Lighter-X: An Efficient and Plug-and-play Strategy for Graph-based Recommendation through Decoupled Propagation

Yanping Zheng
Renmin University of China
zhengyanping@ruc.edu.cn

Zhewei Wei*
Renmin University of China
zhewei@ruc.edu.cn

Frank de Hoog
Data 61, CSIRO
Frank.Dehoog@data61.csiro.au

Xu Chen
Hongteng Xu
Renmin University of China
xu.chen@ruc.edu.cn
hongtengxu@ruc.edu.cn

Yuhang Ye
Jiadeng Huang
Huawei Poisson Lab
yuhang.ye@huawei.com
huangjiadeng96@sina.com

ABSTRACT

Graph Neural Networks (GNNs) have demonstrated remarkable effectiveness in recommendation systems. However, conventional graph-based recommenders, such as LightGCN, require maintaining embeddings of size d for each node, resulting in a parameter complexity of $O(n\times d)$, where n represents the total number of users and items. This scaling pattern poses significant challenges for deployment on large-scale graphs encountered in real-world applications. To address this scalability limitation, we propose **Lighter-X**, an efficient and modular framework that can be seamlessly integrated with existing GNN-based recommender architectures. Our approach substantially reduces both parameter size and computational complexity while preserving the theoretical guarantees and empirical performance of the base models, thereby enabling practical deployment at scale. Specifically, we analyze the original structure and inherent redundancy in their parameters, identifying opportunities for optimization. Based on this insight, we propose an efficient compression scheme for the sparse adjacency structure and high-dimensional embedding matrices, achieving a parameter complexity of $O(h \times d)$, where $h \ll n$. Furthermore, the model is optimized through a decoupled framework, reducing computational complexity during the training process and enhancing scalability. Extensive experiments demonstrate that Lighter-X achieves comparable performance to baseline models with significantly fewer parameters. In particular, on large-scale interaction graphs with millions of edges, we are able to attain even better results with only 1% of the parameter over LightGCN.

PVLDB Reference Format:
Yanping Zheng, Zhewei Wei, Frank de Hoog, Xu Chen, Hongteng Xu, Yuhang Ye, and Jiadeng Huang. Lighter-X: An Efficient and Plug-and-play Strategy for Graph-based Recommendation through Decoupled Propagation. PVLDB, 18(11): 3721 - 3729, 2025.
doi:10.14778/3749646.3749649

*Zhewei Wei is the corresponding author. The work was partially done at Gaoling School of Artificial Intelligence, Beijing Key Laboratory of Research on Large Models and Intelligent Governance, MOE Key Lab of Data Engineering and Knowledge Engineering, Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE, and Pazhou Laboratory (Huangpu), Guangzhou, Guangdong 510555, China.

Proceedings of the VLDB Endowment, Vol. 18, No. 11 ISSN 2150-8097.
doi:10.14778/3749646.3749649

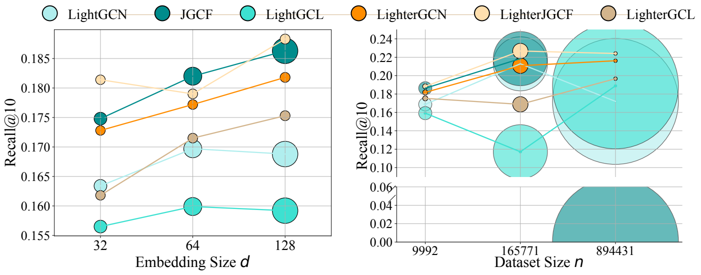

Figure 1: Performance vs. Training Parameters: Circle sizes represent parameter counts. Baseline models' parameters scale proportionally with embedding size (d) and dataset size (n), while Lighter-X achieves higher accuracy with more compact parameter sizes.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/zheng-yp/Lighter-X.

1 INTRODUCTION

Recent studies have shown that recommender systems based on Graph Neural Networks (GNNs) outperform traditional collaborative filtering methods [31]. Since much of the data in recommender systems can be naturally represented as graphs, GNNs leverage their powerful representation learning capabilities to capture complex relationships, thereby enhancing recommendation accuracy. For example, modeling user-item interactions as a bipartite graph allows for better exploitation of collaborative filtering information through neighbor convolution. By stacking more convolutional layers, the users and items with longer distances can be associated and share similar propagated gradients in the optimization process [7]. Despite effectiveness, graph-based recommender models usually contain a large number of parameters and need complex convolutional operations, which hinders their application in real-world scenarios [1, 15]. This problem necessitates the studies of more efficient graph-based recommender models.

LightGCN [9] simplifies traditional graph-based models by retaining only the essential neighbor aggregation operation. However, it still contains a large number of training parameters, expressed as $n \times d$, where n is the total number of users and items, and d is the embedding size. As shown in Figure 1, LightGCN's parameter count grows dramatically with both embedding dimension d and dataset size n. Specifically, the left panel examines Recall@10 for varying embedding dimensions d on the MovieLens-1M dataset. Overall, increasing d leads to improved performance, but LightGCN [9] requires significantly more parameters. The right shows results for models with fixed embedding dimensions across three datasets of increasing size: MovieLens-1M (n=9,992), MovieLens-20M (n=165,771), and Alimama (n=894,431). Similarly, LightGCN's parameter scales proportionally with dataset size n.

Recent works have introduced polynomial-based filters [8] and Graph Contrastive Learning (GCL) [3, 34] to improve recommendation accuracy. However, these approaches rely on LightGCN [9] as their backbone network, thereby inheriting its scalability limitations when applied to large-scale datasets, as shown in Figure 1. Notably, JGCF [8] encountered an out-of-memory (OOM) error on the Alimama dataset. This raises an important question: **How can we design a lighter, more parameter-efficient framework while maintaining model performance?**

In this paper, we propose **Lighter-X**, a plug-and-play framework that can be seamlessly integrated into existing graph-based recommendation models to significantly reduce parameter cost. Motivated by the observation of inherent parameter redundancy in such models, we introduce a compression mechanism for both sparse graph structures and embedding matrices. As shown in Figure 1, Lighter-X models maintain stable model sizes regardless of embedding dimension d or dataset size n, achieving parameter efficiency and competitive performance. Our contributions can be summarized as follows:

- We introduce Lighter-X, which reduces parameter complexity to $O(h \times d)$, where $h \ll n$ corresponds to dataset sparsity.
- Employing the Lighter-X framework, we improve existing recommender models and construct *LighterGCN*, *LighterJGCF* and *LighterGCL*. Theoretical analysis shows that proposed models preserve the key properties of base models while significantly reducing parameter counts and computational complexity.
- We conduct extensive experiments on several datasets and demonstrated that the proposed method achieves comparable or even better results with significantly fewer parameters, leading to substantially faster training times.

