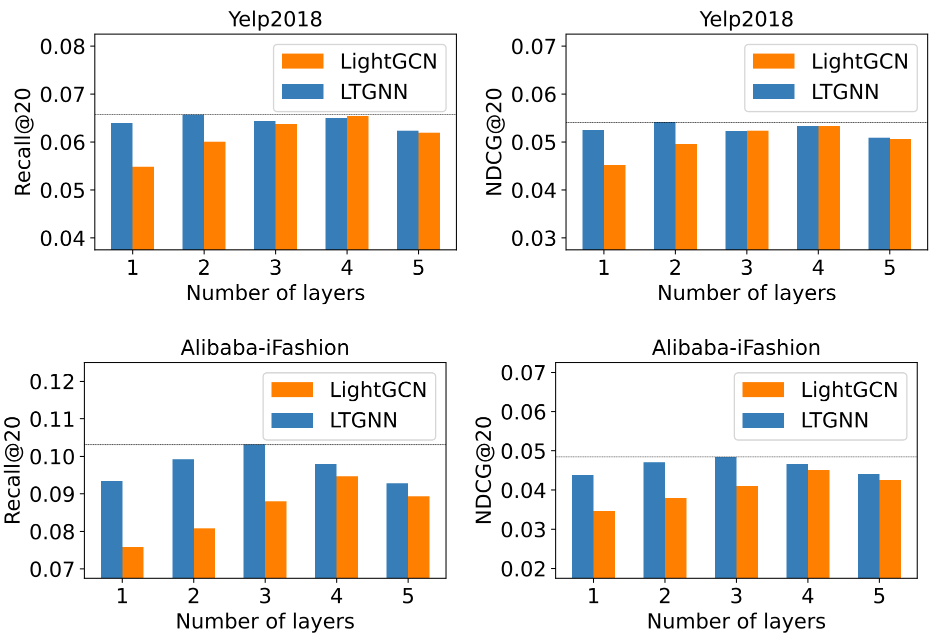

Figure 4.1: Performance comparison between LTGNN and LightGCN using different layers on Yelp2018 and Alibaba-iFashion.

Effectivenss of implicit graph modeling. We conduct an ablation study to show the effect of embedding propagation layers and long-range collaborative signals. Particularly, we use the same setting for LightGCN and LTGNN and change the number

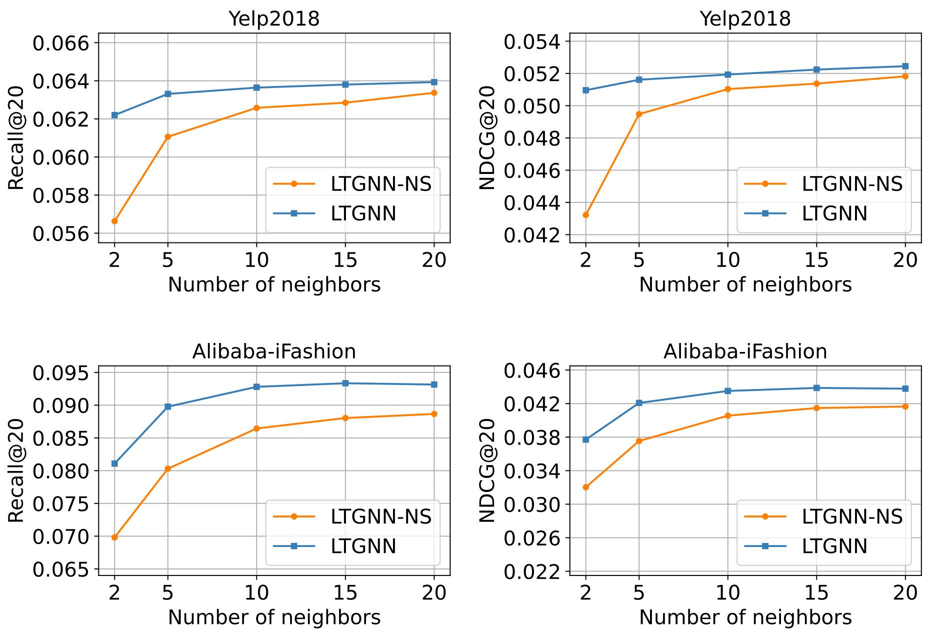

Figure 4.2: Performance of a 1-layer LTGNN w.r.t different numbers of sampled neighbors on Yelp2018 and Alibaba-iFashion.

of propagation layers L. As illustrated in Figure 4.1, we have two key observations: 1) LTGNN with only 1 or 2 propagation layers can reach better performance in comparison with LightGCN with more than 3 layers, which demonstrates the strong long-range modeling capability of our proposed model; 2) Adding more propagation layers into LTGNN will not significantly improve its performance, which means $L = 1$ or $L = 2$ are the best choices for LTGNN to balance its performance and scalability.

