A APPENDIX

A.1 Pruning Algorithm of LightGNN

The pruning algorithm of our LightGNN is presented as follows [1].

Algorithm 1: The Pruning Algorithm of LightGNN

Input: Original adjacent matrix $\mathbf{A}$ of the interaction graph.
Output: Pruning edge mask $\mathbb{W}$ and embedding mask $\mathbb{E}$.

Randomly initialize the user/item embeddings $\mathbf{E}^s \in \mathbb{R}^{(I+J)\times d}$ and set the edge weights in $\mathbf{W}^s \in \mathbb{R}^{I\times J}$ to 1;
Initialize the binary edge mask $\mathbb{W} \in \mathbb{B}^{I\times J}$ for pruning, where $\mathbb{W}_{i,j} = 1$ if $a_{i,j} = 1$ and $\mathbb{W}_{i,j} = 0$ otherwise;
Initialize the embedding mask $\mathbb{E} \in \mathbb{B}^{(I+J)\times d}$ to all ones;
for *run* $i_{pr} = 1$ **to** N_{prune} **do**
  for *epoch* $j_{tr} = 1$ **to** N_{train} **do**
    $\mathbf{W}^s \coloneqq \mathbf{W}^s \odot \mathbb{W}, \mathbf{E}^s \coloneqq \mathbf{E}^s \odot \mathbb{E}$;
    Forward student GNN using $\mathbf{W}^s$ and $\mathbf{E}^s$ as in Eq. 4;
    Minimize the loss (Eq. 11) to optimize $\mathbf{W}^s$ and $\mathbf{E}^s$;
  end
  Find the edges $\tilde{\mathcal{E}} = \{(u_i, v_j)\}$ to prune, which have the lowest $\rho\%$ nonzero values in matrix $|\bar{\mathbf{W}}^s \odot \mathbb{W}|$ (Eq. 8);
  Find the embedding entries $\tilde{\mathbf{E}}^s = \{(x, y)\}$ to prune, which have the lowest $\rho'\%$ nonzero values in $|\mathbf{E}^s|$;
  $\mathbb{W}_{i,j} \coloneqq 0$ for $(u_i, v_j) \in \tilde{\mathcal{E}}$, $\mathbb{E}_{x,y} \coloneqq 0$ for $(x, y) \in \tilde{\mathbf{E}}^s$;
end
Return $\mathbb{W}, \mathbb{E}$.

A.2 Interpretations of Uniformity Constraint

This section illustrates how the adaptive uniformity constraint in our LightGNN samples positive pairs. As the pruning process progresses, the node embedding vectors tend to become sparse. Specifically, we represent the sparse embedding vector for user u_i as $\mathbb{e}_i^s = \mathbb{e}_i \odot \mathbf{e}_i$, where $\mathbb{e}_i$ represents the binary embedding mask. Similarly, the embedding vector for user u_{i^1} is denoted as $\mathbb{e}_{i^1}^s$. To illustrate, consider the following example of $\mathbb{e}_i^s$ and $\mathbb{e}_{i^1}^s$.

$$\mathbb{e}_i^s = [\mathbb{0}, \mathbb{0}, a_1, a_2, \mathbb{0}, a_3, \mathbb{0}, ..., a_4, ..., a_i, \mathbb{0}, ...]$$
$$\mathbb{e}_{i^1}^s = [\mathbb{0}, \mathbb{0}, b_1, b_2, \mathbb{0}, b_3, \mathbb{0}, ..., b_4, ..., b_i, \mathbb{0}, b_{i+1}, \mathbb{0}, ..., b_{i+\delta}] \quad (12)$$

In this example, the valid dimensions of $\mathbb{e}_{i^1}^s$ (*i.e.* $b_1, b_2, ...$) are almost the same as those of $\mathbb{e}_i^s$. This indicates that the embeddings for u_i and u_{i^1} exist in similar subspaces. Motivated by this discovered similarity, we sample positive samples for the adaptive uniformity constraint using the method described in Eq. 10.

A.3 Details of Baseline Methods

We present baseline methods used in our experiments as follows:

Traditional Collaborative Filtering Models:

- **BiasMF** [18] is a traditional matrix factorization method utilizing user/item biases to enhance representation learning.