2 BACKGROUND AND PRELIMINARY

A recommender system typically consists of a user set U, an item set I, and a user-item interaction matrix $\mathbf{R} \in \{0,1\}_{|U|\times|I|}$, where $\mathbf{R}_{ui} = 1$ indicates an interaction between user u and item i. Graph-based recommender models represent these interactions as a bipartite graph $G = (V, E)$, where the node set $V = U \cup I$ includes all users and items, and the edge set $E = \{(u,i) \mid \mathbf{R}_{ui} = 1, u \in U, i \in I\}$. The goal is to estimate user u's preference for item $i \in I$ using their learned representation $\boldsymbol{e}_u$ and $\boldsymbol{e}_i$, formulated as $\hat{\boldsymbol{y}}_{u,i} = \boldsymbol{e}_u^\top \boldsymbol{e}_i$.

2.1 Decoupled GNNs

GNNs are powerful tools for modeling graph data and have achieved impressive performance across various graph-related tasks. However, applying conventional GNNs such like GCN [11] to large-scale graphs is challenging due to the limitations of full-batch training. To improve scalability without compromising accuracy, several methods, including SGC [29], PPRGo [2], and AGP [24], decoupled feature propagation from the training process. In general, feature propagation is computed as:

$$\mathbf{Z} = \sum_{\ell=0}^{L} w_\ell \mathbf{Z}^{(\ell)} = \sum_{\ell=0}^{L} w_\ell \mathbf{P}^\ell \mathbf{X}, \tag{1}$$

where L is the number of layers, $\mathbf{P} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix, and w_ℓ denotes the importance of the ℓ-th layer. Each $\mathbf{Z}^{(\ell)}$ s recursively defined as $\mathbf{Z}^{(\ell)} = \mathbf{P}\mathbf{Z}^{(\ell-1)}$, with the initial representation $\mathbf{Z}^{(0)} = \mathbf{X}$, the input feature matrix (e.g., user attributes such as age, gender, or occupation). Typically, the feature propagation matrix $\mathbf{Z}$ can be precomputed and then used as input to a downstream model like a Multilayer Perceptron (MLP). In recommendation tasks, the goal is to learn node embeddings rather than prediction scores. With a single-layer MLP, the final embedding matrix is computed as $\mathbf{E} = \mathbf{Z}\mathbf{W}$, where $\mathbf{W}$ is the MLP weight matrix.

2.2 Graph-based Recommender Models

Graph-based recommender models learn powerful node embeddings by leveraging collaborative signals from high-order neighbors. NGCF [25] is built on the standard GCN [11] architecture. LightGCN [9] simplifies NGCF by removing the weight matrices and the activation function in each layer. Formally, the embedding calculation in LightGCN can be represented by:

$$\mathbf{E} = \frac{1}{L+1}\sum_{\ell=0}^{L} \mathbf{E}^{(\ell)} = \frac{1}{L+1}\sum_{\ell=0}^{L} \mathbf{P}^\ell \mathbf{E}^{(0)}, \tag{2}$$

where L is the number of layers, $\mathbf{E}^{(\ell)}$ is the embedding matrix at layer ℓ, and $\mathbf{E}^{(0)}$ is the initial embedding matrix, randomly initialized and used as the only learnable parameter. Each layer-wise embedding is computed recursively as $\mathbf{E}^{(\ell)} = \mathbf{P}\mathbf{E}^{(\ell-1)} = \mathbf{P}^\ell \mathbf{E}^{(0)}$. The repeated application of the propagation matrix $\mathbf{P}$ allows the model to capture multi-hop neighborhood information. Recent extensions introduce polynomial graph filters [8] and graph contrastive learning [3, 30, 34] to further boost performance.

Polynomial graph filters. Some works attribute the success of graph collaborative filtering to its effective implementation of low-pass filtering, and introduce polynomials to enable more flexible frequency responses [8, 16]. JGCF [8] utilizes Jacobi polynomial bases, denoted as $\mathbf{J}_\ell^{a,b}(x)$, to approximate graph signal filters, facilitating efficient frequency decomposition and signal filtration. The ℓ-th order Jacobi basis $\mathbf{J}_\ell^{a,b}(x)$ is parameterized by $a, b > -1$, which control the filter's response characteristics. This formulation enables separate modeling of low- and mid-frequency signals, whose effects are combined to form the final embeddings:

$$\mathbf{E} = \mathrm{concat}(\mathbf{E}_{low}, \mathbf{E}_{mid}), \quad \mathbf{E}_{low} = \frac{1}{L+1}\sum_{\ell=0}^{L} \mathbf{J}_\ell^{a,b}(\mathbf{P})\mathbf{E}^{(0)}. \tag{3}$$

The mid-frequency component is calculated as $\mathbf{E}_{mid} = \tanh(\beta \mathbf{E}^{(0)} - \mathbf{E}_{low}))$, where β is a weighting factor controlling the balance between low- and high-frequency information.

Graph contrastive learning. To address the issue of sparse information in recommender systems, recent studies have introduced contrastive learning to enhance performance [3, 30, 34]. The core idea is to modify the original graph structure to generate augmented representations. LightGCL [3] employs Singular Value Decomposition (SVD) to guide data augmentation. Specifically, SVD is applied to the interaction matrix $\mathbf{R}$, yielding $\mathbf{R} = \mathbf{U}\mathbf{Q}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{|U|\times|U|}$ and $\mathbf{V} \in \mathbb{R}^{|I|\times|I|}$ are orthogonal matrices, and $\mathbf{Q}$ is a diagonal matrix of singular values. Since principal components correspond to top-k singular values, LightGCL uses them to construct a perturbed interaction matrix $\hat{\mathbf{R}}$. The perturbed adjacency matrix $\hat{\mathbf{A}} = [[\mathbf{0}, \hat{\mathbf{R}}], [\hat{\mathbf{R}}^\top, \mathbf{0}]]$, which is then used in Equation 2 to compute the perturbed embedding:

$$\hat{\mathbf{E}} = \sum_{\ell=0}^{L} \hat{\mathbf{E}}^{(\ell)}, \quad \hat{\mathbf{E}}^{(\ell)} = \hat{\mathbf{P}} \cdot \mathbf{E}^{(\ell-1)}, \tag{4}$$

where $\hat{\mathbf{P}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ is the perturbed propagation matrix, $\hat{\mathbf{E}}^{(\ell)}$ refers to the perturbed embedding at layer ℓ, and $\hat{\mathbf{E}}^{(0)} = \mathbf{E}^{(0)}$.

Scalable methods. To improve the scalability of graph-based recommendation systems, several approaches have been proposed to balance efficiency and memory use. XGCN [19] is a library designed for GNN-based recommendations, incorporating optimized implementations and scaling strategies to process large datasets with low memory overhead. LTGNN [35] enhances propagation efficiency by adopting an implicit modeling approach inspired by PPNP and integrating a variance-reduced neighbor sampling strategy to further improve scalability and efficiency. GraphHash [32] focuses on parameter reduction by employing modularity-based bipartite graph clustering to compress the embedding table. This approach is orthogonal to our work, as Lighter-X improves parameter efficiency by optimizing the model's computational structure.