Effectiveness of efficient variance reduction. In this study, we aim to demonstrate the effectiveness of our proposed EVR algorithm by showing the impact of the number of neighbors on recommendation performance. As shown in Figure 4.2, LTGNN with efficient variance reduction consistently outperforms its vanilla neighbor sampling variant (i.e., LTGNN-NS) regardless of the number of neighbors, illustrating its effect in reducing the large approximation error caused by neighbor sampling. Notably, the recommendation performance of LTGNN with efficient variance reduction remains stable, even under extreme conditions, such as sampling only 2 neighbors

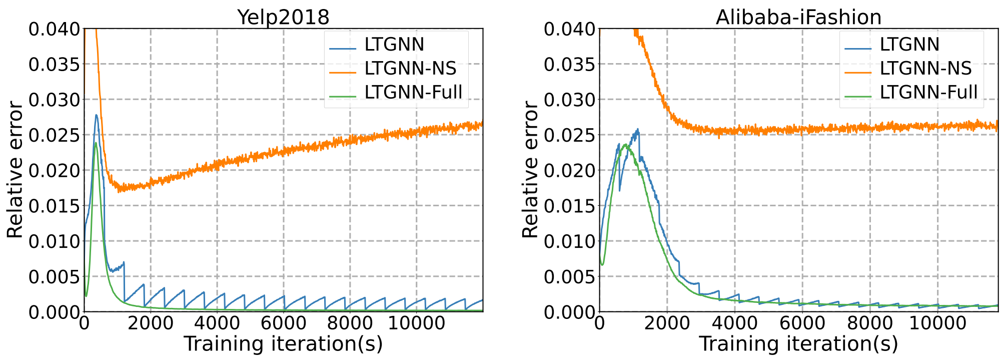

Figure 4.3: The relative error between the model output $\boldsymbol{E}^k_{out}$ and the exact PPNP propagation result $\boldsymbol{E}^k_{PPNP}$.

per target node. This is much smaller than the average node degree in each dataset (e.g., 10 for Alibaba-iFashion and 46 for Yelp2018, as shown in Table 4.1). This highlights the potential of our proposed LTGNN for large-scale recommendations, as a GNN with only one propagation layer and two random neighbors can achieve high efficiency compared to previous designs that rely on numerous layers and neighbors.

Numerical Analysis. In this experiment, we evaluate the long-range modeling ability by computing the PPNP embedding propagation result $\boldsymbol{E}^k_{PPNP}$ for target nodes and measuring the relative error between the model output $\boldsymbol{E}^k_L$ and the PPNP result, defined as $||\boldsymbol{E}^k_{out} - \boldsymbol{E}^k_{PPNP}||_F / ||\boldsymbol{E}^k_{PPNP}||_F$. We use $L = 1$ for LTGNN and its two inferior variants without efficient variance reduction: LTGNN-NS, which uses random neighbor sampling, and LTGNN-Full, which performs exact full neighbor aggregation. From Figure 4.3, we have two observations: 1) On both datasets, LTGNN's output converges to the PPNP result after approximately 4000 training iterations (i.e., fewer than 10 epochs), showing that LTGNN can capture long-range dependencies in user-item graphs with just one propagation layer; 2) Comparing LTGNN with its variants reveals that neighbor sampling without variance reduction significantly reduces LTGNN's modeling ability, while LTGNN with efficient variance reduction shows a convergence curve similar to that of LTGNN-Full, highlighting the effectiveness of our variance reduction method.

Effect of Hyperparameter α. Similar to APPNP [31], the teleport factor α in LTGNN controls the trade-off between capturing long-range dependencies and graph localities, directly impacting recommendation performance. As shown in Fig. 4.4, we find that LTGNN is rather robust to α, as the performance does not drop sharply in $\alpha \in [0.35, 0.55]$. In practice, we can set $\alpha = 0.5$, which is the midpoint of capturing long-range dependency and preserving the local structure expressiveness, and conduct a rough grid search to find the optimal setting of α.

