An Out-of-the-Box Application for Reproducible Graph Collaborative Filtering extending the Elliot Framework*

Daniele Malitesta†
Politecnico di Bari
Bari, Italy
daniele.malitesta@poliba.it

Claudio Pomo†
Politecnico di Bari
Bari, Italy
claudio.pomo@poliba.it

Vito Walter Anelli
Politecnico di Bari
Bari, Italy
vitowalter.anelli@poliba.it

Tommaso Di Noia
Politecnico di Bari
Bari, Italy
tommaso.dinoia@poliba.it

Antonio Ferrara
Politecnico di Bari
Bari, Italy
antonio.ferrara@poliba.it

ABSTRACT

Graph convolutional networks (GCNs) are taking over collaborative filtering-based recommendation. Their message-passing schema effectively distills the collaborative signal throughout the user-item graph by propagating informative content from neighbor to ego nodes. In this demonstration, we show how to run complete experimental pipelines with six state-of-the-art graph recommendation models in Elliot (i.e., our framework for recommender system evaluation). We seek to highlight four main features, namely: (i) we support reproducibility in PyTorch Geometric (i.e., the library we use to implement the baselines); (ii) reproduced graph models span across various GCN families; (iii) we prepare a Docker image to provide a self-consistent ecosystem for the running of experiments. Codes, datasets, and a video tutorial to install and launch the application are accessible at: https://github.com/sisinflab/Graph-Demo.

CCS CONCEPTS

• **Information systems → Recommender systems**; • **Computing methodologies → Neural networks**.

KEYWORDS

Recommendation, Graph Convolutional Networks, Docker

ACM Reference Format:
Daniele Malitesta, Claudio Pomo, Vito Walter Anelli, Tommaso Di Noia, and Antonio Ferrara. 2023. An Out-of-the-Box Application for Reproducible Graph Collaborative Filtering extending the Elliot Framework. In *The 3rd Workshop on Graph Learning Benchmarks, August 07, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/XXXXXXX.XXXXXXX

*With the only exception of Table 1, the content of the current work is taken from the paper by Malitesta et al. [10], published in Adjunct Proceedings of the 31st ACM Conference on User Modeling, Adaptation and Personalization (UMAP 2023).
†Corresponding authors: Daniele Malitesta (daniele.malitesta@poliba.it) and Claudio Pomo (claudio.pomo.@poliba.it).

1 INTRODUCTION

Recommendation algorithms are pervasive in several user applications. Their role is to fill the gap among users' needs and products/services by presenting customers with a list of possible preferred items. The most popular recommendation paradigm, collaborative filtering (CF), is driven by the intuition that users interacting with the same items could share similar preferences.

A bipartite and undirected graph can naturally represent user-item interactions. Graph convolutional networks (GCNs) have recently emerged as a popular technique in CF to refine users' and items' representations by iteratively aggregating the representations of neighbor nodes into each ego node (i.e., the message-passing schema). GCNs are currently adopted in many recommendation tasks, such as session-based [5], sequential [6], social [21], and multimedia [23] recommendation.

Despite their outbreak in both academia and industry by surpassing traditional CF approaches, limited effort has been put into building *unified* and *comprehensive* frameworks to train and evaluate state-of-the-art *graph recommendation systems*. Among the most noticeable mentions, we may recall RecBole [24, 25], which implements eight graph recommendation models for general recommendation (e.g., NGCF [17], LightGCN [7], DGCF [18], SGL [19], NCL [8], and SimGCL [22]). Recently, Zhu et al. [26] pave the way to a shared benchmarking pipeline for recommendation (i.e., BARS), and integrate thirteen models from the graph CF literature (besides some of the aforementioned models, they also reproduce, for instance, PinSage [20], DisenGCN [9], NGAT4Rec [16], GFCF [15], and UltraGCN [11]).

In this demonstration, we show how to run extensive experimental settings for six popular graph collaborative filtering models (i.e., NGCF, LightGCN, DGCF, SGL, UltraGCN and GFCF) that we recently integrated into Elliot [1], our framework for recommender systems evaluation. The paper outlines the following contributions:

- Differently from the stable version of Elliot[1] which uses TensorFlow as the primary backend [2], we introduce PyTorch Geometric[2] (i.e., one of the most popular Python libraries for geometric

[1]https://github.com/sisinflab/elliot.
[2]https://pytorch-geometric.readthedocs.io/en/latest/.

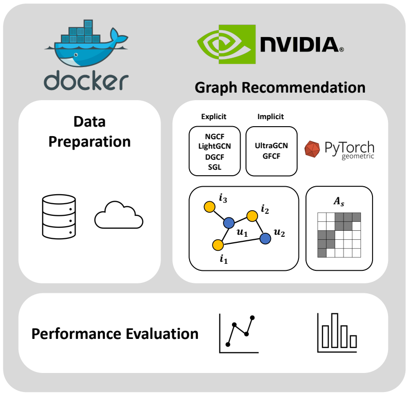


Figure 1: Architecture of Elliot for graph collaborative filtering. We integrate PyTorch Geometric as backend, categorize graph models into two classes, and dockerize the application.

deep learning) as the additional backend to design graph-based baselines adopting the explicit message-passing schema; at the time of this publication, only a few other frameworks have started to adopt it [24].
- Given the known reproducibility issues related to some non-deterministic operations in PyTorch Geometric [14], we implement message-passing with sparse adjacency matrices [12].
- In contrast to existing similar solutions (i.e., RecBole and BARS), we implement six state-of-the-art graph baselines by following a novel model categorization [3, 4] that distinguishes between methods using explicit message aggregation (i.e., NGCF, LightGCN, DGCF, and SGL) and going beyond the concept of graph convolution (i.e., UltraGCN and GFCF).
- As we leverage GPU boost by running PyTorch baselines with CUDA, we prepare a Docker image[3] creating a self-consistent and out-of-the-box experimental environment that requires minimal third-party libraries installed on the local machine. To the best of our knowledge, Elliot is the first recommendation framework to offer such functionalities.