Simplified methods. Recently, some works have been proposed to optimize and simplify graph-based recommendation models. LightGODE [36] reduces training cost by modeling graph convolution as differential equations, removing graph operations during training and reintroducing them only for validation. However, its structure remains similar to LightGCN, with no reduction in parameters. Another line of work, such as SVD-GCN [15], reduce parameters via truncated SVD for low-rank embedding approximation. While effective, SVD incurs high time and memory costs on large-scale graphs, limiting scalability. In contrast, the proposed Lighter-X achieves both computational simplification and parameter compression.

3 INVESTIGATION OF GRAPH-BASED RECOMMENDATION MODELS

In this section, we analyze the connection between LightGCN [9] and decoupled GNN models, highlighting the reasons behind the large parameter sizes in graph-based recommendation models, using LightGCN as a representative example. We then demonstrate through experimental observations that this large parameter matrix is largely redundant.

Origins for Large Parameter Counts. LightGCN [9] simplifies NGCF [25] by removing feature transformations and nonlinear activations, relying solely on linear neighborhood aggregation to capture collaborative signals. It can be viewed as a simplified form of a decoupled GNN. While LightGCN is not fully decoupled, since

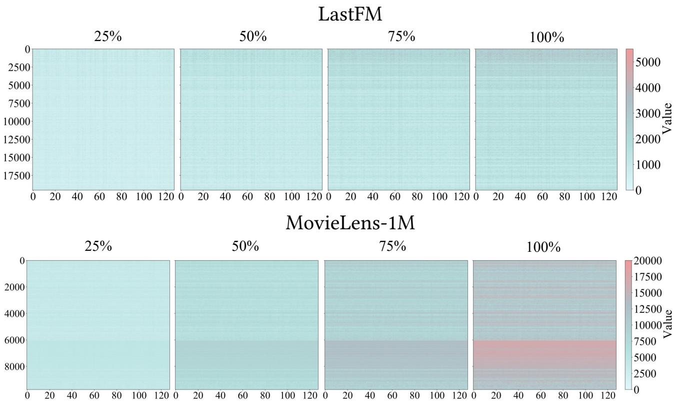

Figure 2: Parameter matrix updates during training.

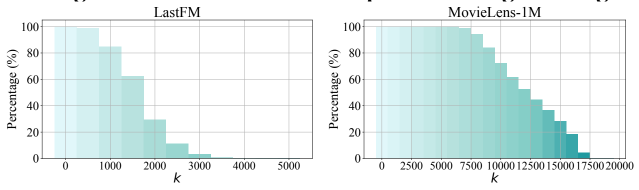

Figure 3: Percentage of parameter updated more than k times.

it still aggregates node representations at each layer, it behaves equivalently to a decoupled GNN in terms of parameterization and embedding learning. This equivalence can be demonstrated by setting $w_\ell = 1/(L+1)$ in Equation 1 and letting $\mathbf{X} = \mathbf{I}$, where $\mathbf{I}$ is the $n \times n$ identity matrix. Under these settings, Equation 1 becomes $\mathbf{Z} = 1/(L+1)\sum_{\ell=0}^{L} \mathbf{P}^\ell \mathbf{I}$. Substituting this into the embedding computation yields $\mathbf{E} = \mathbf{Z}\mathbf{W} = 1/(L+1)\sum_{\ell=0}^{L} \mathbf{P}^\ell \mathbf{I}\mathbf{W}$, which matches Equation 2, where $\mathbf{E}^{(0)}$ corresponds to the parameter matrix $\mathbf{W}$ in decoupled GNNs. This equivalence is further supported by empirical results presented in our technical report. Observation 1 aligns perfectly with the statement in recommender systems that **the IDs of users and items are used as input features**. In these systems, users and items lack intrinsic features beyond their IDs, which effectively results in a one-hot encoded input. This setup is analogous to a scenario in decoupled GNNs where an identity matrix serves as the feature matrix.

Observation 1. *In terms of embedding learning and model parameters, LightGCN can be seen as a specialized form of decoupled GNN, where the input feature matrix is set to an identity matrix.*

According to the mathematical formulation, the dimensions of the parameter matrix $\mathbf{W}$ are determined by the feature dimensions. When $\mathbf{X}$ is an identity matrix, the feature dimension becomes n, resulting in a parameter size of $n \times d$ for LightGCN [9]. JGCF [8] and LightGCL [3] employ polynomial-based filters and GCL, respectively, to improve model performance. Due to their adherence to LightGCN's embedding learning framework, their large parameter sizes can be attributed to the same factors outlined previously.

Redundancy in Parameter Matrices. Considering that the parameter matrix in LightGCN [9] and its variants scales with $n \times d$, we conducted experiments on the LastFM and MovieLens-1M datasets to examine its necessity and potential redundancy. Specifically, we tracked parameter update frequencies during training to assess utilization. As shown in Figure 2, only a small portion of the parameter

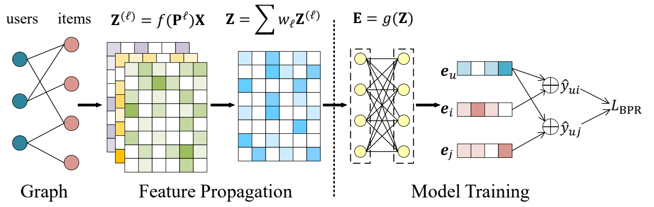

Figure 4: An overview of the proposed Lighter-X framework.

matrix continues to update during the later stages of training. This effect is particularly evident on the LastFM dataset, where many parameters become static in the early phases, indicating redundancy. To further investigate this, Figure 3 shows the percentage of parameters updated more than k times. On MovieLens-1M, most parameters are updated infrequently, and the trend is even more pronounced in the LastFM dataset, where fewer than 20% of the parameters are updated more than 2,500 times out of a possible 5,000. These findings suggest that the parameter matrix $\mathbf{W}$, also referred to as $\mathbf{E}^{(0)}$, in LightGCN is highly redundant. Since many graph-based recommender models adopt similar parameter settings, this highlights a broader need for parameter optimization.

Observation 2. *Parameter matrices in models like LightGCN exhibit significant redundancies, demonstrating that the training parameter matrix is inherently sparse.*

To address this, we propose a foundational assumption: $\mathbf{W} = \mathbf{W}_{\text{s}} + \mathbf{W}_{\text{v}}$, where $\mathbf{W}_{\text{s}}$ consists of static parameters and $\mathbf{W}_{\text{v}}$ contains the learnable, varying components. Correlating the results from Figures 2 and 3, it becomes evident that $\mathbf{W}_{\text{v}}$ should be a sparse matrix. The redundancy observed in the parameter matrix suggests that a large portion of $\mathbf{W}$ remains effectively unchanged during training and does not significantly contribute to the learning process. Our empirical results confirm that only a small subset of parameters in $\mathbf{W}_{\text{v}}$ are meaningfully updated, highlighting a clear opportunity for reducing model size and improving efficiency.