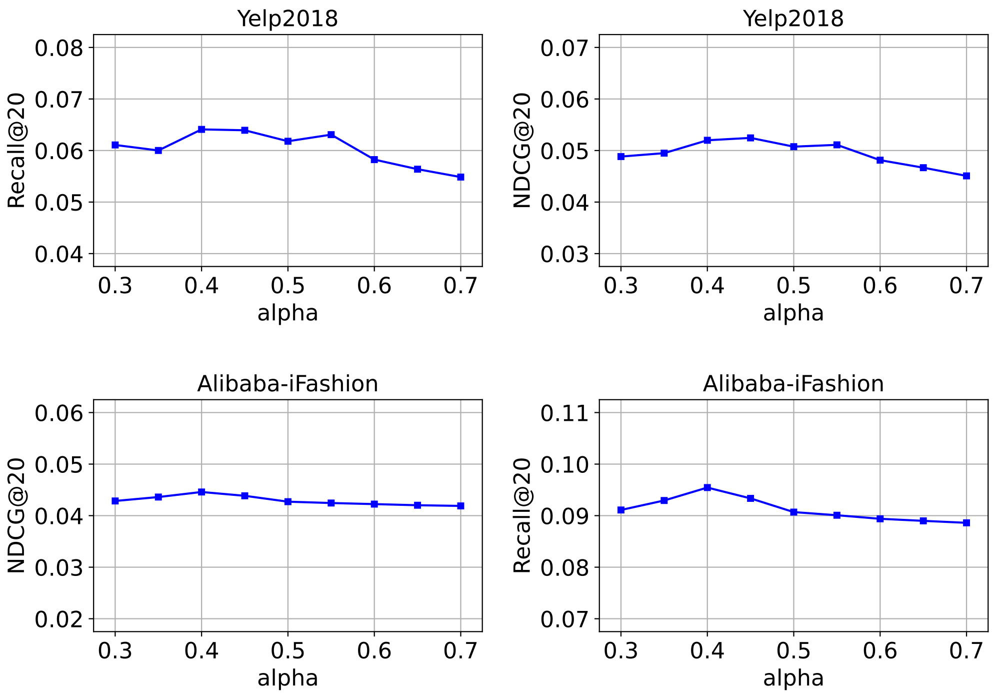

Figure 4.4: The effect of hyper-parameter α under Recall@20 and NDCG@20 metrics.

Effect of Different Variance Reduction Designs. We developed an efficient variance reduction (EVR) mechanism in Section 3.2, which can be applied to both forward and backward computations. To evaluate the impact of different variance reduction designs on the recommendation performance, we compare four variants of LTGNN, namely *LTGNN-NS*, *LTGNN-FVR*, *LTGNN-BVR*, and *LTGNN-BiVR*. LTGNN-NS is the baseline model that uses implicit modeling and vanilla neighbor sampling, while LTGNN-FVR, LTGNN-BVR, and LTGNN-BiVR use forward variance reduction, backward variance reduction, and bidirectional variance reduction, respectively.

Table 4.7: Ablation study on different variance reduction designs.

Dataset	**Yelp2018**		**Alibaba-iFashion**	
Method	**Recall@20**	**NDCG@20**	**Recall@20**	**NDCG@20**
LTGNN-NS	0.06285	0.05137	0.08804	0.04147
LTGNN-BVR	0.06309	0.05160	0.08878	0.04185
LTGNN-BiVR	<u>0.06321</u>	<u>0.05194</u>	<u>0.09241</u>	<u>0.04338</u>
LTGNN-FVR	**0.6380**	**0.05224**	**0.09335**	**0.04387**

As can be seen from Table. 4.7, LTGNN variants with variance reduction outperform LTGNN-NS without variance reduction, demonstrating the effectiveness of our proposed EVR approach. Moreover, we observe that forward variance reduction (FVR) alone is sufficient to achieve satisfactory recommendation performance. Surprisingly, bidirectional variance reduction does not outperform forward variance reduction. We intend to investigate the reason behind this phenomenon and explore the potential of bidirectional variance reduction (BiVR) in our future work.

Chapter 5

Conclusions and Suggestions for Future Research

5.1 Conclusions

Scalability is a major challenge for GNN-based recommender systems, as they often require many computational resources to handle large-scale user-item interaction data. While existing methods aimed at enhancing the scalability of GNNs and accelerating GNN-based collaborative filtering models offer some relief for the scalability problem, these prior works hardly achieve linear complexity, rendering them less scalable compared to simpler CF models like MF and NCF.