Non-GNN Deep Neural Networks for Collaborative Filtering:

- **NCF** [14] is one of the earliest CF models that utilizes deep neural networks to enhance user-item interaction modeling with MLP.
- **AutoR** [23] utilizes a three-layer autoencoder based on fully-connected layers to obtain user embedding vectors.

[1] $\mathbb{B} = \{0, 1\}$ and $\mathbb{e}_i$ in Eq. 10 denotes a row in the embedding mask matrix $\mathbb{E}$

Graph Neural Networks for Collaborative Filtering:

- **GCMC** [3] is an early approach that suggests utilizing graph convolutional techniques for matrix completion tasks.
- **PinSage** [39] utilizes random walk techniques to enhance GCN for mining web-scale graph-structured data for recommendation.
- **STGCN** [43] utilizes autoencoding sub-networks on hidden features for the GCN architecture to improve performance.
- **NGCF** [29] proposes utilizing GNN layers for collaborative filtering by performing graph convolutions on the interaction graph.
- **GCCF** [6] and **LightGCN** [12] both employ simplified GCN architectures to improve performance in recommendation tasks.
- **DGCF** [30] proposes to enhance the GNN-based CF paradigm with disentangled user/item representation learning.

Self-Supervised Learning-based Recommendation Models:

- **SLRec** [37] applies feature-level data augmentation techniques and contrastive learning to improve recommender systems.
- **SGL** [32] proposes several techniques for graph augmentations and feature augmentations for graph contrastive learning.
- **NCL** [20] proposes a neighbor-wise contrastive learning technique to enrich self-supervised graph collaborative filtering.
- **SimGCL** [41] is a contrastive learning model with simple feature-level augmentation method based on random permutation.
- **HCCF** [35] employs global hypergraph signals to augment GNN-based models and performs cross-view contrastive learning.

Compressed Collaborative Filtering Approaches:

- **GLT** [7] proposes a unified graph sparsification framework for GNN pruning based on the lottery ticket theory for graphs.
- **UnKD** [5] proposes a stratified distillation strategy to rectify the biases in distillation signals for recommender compression.
- **SimRec** [34] leverages contrastive knowledge distillation techniques to compress a GNN-based CF method into a MLP network.

Additional Approaches:

- **GraphDA** [8] optimizes recommendation quality by refining neighbor selection through user-user and item-item correlations, enhancing interactions for users with varying activity levels.
- **XSimGCL** [40] enhances recommendations by balancing user and item representations, reducing bias, and improving coverage of diverse items through noise-based embeddings, outperforming graph augmentation techniques on sparse datasets.
- **AutoCF** [33] automates data augmentation for recommendation using a self-supervised learning framework, surpassing baselines in adaptability and representation quality.

A.4 More Implementation Details

For our LightGNN, we set the embedding size of the initial teacher model to 64, and distill it into the final student model with sparse embeddings. Specifically, the results for LightGNN in Table 2 are obtained with an embedding preservation ratio of 33%. Namely, the valid embedding size for the final predictive model of our LightGNN is $64 \times 33\% \approx 21$. We use LightGCN [12] as the backbone model for the teacher model, intermediate model, and student model. The

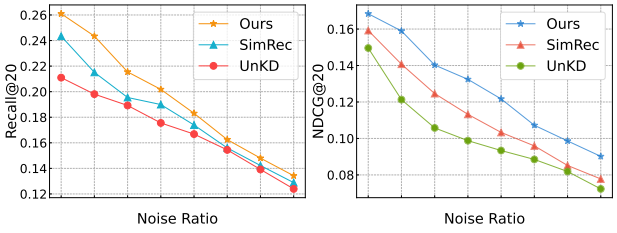


Figure 7: Performance *w.r.t* noise ratios on Gowalla data.

teacher model is configured with 8 GCN layers, and the other two models use 2 layers. The edge preservation ratio is set to 44% for Amazon and Gowalla, as well as 77% for Yelp, to produce the results in Table 2. β_1, β_2 in Eq. 8 are tuned from $\{0.05, 0.10, ..., 1.0\}$