4 THE LIGHTER-X METHOD

In this section, we introduce the **Lighter-X** framework and demonstrate its universal applicability by applying it to various representative models. As discussed in Section 3, LightGCN uses an identity matrix of dimension $n \times n$ as the feature matrix $\mathbf{X}$, which necessitates a weight matrix $\mathbf{W}$ of size $n \times d$. This setup results in a substantial number of parameters. To tackle this issue, we propose using a low-rank matrix $\mathbf{X} \in \mathbb{R}^{n \times h}$ as the input feature matrix. This adjustment results in a weight matrix $\mathbf{W}$ of size $h \times d$ according to the standard computation, where $h \ll n$. **Given this context, what constitutes an optimal low-rank matrix X?**

In recommender systems, data is typically large-scale but inherently sparse. As discussed in Section 3, this sparsity extends to the training parameters of the models. Building on this characteristic, our approach leverages compressed sensing to efficiently derive low-rank matrices, which is essential for managing large data volumes with reduced computational overhead. This method provides a robust alternative to traditional techniques such as SVD. Although popular for achieving low-rank approximations, SVD is challenged by significant computational demands in large graph scenarios [5]. To achieve the restricted isometry property (RIP) in the optimal regime for compressed sensing, we utilize random matrices, a common technique for rapid dimensionality reduction. Even under significant compression, the original signal can be accurately reconstructed from a small number of observations, provided the signal retains sparse characteristics. This reconstruction is achieved through optimization algorithms [6]. In other words, the compressed matrix can preserve the essential features of the data, making compressed sensing a promising approach for accelerating convolutional operations by effectively reducing dimensionality early in the network.

Optimizing sparse data in graph structures. To optimize the sparse data in graph structures, we construct an efficient input feature matrix $\mathbf{X} = \mathbf{P} \cdot \mathbf{S}$ using cost-effective random sampling, where $\mathbf{S} \in \mathbb{R}^{n \times h}$ is a random matrix designed to satisfy the RIP condition. Specifically, $\mathbf{S}$ can be either a Gaussian or Bernoulli random matrix, both of which are widely used in compressed sensing due to their simplicity, generality, and ability to satisfy the RIP condition [4, 13, 33]. The dimensions of $\mathbf{S}$ are chosen to meet the following requirement:

$$h = c \cdot r \log(n/r), \tag{5}$$

where r represents the sparsity level and c is a customizable constant. Traditional methods maintain graph propagation precision by generating an ID-specific one-hot vector for each node, which leads to inefficient resource usage, as this approach requires n entries for n signals. In contrast, by utilizing compressed sensing and random sampling, as described in Equation 5, our method scales with $\log(n)$, significantly reducing resource consumption while preserving essential features.

Optimizing sparse trainable parameters. Following a similar strategy used for sparse graph structures, we replace the sparse parameter matrix discussed in Section 3 with a trainable matrix $\mathbf{W}'$, initialized from a Gaussian distribution. Since the random projection matrix $\mathbf{S}'$ is also sampled from a Gaussian distribution [33], their product $\mathbf{S}'\mathbf{W}'$ satisfies the distributional properties required in compressed sensing [4]. Importantly, the dimensionality of learnable weight $\mathbf{W}'$ is $h \times d$, independent of the number of nodes n, which contributes to improved scalability. Therefore, the model can bypass the traditional reconstruction step and instead rely on end-to-end training to learn effective representations.

Decoupled framework for graph-based recommendation. The coupled model structure is another important factor limiting the scalability of traditional GCN [11] and LightGCN [9]. Specifically, these models typically require convolutional operations to be performed on the entire graph, which is computationally expensive and difficult to scale to large graphs. A series of studies has improved GCN scalability by decoupling feature propagation from the training process, allowing computationally intensive convolution operations to be precomputed [2, 24, 29]. Extending this idea, our introduction of low-rank random matrices enables the decoupling of Lighter-X, allowing the costly and time-consuming feature propagation operations to be executed only once during the pre-computation phase. Figure 4 illustrates the final Lighter-X framework, where $f(\cdot)$ is the propagation function responsible for spreading information across nodes, and $g(\cdot)$ is the learning

Algorithm 1 Training Algorithm for LighterGCN

```
1:  Input: User-item interaction matrix R, adjacency matrix A, degree matrix D, number of GNN layers $L$, random matrix dimension coefficients $c$
2:  Output: Predicted score matrix $\hat{\mathbf{Y}}$, learned embeddings $\mathbf{E}$
3:  # Preprocessing
4:  Compute normalized adjacency matrix $\mathbf{P} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$
5:  Generate feature matrix $\mathbf{X} = \textsc{GenFeat}(\mathbf{R}, c)$
6:  Compute feature propagation matrix $\mathbf{Z} = \sum_{\ell=0}^{L} w_\ell \mathbf{P}^\ell \mathbf{X}$
7:  # Training
8:  for each mini-batch with $B$ user-item pairs $(u, i, i^-)$ do
9:      $\mathbf{Z}_B$ = rows of $\mathbf{Z}$ indexed by $\{u, i, i^-\}$
10:     Get embeddings for nodes in batch $\mathbf{E}_B = MLP(\mathbf{Z}_B)$
11:     $\mathcal{L}_{\mathrm{BPR}} = -\log\left[\mathrm{sigmoid}\left(\boldsymbol{e}_u^\top \boldsymbol{e}_i - \boldsymbol{e}_u^\top \boldsymbol{e}_{i^-}\right)\right]$
12:     Update MLP's parameters using gradient descent
13: end for
14: # Inference
15: Get embeddings for all nodes $\mathbf{E} = MLP(\mathbf{Z})$
16: $\mathbf{E}_U$ = rows of $\mathbf{E}$ indexed by $\{1, \ldots, |U|\}$
17: $\mathbf{E}_I$ = rows of $\mathbf{E}$ indexed by $\{|U| + 1, \ldots, |U| + |I|\}$
18: Predict score matrix $\hat{\mathbf{Y}} = \mathbf{E}_U \mathbf{E}_I^\top$
```

function, typically implemented as an MLP trained for downstream tasks. In the feature propagation stage, we complete the convolution related operation and obtain the feature propagation matrix $\mathbf{Z}$. The subsequent neural network takes $\mathbf{Z}$ as input and is trained to generate the final user and item embeddings. This training process is guided by the Bayesian Personalized Ranking (BPR) loss.

4.1 LighterGCN

We begin by applying the proposed Lighter-X framework to extend LightGCN [9], which we call LighterGCN. This is particularly relevant, since LightGCN serves as an foundational backbone for many GNN-based recommendation models. Specifically, LighterGCN adopts a low-rank approximation and decoupling framework to optimize the embedding process. Formally, LighterGCN learns embeddings using the following equation:

$$\mathbf{E} = MLP(\mathbf{Z}) = MLP(\sum_{\ell=0}^{L} w_\ell \mathbf{Z}^{(\ell)}), \quad \mathbf{Z}^{(\ell)} = \mathbf{P}^\ell \mathbf{X}, \tag{6}$$

where $\mathbf{X}$ is the random sampling result with rank h, which is much smaller than the number of nodes n. Based on this low-rank input feature matrix $\mathbf{X}$, LighterGCN performs graph convolutional operations to compute the feature propagation matrix $\mathbf{Z}$. Finally, an MLP is trained to produce the final embedding $\mathbf{E}$. As a result, LighterGCN reduces the number of parameters from $O(nd)$ to $O(hd)$, where $h \ll n$, thereby simplifying computation and improving learning efficiency. By precomputing feature propagation using the introduced low-rank random matrix, LighterGCN not only maintains the expressive power of the original LightGCN but also achieves greater scalability and efficiency.