This thesis identifies two principal obstacles to attaining linear complexity in GNN-based recommender systems: the trade-off between expressive power and the number of layers, and the balance between random error and the number of sampled neighbors. To overcome these challenges, we introduce two significant advancements to pave our way to linear complexity. These include implicit graph modeling and variance-reduced neighbor sampling, which together enable the capture of long-range collaborative signals while maintaining the desirable linear complexity.

Based on these non-trivial designs, our proposed model, LTGNN, requires only a single propagation layer and a limited set of one-hop neighbors, thereby achieving a linear computational complexity with respect to the number of edges. This shows that LTGNN holds considerable promise for deployment in industrial-scale recommendation systems. Extensive experiments on three real-world datasets are conducted to demonstrate the effectiveness and efficiency of our proposed model.

5.2 Suggestions for Future Research

For future work, one of the most straightforward extensions of LTGNN is to incorporate rich side information for users and items, going beyond the collaborative filtering setting used in the evaluations in this thesis. This can be achieved by replacing the embedding table of LTGNN with a trainable side information encoder, which would make LTGNN adaptable for more complex and real-world recommendation scenarios, such as multimedia recommendations.

Another avenue for future research is to tackle the complexity of real-world recommendation graphs. While this thesis primarily focuses on recommendations within a user-item interaction graph featuring a single type of interaction, it is natural to extend LTGNN to accommodate more complicated recommendation graphs. These may encompass multiple types of relationships, including social connections and user-item interactions characterized by diverse user behaviors, culminating in a complex and heterogeneous environment. A potential approach in this new context is to first establish a fixed-point equation (similar to the PPNP fixed point) that addresses graph heterogeneity, and then solve the equilibrium state using the solvers proposed in this thesis.

We are also open to broaden the application of LTGNN to more practical and industrial recommendation settings, such as Click-Through Rate (CTR) prediction and

Conversion Rate (CVR) prediction. These domains often present distinct training objectives, larger-scale and sparser datasets, and rich user and item side information. In these contexts, we plan to explore the integration of LTGNN with existing industrial systems, adapting its architecture to meet the specific demands of these applications while maintaining its computational efficiency and predictive accuracy.

References

[1] Emily Alsentzer, Samuel Finlayson, Michelle Li, and Marinka Zitnik. Subgraph neural networks. *Advances in Neural Information Processing Systems*, 33:8017–8029, 2020.

[2] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

[3] Ting Bai, Ji-Rong Wen, Jun Zhang, and Wayne Xin Zhao. A neural collaborative filtering model with interaction-based neighborhood. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1979–1982, 2017.

[4] Muhammed Fatih Balin and Ümit Çatalyürek. Layer-neighbor sampling—defusing neighborhood explosion in gnns. *Advances in Neural Information Processing Systems*, 36:25819–25836, 2023.

[5] Stefan Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fundamenta mathematicae*, 3(1):133–181, 1922.

[6] Robert M Bell and Yehuda Koren. Scalable collaborative filtering with jointly

derived neighborhood interpolation weights. In *Seventh IEEE International conference on data mining (ICDM 2007)*, pages 43–52. IEEE, 2007.

[7] James Bennett, Stan Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York, 2007.

[8] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. Latent cross: Making use of context in recurrent recommender systems. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 46–54, 2018.

[9] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2464–2473, 2020.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] Hao Chen, Yuanchen Bei, Qijie Shen, Yue Xu, Sheng Zhou, Wenbing Huang, Feiran Huang, Senzhang Wang, and Xiao Huang. Macro graph neural networks for online billion-scale recommender systems. In *Proceedings of the ACM on Web Conference 2024*, pages 3598–3608, 2024.

[12] Jianfei Chen, Jun Zhu, and Le Song. Stochastic training of graph convolutional networks with variance reduction. In *International Conference on Machine Learning*, pages 942–950. PMLR, 2018.

[13] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph

convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018.

[14] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 335–344, 2017.

[15] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 27–34, 2020.

[16] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[17] Andrzej Cichocki, Hyekyoung Lee, Yong-Deok Kim, and Seungjin Choi. Non-negative matrix factorization with α-divergence. *Pattern Recognition Letters*, 29(9):1433–1440, 2008.