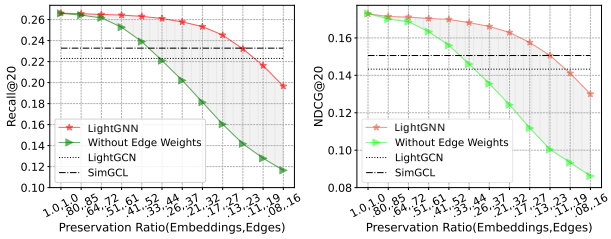
We use the released code of UnKD [5] in our experiments. LightGCN is used as the backbone model of UnKD for fair comparison. The teacher model in UnKD is configured with 100 embedding dimensions, and 8 GCN layers, and its student model uses 32 dimensions with 2 layers. We use the proposed UnKD sampler and the "group" parameter is selected from {2,3,...,8 }. Moreover, the decay regularization weights for both teacher and student models are tuned from $\{1e^{-2}, 1e^{-3}, ..., 1e^{-7}\}$. For SimRec [34], we also use the released code. Analogously, the teacher model is a LightGCN with 8 GCN layers and 32 embedding dimensions. The decay weight is selected from the same range as UnKD.

A.5 Robustness against Noise

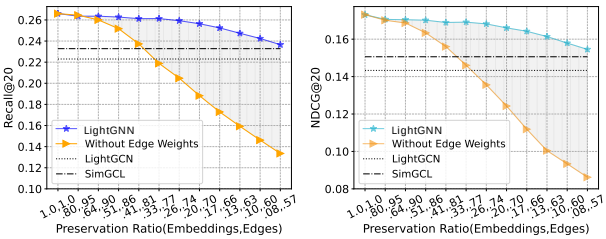
To evaluate the noise resistance of our model, we conduct experiments on Gowalla by replacing a certain percentage of interaction edges with random noise edges. We train our LightGNN using this noisy dataset and measure the performance in terms of Recall@20 and NDCG@20. Figure 7 illustrates the results for different noise ratios, demonstrating that our LightGNN maintains the superior performance under high noise ratio. This validates the superior noise resistance ability compared to state-of-the-art KD models. As the noise increases, the performance gap between our LightGNN and the baselines narrows, particularly in terms of the Recall metric. This trend is understandable because higher noise levels reduce the predictability of the training set for the test set. Excessive noise can disrupt predictability, leading to reduced efficacy and potential failure for all models. Additionally, in terms of the NDCG metric, the descent curve appears smoother, and the effect of narrowing the gap between the curves is less pronounced.

A.6 Impact of Learned Edge Weights

In this section, we introduce an additional experiment conducted on the Gowalla dataset to investigate the impact of the learned edge weights $\mathbf{W}$ (namely, $\mathbf{W}^s$) on model performance. Figure 8 presents the evaluated performance of the full version of our LightGNN framework and a variant of LightGNN without the student's learned edge weights, considering both mild and aggressive pruning strategies. The results clearly demonstrate the significant advantage of our LightGNN model in retaining model performance, especially when subjected to high-rate pruning. This finding emphasizes the importance of the learned weights, which effectively preserve the node-wise relatedness even after pruning a substantial number of observed edges. In contrast, the variant without edge weights, while preserving the same subset of interaction edges, fails to capture the differences in strength among different edges. This highlights how our LightGNN framework achieves high performance by retaining the pruned information in the learned edge weights, enabling the preservation of nuanced relationships among edges.



(a) Performance comparison under the aggressive pruning strategy.



(b) Performance comparison under the mild pruning strategy.

Figure 8: Performance of LightGNN with and without the learned edge weights, under different preservation ratios.

A.7 Details of Over-Smoothing Investigation

In this section, we elaborate how we investigate the anti-over-smoothing effect of our LightGNN, specifically as follows.