Learning Algorithm. The LighterGCN method, summarized in Algorithm 1, consists of three main stages: preprocessing, training, and inference. During preprocessing (Lines 4–6), we first compute the normalized adjacency matrix, as is standard in many existing methods. We then generate feature matrices using a randomized approach and construct the feature propagation matrix following the LightGCN formulation, using the low-rank feature matrix as input. This shared propagation mechanism enables LighterGCN to effectively preserve the strengths of LightGCN. In the training phase (Lines 8–12), we sample mini-batches of user-item pairs and learn embeddings using an MLP. Importantly, no graph-related operations are required during training, which significantly improves efficiency. Since each row of the feature propagation matrix is independent, computations are restricted to the relevant nodes in each mini-batch, avoiding redundant full-graph convolutions and further enhancing scalability. Finally, during inference (Lines 15–18), we compute predicted user-item relevance scores by multiplying the learned embeddings. To facilitate understanding and comparison of computational stages, we present an overview of the training pipeline. Further details are available in the technical report.

4.2 Lighter-X in Polynomial-based Graph Filters

As mentioned in Section 2.2, polynomial-based graph collaborative filtering is formally equivalent to applying different polynomial bases to compute the aggregation weights for each convolutional layer, such as Jacobi polynomial bases used in JGCF [8]. Under the Lighter-X framework, we can naturally incorporate polynomial-based methods by aggregating the propagation matrix $\mathbf{Z}$ using different polynomial bases. This approach leverages the representational power of varied bases while allowing the aggregations to be precomputed, thereby reducing computational complexity.

LighterJGCF. We use a low-rank random matrix as input features and precompute polynomial features at each level. The precomputed results are then fed into an MLP to learn the final embeddings of users and items. Specifically, we utilize the low-rank feature matrix $\mathbf{X}$ and the decoupled framework introduced in Section 4 to reformulate Equation 3 into the following form:

$$\mathbf{E}_{low} = MLP(\mathbf{Z}) = MLP(\sum_{\ell=0}^{L} w_\ell \mathbf{Z}^{(\ell)}), \quad \mathbf{Z}^{(\ell)} = \mathbf{J}_\ell^{a,b}(\mathbf{P})\mathbf{X}. \tag{7}$$

Similarly, we obtain $\mathbf{E}_{mid} = tanh(\beta MLP(\mathbf{X}) - \mathbf{E}_{low})$. Taking a single-layer MLP as an example, the dimensionality of the model parameter matrix is $h \times d$, which is much smaller than that of original JGCF model $(n \times d)$. In addition, the polynomial basis functions can be precomputed to accelerate the graph convolution process.

4.3 Lighter-X in GCL for Recommendation

The core of GCL for recommendation, as discussed in Section 2.2, involves generating a perturbed adjacency matrix $\hat{\mathbf{A}}$ through various data augmentation techniques. This matrix is then substituted into the embedding formula to derive the perturbed embedding. For example, LightGCL [3] uses truncated SVD to obtain $\hat{\mathbf{A}}$. Within the Lighter-X framework, we adopt the same precomputation approach to obtain the perturbed propagation matrix $\hat{\mathbf{Z}}$ and its corresponding embedding matrix. This strategy enables the simultaneous precomputation of both the perturbed and standard propagation matrices, thereby improving computational efficiency.

LighterGCL. Since LightGCN underlies the embedding learning in LightGCL, its parameter size is $n \times d$, identical to that of LightGCN. To reduce this scale, LighterGCL adopts LighterGCN as its backbone, producing embeddings $\mathbf{E}$ with a parameter size of $h \times d$, where $h \ll n$,

Table 1: The comparison of time complexity between baseline and proposed models. n, m, $|U|$ and $|I|$ represent the number of nodes, edges, users and items, respectively. B represents the batch size, n_B denotes the number of nodes in a batch, L is the number of layers in the model, d refers to the embedding size, h is the dimension of the feature matrix, and q is the required rank. T denotes the number of iterations in training and is equal to m/B.

Stage		Computation	LightGCN	JGCF	LightGCL	LighterGCN	LighterJGCF	LighterGCL
Pre-processing		Normalization	$O(2m)$	$O(2m)$	$O(2m)$	$O(2m)$	$O(2m)$	$O(2m)$
		SVD	-	-	$O(qm)$	-	-	$O(qm)$
		Graph Convolution	-	-	-	$O(2mLh)$	$O(2mLh)$	$O(2mLh+2qnLh)$
Training	One Batch	t_{conv}: Graph Convolution	$O(2mLd)$	$O(2mLd)$	$O(2mLd+2qnLd)$	$O(3Bhd)$	$O(3Bhd)$	$O(3Bhd+n_Bhd)$
		t_{bpr}: BPR Loss	$O(2Bd)$	$O(2Bd)$	$O(2Bd)$	$O(2Bd)$	$O(2Bd)$	$O(2Bd)$
		t_{ssl}: InfoNCE Loss	-	-	$O(Bd+Bn_Bd)$	-	-	$O(Bd+Bn_Bd)$
	Total		$(t_{\text{conv}}+t_{\text{bpr}}+t_{\text{ssl}})T$					
Inference		Graph Convolution	$O(2mLd)$	$O(2mLd)$	$O(2mLd)$	$O(nhd)$	$O(nhd)$	$O(nhd)$
		Calculate Scores	$O(\|U\|\|I\|d)$	$O(\|U\|\|I\|d)$	$O(\|U\|\|I\|d)$	$O(\|U\|\|I\|d)$	$O(\|U\|\|I\|d)$	$O(\|U\|\|I\|d)$

Table 2: The statistics of datasets.

Dataset	#User	#Item	#Interaction	Sparsity
LastFM	1,892	17,632	92,834	99.72%
MovieLens-1M	6,040	3,952	1,000,209	95.81%
MovieLens-20M	138,493	27,278	20,000,263	99.47%
Yelp-2018	31,668	38,048	1,561,406	99.87%

significantly reducing model complexity compared to LightGCL. To further improve efficiency and scalability, LighterGCL precomputes the perturbation component $\hat{\mathbf{Z}}$ using the low-rank input matrix $\mathbf{X}$. This strategy eliminates the need to compute perturbations during training, which is often a major bottleneck in graph contrastive learning. Specifically, the perturbed representations $\hat{\mathbf{Z}}^{(\ell)}$ at each layer are computed in advance using the perturbed adjacency matrix $\hat{\mathbf{P}}$ and the input features $\mathbf{X}$. The final perturbed embeddings are obtained by aggregating the precomputed $\hat{\mathbf{Z}}^{(\ell)}$ and passing the result through an MLP for training:

$$\hat{\mathbf{E}} = MLP(\hat{\mathbf{Z}}) = MLP(\sum_{\ell=0}^{L} w_\ell \hat{\mathbf{Z}}^{(\ell)}), \quad \hat{\mathbf{Z}}^{(\ell)} = \hat{\mathbf{P}} \cdot \mathbf{P}^{\ell-1}\mathbf{X}. \qquad (8)$$

where $\hat{\mathbf{Z}}^{(0)} = \mathbf{X}$. As a result, the repetitive perturbation generation required in conventional approaches is circumvented by leveraging the low-rank feature matrix and the decoupling framework in LighterGCL. This substantially reduces both the time and space complexity, making LighterGCL more suitable for large-scale graph-based recommendation scenarios.