[18] Weilin Cong, Rana Forsati, Mahmut Kandemir, and Mehrdad Mahdavi. Minimal variance sampling with provable guarantees for fast training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1393–1403, 2020.

[19] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[20] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296, 2010.

[21] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[23] Xingzhong Du, Hongzhi Yin, Ling Chen, Yang Wang, Yi Yang, and Xiaofang Zhou. Personalized video recommendation using rich contents from videos. *IEEE Transactions on Knowledge and Data Engineering*, 32(3):492–505, 2018.

[24] Keyu Duan, Zirui Liu, Peihao Wang, Wenqing Zheng, Kaixiong Zhou, Tianlong Chen, Xia Hu, and Zhangyang Wang. A comprehensive study on large-scale graph training: Benchmarking and rethinking. *Advances in Neural Information Processing Systems*, 35:5376–5389, 2022.

[25] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. Implicit deep learning. *SIAM Journal on Mathematics of Data Science*, 3(3):930–958, 2021.

[26] Ali Mamdouh Elkahky, Yang Song, and Xiaodong He. A multi-view deep learning approach for cross domain user modeling in recommendation systems. In *Proceedings of the 24th international conference on world wide web*, pages 278–288, 2015.

[27] Wenqi Fan, Xiaorui Liu, Wei Jin, Xiangyu Zhao, Jiliang Tang, and Qing Li. Graph trend filtering networks for recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 112–121, 2022.

[28] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426, 2019.

[29] Wenqi Fan, Yao Ma, Qing Li, Jianping Wang, Guoyong Cai, Jiliang Tang, and Dawei Yin. A graph neural network framework for social recommendations. *IEEE Transactions on Knowledge and Data Engineering*, 34(5):2033–2047, 2020.

[30] Matthias Fey, Jan E Lenssen, Frank Weichert, and Jure Leskovec. Gnnautoscale: Scalable and expressive graph neural networks via historical embeddings. In *International conference on machine learning*, pages 3294–3304. PMLR, 2021.

[31] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2019.

[32] Sahin Cem Geyik, Qi Guo, Bo Hu, Cagri Ozcaglar, Ketan Thakkar, Xianren Wu, and Krishnaram Kenthapadi. Talent search and recommendation systems at linkedin: Practical challenges and lessons learned. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1353–1354, 2018.

[33] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[34] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[35] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. *Advances in Neural Information Processing Systems*, 33:11984–11995, 2020.

[36] Linxin Guo, Yaochen Zhu, Min Gao, Yinghui Tao, Junliang Yu, and Chen Chen. Consistency and discrepancy-based contrastive tripartite graph learning for recommendations. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 944–955, 2024.

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

[38] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[40] Li He, Xianzhi Wang, Dingxian Wang, Haoyuan Zou, Hongzhi Yin, and Guandong Xu. Simplifying graph-based collaborative filtering for recommendation. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 60–68, 2023.

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

ommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[42] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. Outer product-based neural collaborative filtering. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 2227–2233. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[43] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 30(12):2354–2366, 2018.

[44] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[45] Roger A Horn and Charles R Johnson. *Matrix analysis.* Cambridge university press, 2012.

[46] Patrik O Hoyer. Non-negative sparse coding. In *Proceedings of the 12th IEEE workshop on neural networks for signal processing*, pages 557–565. IEEE, 2002.

[47] Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(9), 2004.

[48] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE international conference on data mining*, pages 263–272. Ieee, 2008.

[49] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations*, 2021.

[50] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. Mixgcf: An improved training method for graph neural network-based recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 665–674, 2021.

[51] Aleksandar Ilic and Maja Kabiljo. Recommending items to more than a billion people. Technical report, Facebook, 2015.

[52] Wei Jin, Haitao Mao, Zheng Li, Haoming Jiang, Chen Luo, Hongzhi Wen, Haoyu Han, Hanqing Lu, Zhengyang Wang, Ruirui Li, et al. Amazon-m2: A multilingual multi-locale shopping session dataset for recommendation and text generation. *Advances in Neural Information Processing Systems*, 36, 2024.

[53] Santosh Kabbur, Xia Ning, and George Karypis. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 659–667, 2013.

[54] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *The 3rd International Conference on Learning Representations*, 2015.

[55] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[56] Raul Kompass. A generalized divergence measure for nonnegative matrix factorization. *Neural computation*, 19(3):780–791, 2007.

[57] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434, 2008.