- **Visualization of Embedding Distribution**. We begin by visualizing the embedding distribution of LightGNN alongside three best-performing CF baselines, namely LightGCN, SGL, and SimGCL. For visualization, following [41], we randomly select 2000 item nodes as samples. These node embeddings are then projected onto a 2-D space using t-SNE and normalized using their l_2 norms. Next, we estimate the distribution of these sampled embeddings using density estimation based on Gaussian kernels (KDE). In Figure 5, we present the results as heat maps in the 2-D space, where darker color indicates higher probability. Moreover, we display the estimated probability density *w.r.t* angles.
- **Mean Average Distance (MAD) Values**. To perform a more quantitative analysis of the this issue, we further measure the MAD values [4] for our LightGNN and six baselines on the Yelp and Gowalla datasets. A lower MAD value indicates a stronger over-smoothing effect among the test nodes. Following [34], we evaluate the MAD metric on a sampled node set consisting of 1000 popular users and 1000 popular items. This choice of test node set is particularly effective in detecting the over-smoothing effect with MAD. The evaluation results are presented in Table 4.

Table 5: Results of our method w.r.t. different preservation ratios on the MovieLens-1M dataset.

Preserv. Ratio	77% edges, 21 entries		81% edges, 26 entries		85% edges, 32 entries	
Metrics	@20	@40	@20	@40	@20	@40
Recall	0.2413	0.3612	0.2433	0.3640	0.2460	0.3665
NDCG	0.3436	0.3623	0.3463	0.3653	0.3507	0.3690

Table 6: Statistical details of extra experimental datasets.

Dataset	# Users	# Items	# Interactions	Interaction Density
Tmall	35954	41390	1964361	1.32×10^{-3}
ML1M	6040	3706	920193	4.11×10^{-2}
ML10M	69878	10677	9200050	1.23×10^{-2}

Table 7: Results of additional baseline methods

Dataset	Metrics	SimRec	UnKD	AutoCF	GraphDA	Ours
Yelp	Recall@20	0. 0823	0. 0819	0. 0869	0.0814	0. 0879
	NDCG@20	0. 0414	0. 0392	0. 0437	0.0399	0. 0443
	Recall@40	0. 1251	0. 1202	0. 1273	0.1239	0. 1328
	NDCG@40	0. 0519	0. 0493	0. 0533	0. 0506	0. 0553
Gowalla	Recall@20	0.2434	0. 2331	0. 2538	0. 2374	0. 2610
	NDCG@20	0. 1592	0. 1496	0. 1645	0.1518	0. 1684
	Recall@40	0. 3399	0. 3301	0. 3441	0.3334	0. 3597
	NDCG@40	0. 1865	0. 1766	0. 1898	0. 1748	0. 1962

A.8 Additional Results of More Baselines on More Datasets

A.8.1 **Extra Benchmark Datasets.** Besides the datasets mentioned in the main text of the paper, we provide the test results of three additional test datasets here, namely **Tmall** (less than 0.1 million nodes and about 2 million interactions), **MovieLens-1M** (ML-1M, about 1 million records), and **MovieLens-10M** (ML-10M, about 10 million ratings). Tmall is an important online shopping platform of Alibaba Group, which is a top B2C platform in China. And here the public data set we use is a set of recommendation-related data provided by Tmall. The MovieLens dataset is a widely used dataset in the field of recommender systems. It contains movie ratings provided by users along with demographic information. You can find the statistical information about the datasets in Table 6.

Moreover, we tested our method and an additional state-of-the-art baseline, XSimGCL [40], on all of these extra datasets. The experimental results are presented in Table 8. We use two GNN layers for both ours and XSimGCL. For XSimGCL, the trained model uses the whole graph and 32 embedding entries for prediction while our model has 21 embedding entries for each node on average and utilizes 77% edges of those datasets (except for Amazon, for which we utilize only 44% edges), to make predictions. More results of our method obtained on ML-1M dataset with respect to different preservation ratios are also provided in Table 5. From Table 8 and Table 5, we can see both of these state-of-the-art models (ours and XSimGCL) can give pretty good results under such sufficient supervision signals since Movielens 1M/10M are pretty dense and contain sufficient user-item interactions, i.e., the supervisory signals.