4.4 Analysis

GNN-based recommendation models typically incur significant computational costs due to the need to repeatedly perform convolution operations on the entire graph during training. In contrast, we decouple the costly feature propagation from the training process, enabling models to precompute these convolution operations. This avoids redundant computations throughout training and significantly improves efficiency. Specifically, Lighter-X models only perform graph convolution during the preprocessing stage, and it only needs to be performed **once**. In contrast, baseline methods must repeat the convolution over the entire graph in **each training batch**. As shown in Table 1, we compare preprocessing cost, per-batch training complexity, total training complexity, and inference complexity between Lighter-X and baseline models. Due to space constraints, the detailed derivation is deferred to the technical report. The results demonstrate that Lighter-X retains the theoretical advantages of its base models while substantially improving training efficiency across various applications.

5 EXPERIMENTS

5.1 Experimental Setup

Datasets. We conduct experiments on four datasets. (1) **LastFM** contains the listening history of users on the Last.fm online music system. (2) **MovieLens-1M** and (3) **MovieLens-20M** contain movie rating data from the MovieLens website, with each record reflecting a user's rating for a particular movie. (4) **Yelp2018** is collected from users' reviews of merchants on Yelp[1].

Baselines. We consider three representative models LightGCN [9], JGCF [8] and LightGCL [3] as important baselines and conduct a comprehensive comparison of their performance and training efficiency against Lighter-X. Furthermore, we evaluate our models against other recommendation systems, including BPR [18], NeuMF[10], NGCF [25], DGCF [26], RGCF [21], DirectAU [23], LT-GNN [35], LightGODE [36], and SVD-GCN [15], which also aims to reduce parameter counts in recommendation models.

5.2 Experiments on Public Datasets

Evaluation Protocols. In this experiment, for each user, we randomly select 80% and 10% of the interactions as the training and

[1]https://www.yelp.com/

Table 3: Performance comparison at public datasets, with metrics evaluated at @10.

Method		LastFM			MovieLens-1M			MovieLens-20M			Yelp2018		
		Recall	NDCG	*#Params*	Recall	NDCG	*#Params*	Recall	NDCG	*#Params*	Recall	NDCG	*#Params*
Standard Models	BPR	0.1699	0.1632	*2.50M*	0.1658	0.2583	*1.25M*	0.1757	0.2207	*21.15M*	0.0452	0.0355	*4.46M*
	NeuMF	0.1633	0.1556	*2.50M*	0.1416	0.2239	*1.25M*	0.1645	0.1965	*21.15M*	0.0313	0.0235	*4.46M*
	NGCF	0.1809	0.1772	*2.53M*	0.1462	0.2413	*1.28M*	0.2027	0.2636	*21.18M*	0.0459	0.0364	*4.49M*
	DGCF	0.1876	0.1802	*2.50M*	0.1783	0.2700	*1.25M*	OOM	OOM	*21.15M*	0.0527	0.0419	*4.46M*
	RGCF	0.1959	0.1904	*2.50M*	**0.1909**	0.2774	*1.25M*	OOM	OOM	*21.15M*	0.0633	0.0503	*4.46M*
	DirectAU	0.1771	0.1657	*2.50M*	0.1569	0.2087	*1.25M*	0.1098	0.1363	*21.15M*	0.0557	0.0435	*4.46M*
	LTGNN	0.1924	0.1789	*2.50M*	0.1780	0.2752	*1.25M*	0.1303	0.1743	*21.15M*	0.0430	0.0333	*4.46M*
	LightGODE	0.2037	0.1965	*2.50M*	0.1546	0.1978	*1.25M*	0.1843	0.2293	*21.15M*	0.0585	0.0468	*4.46M*
	SVD-GCN	0.1688	0.162	*0.02M*	0.1598	0.2484	*0.02M*	-	-	-	0.0508	0.0402	*0.01M*
Base Models	LightGCN	0.1952	0.1878	*2.50M*	0.1688	0.2650	*1.25M*	0.2129	0.2730	*21.15M*	0.0560	0.0450	*4.46M*
	JGCF	0.2054	0.1971	*2.50M*	0.1863	0.2823	*1.25M*	0.2185	0.2804	*21.15M*	0.0687	**0.0556**	*4.46M*
	LightGCL	0.2050	0.2018	*2.50M*	0.1592	0.2539	*1.25M*	0.1172	0.1578	*21.15M*	0.0617	0.0496	*4.46M*
Lighter-X	LighterGCN	0.1946	0.1882	*0.40M*	0.1818	0.2731	*0.19M*	0.2108	0.2780	*1.70M*	0.0566	0.0451	*0.17M*†
	LighterJGCF	**0.2095**	0.1952	*0.40M*	0.1883	**0.2839**	*0.19M*	**0.2268**	**0.2882**	*1.70M*	**0.0694**	0.0538	*0.17M*†
	LighterGCL	0.2059	**0.2021**	*0.40M*	0.1753	0.2642	*0.19M*	0.1688	0.2217	*1.70M*	0.0627	0.0497	*0.17M*†

† The workshop version contained minor typographical errors, which have been corrected in this revised submission.

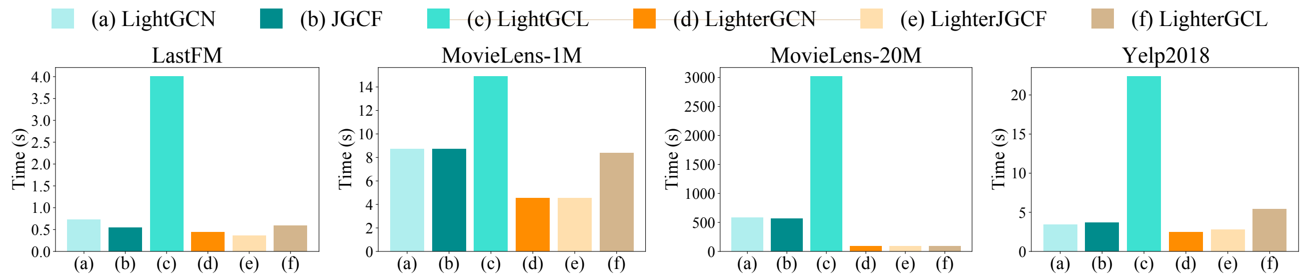

Figure 5: Comparison of training time per epoch.

validation sets, while the others are left for testing. We use Recall and NDCG as the evaluation metrics and have the recommender models generate a ranked list of 10 or 20 items to compare against the ground truth. Due to space limitation, we focus on Recall@10 and NDCG@10 in the main paper, while the other results are provided in the technical report.