[58] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

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

[60] Daniel Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13, 2000.

[61] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *nature*, 401(6755):788–791, 1999.

[62] Mingjie Li, Yifei Wang, Yisen Wang, and Zhouchen Lin. Unbiased stochastic proximal solver for graph neural networks with equilibrium states. In *The Eleventh International Conference on Learning Representations*, 2023.

[63] Zhao Li, Xin Shen, Yuhang Jiao, Xuming Pan, Pengcheng Zou, Xianling Meng, Chengwei Yao, and Jiajun Bu. Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1677–1688. IEEE, 2020.

[64] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. Variational autoencoders for collaborative filtering. In *Proceedings of the 2018 world wide web conference*, pages 689–698, 2018.

[65] Xurong Liang, Tong Chen, Lizhen Cui, Yang Wang, Meng Wang, and Hongzhi Yin. Lightweight embeddings for graph collaborative filtering. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1296–1306, 2024.

[66] Zihan Lin, Changxin Tian, Yupeng Hou, and Wayne Xin Zhao. Improving graph collaborative filtering with neighborhood-enriched contrastive learning. In *Proceedings of the ACM Web Conference 2022*, pages 2320–2329, 2022.

[67] Chengyi Liu, Wenqi Fan, Yunqing Liu, Jiatong Li, Hang Li, Hui Liu, Jiliang Tang, and Qing Li. Generative diffusion models on graphs: methods and applications. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pages 6702–6711, 2023.

[68] Juncheng Liu, Bryan Hooi, Kenji Kawaguchi, Yiwei Wang, Chaosheng Dong, and Xiaokui Xiao. Scalable and effective implicit graph neural networks on large graphs. In *The Twelfth International Conference on Learning Representations*, 2024.

[69] Jianxin Ma, Chang Zhou, Peng Cui, Hongxia Yang, and Wenwu Zhu. Learning disentangled representations for recommendation. *Advances in neural information processing systems*, 32, 2019.

[70] Kelong Mao, Jieming Zhu, Jinpeng Wang, Quanyu Dai, Zhenhua Dong, Xi Xiao, and Xiuqiang He. Simplex: A simple and strong baseline for collaborative filtering. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1243–1252, 2021.

[71] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. Ultragcn: ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1253–1262, 2021.

[72] Andriy Mnih and Russ R Salakhutdinov. Probabilistic matrix factorization. *Advances in neural information processing systems*, 20, 2007.

[73] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. Technical report, Google DeepMind, 2016.

[74] Lawrence Page. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University, 1998.

[75] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[76] Shaowen Peng, Kazunari Sugiyama, and Tsunenori Mine. Svd-gcn: A simplified graph convolution paradigm for recommendation. In *Proceedings of the 31st ACM international conference on information & knowledge management*, pages 1625–1634, 2022.

[77] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023.

[78] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461, 2009.

[79] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pages 240–248, 2020.

[80] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, 1994.

[81] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factor-

ization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887, 2008.

[82] Badrul Sarwar, George Karypis, Joseph Konstan, and John T. Riedl. Application of dimensionality reduction in recommender system - a case study. Technical report, University of Minnesota, 2000.

[83] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

[84] J Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166, 1999.

[85] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The semantic web: 15th international conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, proceedings 15*, pages 593–607. Springer, 2018.

[86] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *Proceedings of the 24th international conference on World Wide Web*, pages 111–112, 2015.

[87] Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, 1995.

[88] Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I Nikolenko. Recvae: A new variational autoencoder for top-n recommendations with implicit feedback. In *Proceedings of the 13th international conference on web search and data mining*, pages 528–536, 2020.

[89] Zhihao Shi, Xize Liang, and Jie Wang. LMC: Fast training of GNNs via subgraph sampling with provable convergence. In *The Eleventh International Conference on Learning Representations*, 2023.

[90] Brent Smith and Greg Linden. Two decades of recommender systems at amazon. com. *Ieee internet computing*, 21(3):12–18, 2017.

[91] Daniil Sorokin and Iryna Gurevych. Modeling semantics with gated graph neural networks for knowledge base question answering. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3306–3317, 2018.

[92] Jiakai Tang, Sunhao Dai, Zexu Sun, Xu Chen, Jun Xu, Wenhui Yu, Lantao Hu, Peng Jiang, and Han Li. Towards robust recommendation via decision boundary-aware graph contrastive learning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2854–2865, 2024.