A.8.2 **Extra Baselines.** In addition to the eighteen baseline models mentioned in the main text of the paper, we provide three additional state-of-the-art models as baseline methods, namely XSimGCL [40], GraphDA [8], and AutoCF [33]. You can refer to A.3 for more details. We evaluated these state-of-the-art baseline models on Yelp and Gowalla datasets (for the convenience of comparison with the results in the main text of the paper) and their performance is listed in Table 7. We use two GNN layers for all these baselines' final models. For baselines, the trained models use the whole graph and 32 embedding entries for prediction. Our trained model has 21 embedding entries for each node on average and utilizes only 44% edges of Gowalla datasets, as well as 77% edges for Yelp, to make predictions.

Table 8: Results on additional datasets.

Method	Metrics	Tmall	ML-1M	ML-10M	Amazon
XSimGCL	Recall@20	0.06463	0.2366	0.2777	0.1021
	NDCG@20	0.04520	0.3369	0.3204	0.0703
	Recall@40	0.10347	0.3551	0.3961	0.1462
	NDCG@40	0.05867	0.3559	0.3658	0.0839
Ours	Recall@20	0.06883	0.2413	0.2908	0.1189
	NDCG@20	0.04792	0.3436	0.3407	0.0820
	Recall@40	0.10856	0.3612	0.4174	0.1677
	NDCG@40	0.06164	0.3623	0.3762	0.0969

A.9 Discussion on Acceleration Effect

As highlighted in the GLT paper [7], the translation of theoretical advantages of sparse matrices into tangible acceleration and memory-saving benefits still poses an open question. Presently, most DNN accelerators (both software and hardware) are primarily optimized for dense matrices and regular operators. Due to inadequate optimization for sparse matrices in both software and hardware domains, the practical degree of acceleration falls short of the theoretical expectation. To the best of our knowledge, recent research has shown growing interest in hardware acceleration [1, 2, 11, 17, 31]. The development of optimal accelerators in the future holds the potential to significantly enhance the performance of LightGNN by effectively leveraging its capabilities.

REFERENCES

[1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. 2021. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Computing Surveys (CSUR)* 54, 9 (2021), 1–38.

[2] Adam Auten, Matthew Tomei, and Rakesh Kumar. 2020. Hardware acceleration of graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

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

[4] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. 2020. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 34. 3438–3445.

[5] Gang Chen, Jiawei Chen, Fuli Feng, Sheng Zhou, and Xiangnan He. 2023. Unbiased Knowledge Distillation for Recommendation. In *ACM International Conference on Web Search and Data Mining (WSDM)*. 976–984.

[6] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. 2020. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 34. 27–34.

[7] Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. 2021. A unified lottery ticket hypothesis for graph neural networks. In *International Conference on Machine Learning (ICML)*. PMLR, 1695–1706.

[8] Ziwei Fan, Ke Xu, Zhang Dong, Hao Peng, Jiawei Zhang, and Philip S Yu. 2023. Graph collaborative signals denoising and augmentation for recommendation. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*. 2037–2041.

[9] Jonathan Frankle and Michael Carbin. 2018. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In *International Conference on Learning Representations (ICLR)*.

[10] 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. *ACM Transactions on Recommender Systems (TRS)* 1, 1 (2023), 1–51.

[11] Tong Geng, Ang Li, Runbin Shi, Chunshu Wu, Tianqi Wang, Yanfei Li, Pouya Haghi, Antonino Tumeo, Shuai Che, Steve Reinhardt, et al. 2020. AWB-GCN: A graph convolutional network accelerator with runtime workload rebalancing. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 922–936.

[12] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *International ACM SIGIR conference on research and development in Information Retrieval (SIGIR)*. 639–648.

[13] Xiangnan He, Xiaoyu Du, Xiang Wang, Feng Tian, Jinhui Tang, and Tat-Seng Chua. 2018. Outer product-based neural collaborative filtering. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 2227–2233.

[14] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *The ACM Web Conference (WWW)*. 173–182.

[15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[16] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *The ACM Web Conference (WWW)*. 2704–2710.

[17] Kevin Kiningham, Philip Levis, and Christopher Ré. 2022. GRIP: A graph neural network accelerator architecture. *IEEE Trans. Comput.* 72, 4 (2022), 914–925.

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

[19] Yehuda Koren, Steffen Rendle, and Robert Bell. 2021. Advances in collaborative filtering. *Recommender systems handbook* (2021), 91–142.