Effectiveness. The overall performance of our framework compared to different base models is presented in Table 3. We can see, our framework can achieve comparable or even better performances than the base model across all the evaluation metrics and datasets. These results are encouraging, given that our framework uses significantly fewer parameters. This suggests that many parameters in traditional graph-based recommender models may be redundant and contribute little to performance improvement. Usually, recommender systems must process large volumes of real-time data, which demands high training efficiency. The above experiments demonstrate that our lightweight framework is well-suited to meet this requirement. Finally, our framework serves as an efficient plug-and-play strategy, which makes it more flexible and practical in real-world scenarios.

Efficiency. In the above experiments, we demonstrate the effectiveness of our framework. A more significant advantage of our framework is its efficiency. In this experiment, we analyze the time cost of our framework in the training phase. To evaluate the cost, we compare our framework with different base models for training one epoch. As shown in Figure 5, our framework can greatly reduce the time cost as compared with the base model. For example, on the MovieLens-20M dataset, the training time of Lighter-X is about 1/6 of the base model's. This result verifies the potential of our framework for efficient model training, which is crucial for practical recommender systems.

Performance comparison with other models. We also compare the proposed Lighter-X method against other leading recommendation algorithms on these public datasets. Among all the baselines, JGCF [8] performs the best. The proposed LighterJGCF achieves superior performance across most datasets with significantly fewer parameters. Although SVD-GCN [15] also reduces the scale of parameters, it leads to substantial performance degradation. Moreover, computing SVD efficiently on large-scale graphs remains a challenging and unresolved issue. For example, on the MovieLens-20M dataset, SVD-GCN [15] fails due to its inability to complete the SVD computation.

6 EVALUATION IN OTHER SCENARIOS

Beyond general recommendation, our proposed method can be adapted to other recommendation scenarios, including non-bipartite graphs (e.g., social recommendation) and context-aware recommendation. These settings introduce additional challenges, such as increased graph sparsity and the need to incorporate contextual information. In this section, we discuss how our approach can be extended to effectively address these alternative use cases.

6.1 Non-Bipartite Graphs

Most recommendation systems are based on bipartite graphs, where interactions occur between two distinct sets, such as users and items. In contrast, non-bipartite graph recommendation systems model more complex relationships where entities belong to the same set

Table 4: The statistics of non-bipartite graph datasets.

Dataset	#Node	#Edge	Density
Pokec	1,632,803	27,560,308	0.0021%
LiveJournal	4,847,571	62,094,395	0.0005%

Table 5: Performance comparison on non-bipartite graph recommendation, '+ SS' indicates applying SSNet on the base model. Hit@100 (Hit) and NDCG@300 (NDCG) are reported.

Method	Pokec			LiveJournal		
	Hit	NDCG	*# Params*	Hit	NDCG	*# Params*
LightGCN	0.0654	0.0236	*104.50M*	0.0537	0.0240	*310.24M*
LightGCN + SS	0.1645	0.0536	*104.50M*	0.2604	0.0747	*310.24M*
LighterGCN	0.0754	0.0252	*2.93M*	0.2624	0.0822	*2.20M*
LighterGCN + SS	**0.1706**	**0.0552**	*2.94M*	**0.2678**	**0.0831**	*2.20M*

Table 6: The statistics of datasets with context. F_u, F_i, $F_{(u,i)}$ represent the number of attributes for user, item and interaction, respectively.

Dataset	#User	#Item	#Interaction	F_u	F_i	$F_{(u,i)}$
MovieLens-1M-C	6,040	3,952	1,000,209	3	2	2
Yelp-2018-C	213,171	94,305	3,277,932	8	4	5

Table 7: Performance comparison on context-aware recommendation, with metrics evaluated at @10.

Method	MovieLens-1M-C			Yelp2018-C		
	Recall	NDCG	*#Params*	Recall	NDCG	*#Params*
LightGCNC	0.1784	0.2713	*1.28M*	0.0310	0.0170	*39.53M*
LighterGCNC	0.1821	0.2834	*0.20M*	0.0382	0.0213	*1.04M*

and can have direct connections. This is particularly relevant in scenarios like social recommendation, where users interact with each other [14, 20], or when items have inherent relationships, such as movies in a cinematic universe [17, 28]. In graph-based recommendation models, user nodes u and item nodes i are mathematically equivalent in the message-passing framework. As a result, models such as LightGCN [9] can be directly applied to non-bipartite graphs without requiring structural modifications.

Datasets. To evaluate the performance of our model on non-bipartite graph recommendation, we conducted experiments on two real-world social network datasets provided by SSNet [20]: Pokec and LiveJournal[2]. The statistics of these datasets are presented in Table 4. Notably, the graphs in these datasets are larger and sparser compared to the bipartite graph used in Table 2.

Effectiveness. We assess model performance on the candidate retrieval task, where models are required to recall the positive candidate from the entire graph. We adopt Hit@100 and NDCG@300 as evaluation metrics. As shown in Table 5, LighterGCN consistently outperforms LightGCN while using only 0.007% to 0.2% of the parameters, further demonstrating the efficiency and effectiveness of the proposed method. Additionally, we compare the training times of the methods, which can be found in the technical report.

[2]https://snap.stanford.edu/data/index.html#socnets

6.2 Context-Aware Recommendation

In real-world recommendation scenarios, raw features are often extremely sparse, spanning hundreds of fields and millions of dimensions. To handle the high-dimensional and sparse nature of such contextual features, many studies adopt embedding techniques, which map categorical variables into low-dimensional dense vectors to compress representations and uncover latent semantic relationships [12, 22, 27]. Therefore, we first encode the multi-field attributes extracted from user behavior logs (e.g., age, gender, location) and item metadata (e.g., price, historical purchase counts) as one-hot vectors. These vectors are then transformed into dense embeddings using attribute-specific embedding matrices. We concatenate these attribute embeddings with the graph-enhanced embeddings $\mathbf{E}$ obtained from user and item IDs and the graph structure (as described in Equations 2 and 6) to form the final embedding.

Building on this methodology, we propose context-aware variants, namely LightGCNC and LighterGCNC, and evaluate their performance on the MovieLens-1M-C and Yelp2018-C datasets. Detailed model specifications can be found in the technical report. Dataset details are summarized in Table 6, where MovieLens-1M-C shares the same interaction data as MovieLens-1M in Table 2. For all experiments, the attribute embedding size is set to 16, and other experimental settings follow those described in Section 5.2. Experimental results in Table 7 show that LighterGCNC outperforms LightGCN while using only 0.03%–0.16% of the parameters. This demonstrates that our method can be naturally extended to context-aware recommendation scenarios while maintaining strong performance, further validating its generality.