[93] Jiliang Tang, Xia Hu, and Huan Liu. Social recommendation: a review. *Social Network Analysis and Mining*, 3:1113–1133, 2013.

[94] Shang-Hua Teng et al. Scalable algorithms for data and network analysis. *Foundations and Trends® in Theoretical Computer Science*, 12(1–2):1–274, 2016.

[95] Rianne Van Den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017. Presented at KDD 2018 Deep Learning Day.

[96] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020.

[97] Nikolaos Vasiloglou, Alexander G Gray, and David V Anderson. Non-negative matrix factorization, convexity and isometry. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 673–684. SIAM, 2009.

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

[99] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 839–848, 2018.

[100] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 950–958, 2019.

[101] Xiang Wang, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. Item silk road: Recommending items from information domains to social users. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 185–194, 2017.

[102] 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.

[103] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. Disentangled graph collaborative filtering. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pages 1001–1010, 2020.

[104] Yu Wang, Yuying Zhao, Yi Zhang, and Tyler Derr. Collaboration-aware graph convolutional network for recommender systems. In *Proceedings of the ACM Web Conference 2023*, pages 91–101, 2023.

[105] Yu-Xiong Wang and Yu-Jin Zhang. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on knowledge and data engineering*, 25(6):1336–1353, 2012.

[106] Joseph L Watson, David Juergens, Nathaniel R Bennett, Brian L Trippe, Jason Yim, Helen E Eisenach, Woody Ahern, Andrew J Borst, Robert J Ragotte, Lukas F Milles, et al. De novo design of protein structure and function with rfdiffusion. *Nature*, 620(7976):1089–1100, 2023.

[107] Yinwei Wei, Wenqi Liu, Fan Liu, Xiang Wang, Liqiang Nie, and Tat-Seng Chua. Lightgt: A light graph transformer for multimedia recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1508–1517, 2023.

[108] Yinwei Wei, Xiang Wang, Liqiang Nie, Xiangnan He, and Tat-Seng Chua. Graph-refined convolutional network for multimedia recommendation with implicit feedback. In *Proceedings of the 28th ACM international conference on multimedia*, pages 3541–3549, 2020.

[109] Chuhan Wu, Fangzhao Wu, Tao Qi, and Yongfeng Huang. Empowering news recommendation with pre-trained language models. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 1652–1656, 2021.

[110] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[111] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. Self-supervised graph learning for recommendation. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pages 726–735, 2021.

[112] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 35(5):4425–4445, 2022.

[113] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys*, 55(5):1–37, 2022.

[114] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the ninth ACM international conference on web search and data mining*, pages 153–162, 2016.

[115] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[116] Xiangjin Xie, Yuxin Chen, Ruipeng Wang, Xianli Zhang, Shilei Cao, Kai Ouyang, Zihan Zhang, Hai-Tao Zheng, Buyue Qian, Hansen Zheng, et al. Mixdec sampling: A soft link-based sampling method of graph neural network for recommendation. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 598–607. IEEE, 2022.

[117] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. Deep matrix factorization models for recommender systems. In *IJCAI*, volume 17, pages 3203–3209. Melbourne, Australia, 2017.

[118] Rui Xue, Haoyu Han, MohamadAli Torkamani, Jian Pei, and Xiaorui Liu. Lazygnn: Large-scale graph neural networks via lazy propagation. In *International Conference on Machine Learning*, pages 38926–38937. PMLR, 2023.

[119] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[120] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *International conference on machine learning*, pages 7134–7143. PMLR, 2019.

[121] Haiyang Yu, Limei Wang, Bokun Wang, Meng Liu, Tianbao Yang, and Shuiwang Ji. Graphfm: Improving large-scale gnn training via feature momentum. In *International Conference on Machine Learning*, pages 25684–25701. PMLR, 2022.

[122] Junliang Yu, Xin Xia, Tong Chen, Lizhen Cui, Nguyen Quoc Viet Hung, and Hongzhi Yin. Xsimgcl: Towards extremely simple graph contrastive learning for recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 2023.

[123] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pages 1294–1303, 2022.

[124] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020.

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

[126] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. Graph attention multi-layer perceptron. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4560–4570, 2022.

[127] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[128] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1079–1088, 2018.

[129] Xiaojin Zhu. *Semi-supervised learning with graphs.* Carnegie Mellon University, 2005.

[130] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in neural information processing systems*, 32, 2019.