[20] Zihan Lin, Changxin Tian, Yupeng Hou, and Wayne Xin Zhao. 2022. Improving graph collaborative filtering with neighborhood-enriched contrastive learning. In *The ACM Web Conference (WWW)*. 2320–2329.

[21] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

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

[23] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. 2015. Autorec: Autoencoders meet collaborative filtering. In *The ACM Web Conference (WWW)*. 111–112.

[24] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence* 2009 (2009).

[25] Jiaxi Tang and Ke Wang. 2018. Ranking distillation: Learning compact ranking models with high performance for recommender system. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 2289–2298.

[26] 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 *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. 1816–1825.

[27] Wenjie Wang, Fuli Feng, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. 2021. Denoising implicit feedback for recommendation. In *ACM International Conference on Web Wearch and Data Mining (WSDM)*. 373–381.

[28] Wenjie Wang, Yiyan Xu, Fuli Feng, Xinyu Lin, Xiangnan He, and Tat-Seng Chua. 2023. Diffusion Recommender Model. *International ACM SIGIR conference on research and development in Information Retrieval (SIGIR)* (2023).

[29] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 165–174.

[30] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled graph collaborative filtering. In *International ACM SIGIR conference on research and development in information retrieval (SIGIR)*. 1001–1010.

[31] Zhao Wang, Yijin Guan, Guangyu Sun, Dimin Niu, Yuhao Wang, Hongzhong Zheng, and Yinhe Han. 2020. Gnn-pim: A processing-in-memory architecture for graph neural networks. In *Advanced Computer Architecture: 13th Conference, ACA 2020, Kunming, China, August 13–15, 2020, Proceedings 13*. Springer, 73–86.

[32] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised graph learning for recommendation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 726–735.

[33] Lianghao Xia, Chao Huang, Chunzhen Huang, Kangyi Lin, Tao Yu, and Ben Kao. 2023. Automated Self-Supervised Learning for Recommendation. In *The ACM Web Conference (WWW)*. 992–1002.

[34] Lianghao Xia, Chao Huang, Jiao Shi, and Yong Xu. 2023. Graph-less collaborative filtering. In *The ACM Web Conference (WWW)*. 17–27.

[35] Lianghao Xia, Chao Huang, Yong Xu, Jiashu Zhao, Dawei Yin, and Jimmy Huang. 2022. Hypergraph contrastive collaborative filtering. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 70–79.

[36] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Guandong Xu, and Quoc Viet Hung Nguyen. 2022. On-device next-item recommendation with self-supervised knowledge distillation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 546–555.

[37] Tiansheng Yao, Xinyang Yi, Derek Zhiyuan Cheng, Felix Yu, Ting Chen, Aditya Menon, Lichan Hong, Ed H Chi, Steve Tjoa, Jieqi Kang, et al. 2021. Self-supervised learning for large-scale item recommendations. In *ACM International Conference on Information & Knowledge Management (CIKM)*. 4321–4330.

[38] Yaowen Ye, Lianghao Xia, and Chao Huang. 2023. Graph Masked Autoencoder for Sequential Recommendation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

[39] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. 974–983.

[40] Junliang Yu, Xin Xia, Tong Chen, Lizhen Cui, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2023. XSimGCL: Towards extremely simple graph contrastive learning for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 36, 2 (2023), 913–926.

[41] 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 *International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*. 1294–1303.

[42] An Zhang, Wenchang Ma, Xiang Wang, and Tat-Seng Chua. 2022. Incorporating bias-aware margins into contrastive loss for collaborative filtering. *Advances in Neural Information Processing Systems (NeurIPS)* 35 (2022), 7866–7878.

[43] Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. 2019. STAR-GCN: stacked and reconstructed graph convolutional networks for recommender systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 4264–4270.

[44] Shichang Zhang, Yozen Liu, Yizhou Sun, and Neil Shah. 2021. Graph-less Neural Networks: Teaching Old MLPs New Tricks Via Distillation. In *International Conference on Learning Representations (ICLR)*.

[45] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)* 52, 1 (2019), 1–38.