7 CONCLUSION

In this paper, we address a prevalent issue in existing graph-based recommendation models: the extensive and redundant volume of parameters. We propose Lighter-X, an efficient plug-and-play strategy that effectively reduces model parameter count while retaining the theoretical advantages of the base models. By introducing compressed sensing, we achieve considerable expression capabilities with more compact parameters, significantly reducing the overall parameter count. By implementing decoupled propagation, efficiency and scalability of the proposed method are further improved. Empirical evaluations demonstrate that Lighter-X reduces parameter size and improves efficiency while maintaining comparable performance.

ACKNOWLEDGMENTS

This research was supported in part by National Natural Science Foundation of China (No. U2241212, No. 92470128), by Beijing Outstanding Young Scientist Program No.BJJWZYJH012019100020098, by Huawei-Renmin University joint program on Information Retrieval. We also wish to acknowledge the support provided by the fund for building world-class universities (disciplines) of Renmin University of China, by Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education, by Intelligent Social Governance Interdisciplinary Platform, Major Innovation & Planning Interdisciplinary Platform for the "Double-First Class" Initiative, Public Policy and Decision-making Research Lab, and Public Computing Cloud, Renmin University of China.

REFERENCES

[1] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J Nair. 2021. Accelerating recommendation system training by leveraging popular choices. In *VLDB*. 127–140.

[2] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. 2020. Scaling graph neural networks with approximate pagerank. In *KDD*. 2464–2473.

[3] Xuheng Cai, Chao Huang, Lianghao Xia, and Xubin Ren. 2023. LightGCL: Simple Yet Effective Graph Contrastive Learning for Recommendation. In *ICLR*. https://openreview.net/forum?id=FKXVK9dyMM

[4] Thong T Do, Trac D Tran, and Lu Gan. 2008. Fast compressive sampling with structurally random matrices. In *IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 3369–3372.

[5] Xinyu Du, Xingyi Zhang, Sibo Wang, and Zengfeng Huang. 2023. Efficient Tree-SVD for Subset Node Embedding over Large Dynamic Graphs. *PACMMOD* 1, 1 (2023), 1–26.

[6] Simon Foucart, Holger Rauhut, Simon Foucart, and Holger Rauhut. 2013. *An invitation to compressive sensing*. Springer.

[7] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. 2023. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *TORS* 1, 1 (2023), 1–51.

[8] Jiayan Guo, Lun Du, Xu Chen, Xiaojun Ma, Qiang Fu, Shi Han, Dongmei Zhang, and Yan Zhang. 2023. On Manipulating Signals of User-Item Graph: A Jacobi Polynomial-based Graph Collaborative Filtering. In *KDD*. 602–613.

[9] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*. 639–648.

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

[11] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[12] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1754–1763.

[13] Ran Lu. 2019. On the strong restricted isometry property of Bernoulli random matrices. *Journal of Approximation Theory* 245 (2019), 1–22.

[14] Yijun Ma, Chaozhuo Li, and Xiao Zhou. 2024. Tail-STEAK: improve friend recommendation for tail users via self-training enhanced knowledge distillation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 8895–8903.

[15] Shaowen Peng, Kazunari Sugiyama, and Tsunenori Mine. 2022. SVD-GCN: A simplified graph convolution paradigm for recommendation. In *CIKM*. 1625–1634.

[16] Yifang Qin, Wei Ju, Xiao Luo, Yiyang Gu, and Ming Zhang. 2024. PolyCF: Towards the Optimal Spectral Graph Filters for Collaborative Filtering. *arXiv preprint arXiv:2401.12590* (2024).

[17] Khalil Ur Rahman, Huifang Ma, Ali Arshad, and Azad Khan Baheer. 2022. Movie Recommender System Based On Heterogeneous Graph Neural Networks. In *2022 8th International Conference on Systems and Informatics (ICSAI)*. IEEE, 1–7.

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

[19] Xiran Song, Hong Huang, Jianxun Lian, and Hai Jin. 2024. XGCN: a library for large-scale graph neural network recommendations. *Frontiers of Computer Science* 18, 3 (2024), 183343.

[20] Xiran Song, Jianxun Lian, Hong Huang, Mingqi Wu, Hai Jin, and Xing Xie. 2022. Friend recommendations with self-rescaling graph neural networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3909–3919.

[21] Changxin Tian, Yuexiang Xie, Yaliang Li, Nan Yang, and Wayne Xin Zhao. 2022. Learning to denoise unreliable interactions for graph collaborative filtering. In *SIGIR*. 122–132.

[22] Zhen Tian, Ting Bai, Wayne Xin Zhao, Ji-Rong Wen, and Zhao Cao. 2023. EulerNet: Adaptive Feature Interaction Learning via Euler's Formula for CTR Prediction. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1376–1385.

[23] Chenyang Wang, Yuanqing Yu, Weizhi Ma, Min Zhang, Chong Chen, Yiqun Liu, and Shaoping Ma. 2022. Towards representation alignment and uniformity in collaborative filtering. In *KDD*. 1816–1825.

[24] Hanzhi Wang, Mingguo He, Zhewei Wei, Sibo Wang, Ye Yuan, Xiaoyong Du, and Ji-Rong Wen. 2021. Approximate graph propagation. In *KDD*. 1686–1696.

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

[26] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled graph collaborative filtering. In *SIGIR*. 1001–1010.

[27] Tianjun Wei and Tommy WS Chow. 2023. FGCR: Fused graph context-aware recommender system. *Knowledge-Based Systems* 277 (2023), 110806.

[28] Yuecen Wei, Xingcheng Fu, Qingyun Sun, Hao Peng, Jia Wu, Jinyan Wang, and Xianxian Li. 2022. Heterogeneous graph neural network for privacy-preserving recommendation. In *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 528–537.

[29] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *ICML*. PMLR, 6861–6871.

[30] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *SIGIR*. 726–735.

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

[32] Xinyi Wu, Donald Loveland, Runjin Chen, Yozen Liu, Xin Chen, Leonardo Neves, Ali Jadbabaie, Clark Mingxuan Ju, Neil Shah, and Tong Zhao. 2024. GraphHash: Graph Clustering Enables Parameter Efficiency in Recommender Systems. *arXiv preprint arXiv:2412.17245* (2024).

[33] Jianbo Yang, Xuejun Liao, Xin Yuan, Patrick Llull, David J Brady, Guillermo Sapiro, and Lawrence Carin. 2014. Compressive sensing by learning a Gaussian mixture model from measurements. *IEEE Transactions on Image Processing* 24, 1 (2014), 106–119.

[34] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Lizhen Cui, and Quoc Viet Hung Nguyen. 2022. Are graph augmentations necessary? simple graph contrastive learning for recommendation. In *SIGIR*. 1294–1303.

[35] Jiahao Zhang, Rui Xue, Wenqi Fan, Xin Xu, Qing Li, Jian Pei, and Xiaorui Liu. 2024. Linear-time graph neural networks for scalable recommendations. In *Proceedings of the ACM Web Conference 2024*. 3533–3544.

[36] Weizhi Zhang, Liangwei Yang, Zihe Song, Henry Peng Zou, Ke Xu, Liancheng Fang, and Philip S Yu. 2024. Do We Really Need Graph Convolution During Training? Light Post-Training Graph-ODE for Efficient Recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 3248–3258.