Codes, datasets, and a video tutorial to install and launch the application are accessible at a public GitHub repository[4].

2 PROPOSED APPLICATION

This section presents our application for graph collaborative filtering in Elliot. First, we focus on the integration of PyTorch Geometric by addressing its reproducibility issues. Then, we describe the complete procedure to dockerize our application. Finally, we outline the steps to easily install and train/evaluate a graph recommender.

PyTorch Geometric in Elliot. Figure 1 depicts the overall architecture for our framework, organized into: (i) *data preparation*, (ii) *recommendation*, and (iii) *performance evaluation*. Diving into each of these modules is out of the scope of this paper (we extensively explained them in previous works [1, 2]). Conversely, our main focus is on integrating PyTorch Geometric into the existing Elliot environment to build and run graph recommendation models. It is worth mentioning that, in contrast to other graph recommendation frameworks, we propose a novel model categorization [3, 4] where we consider (i) explicit message-passing (e.g., LightGCN) and (ii) the simplification of graph convolution (e.g., UltraGCN). In the following, we deepen into graph convolution for recommendation. Given a user and item embeddings $\mathbf{e}_u$ and $\mathbf{e}_i$, the general formulation for the message-passing schema after l hops is:

$$\mathbf{e}_u^{(l)} = \omega\Big(\underbrace{\Big\{\mathbf{e}_{i'}^{(l-1)}, \forall i' \in \mathcal{N}(u)\Big\}}_{\text{messages}}\Big), \quad \mathbf{e}_i^{(l)} = \omega\Big(\underbrace{\Big\{\mathbf{e}_{u'}^{(l-1)}, \forall u' \in \mathcal{N}(i)\Big\}}_{\text{messages}}\Big), \tag{1}$$

where $\omega(\cdot)$ is the message aggregation, while $\mathcal{N}(u)$ and $\mathcal{N}(i)$ are the sets of 1-hop neighbor nodes for u and i.

Graph approaches leveraging message-passing (i.e., NGCF, LightGCN, DGCF, and SGL) inherit the `MessagePassing` base class [13]. This class provides, among the others, the functions **`propagate`** (which performs both the **`message`** and **`aggregate`** operations, defining the generic message and the message aggregation, respectively) and **`forward`** (which generates the outputs). The required input format to the **`forward`** function are, in the minimum setting, the node embeddings at hop $l-1$ ($\mathbf{E}^{(l-1)}$) and an edge array of dimension $2 \times 2M$ (**edges**), with M as the number of user-item/item-user recorded interactions, storing indices of users and items with a **bidirectional** connection.

Such implementation is straightforward, especially because it permits explicitly defining the custom message formulation for the generic node as done in several works from the literature (refer again to Equation (1)). However, there are known reproducibility issues [14] related to some operations in PyTorch Geometric since it **could** behave non-deterministically (e.g., the **`scatter`** function, called by **`aggregate`**). To handle it, we follow one of the most common strategies [14]. That is, we reformulate the single node message-passing schema from Equation (1) into a matrix formulation adopting sparse adjacency matrices [12]. As an example, let us define the message-passing formula for LightGCN [7] as:

$$\mathbf{e}_u^{(l)} = \sum_{i' \in \mathcal{N}(u)} \mathbf{e}_{i'}^{(l-1)}, \quad \mathbf{e}_i^{(l)} = \sum_{u' \in \mathcal{N}(i)} \mathbf{e}_{u'}^{(l-1)}. \tag{2}$$

We may rewrite it into the following compact expression:

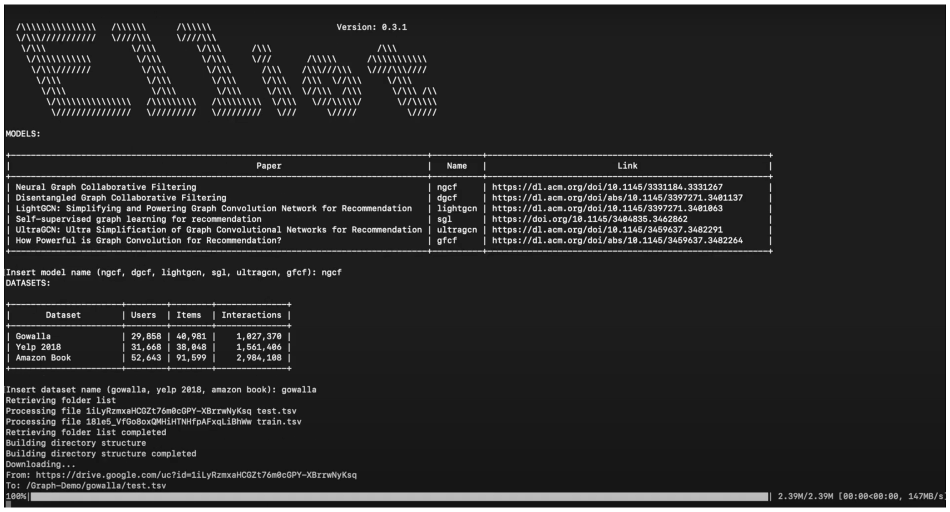
$$\mathbf{E}^{(l)} = \mathbf{A}_s \mathbf{E}^{(l-1)}, \tag{3}$$

where $\mathbf{A}_s$ is the sparse adjacency matrix for the bipartite and undirected user-item graph. By passing $\mathbf{A}_s$ instead of **edges** as input to **`forward`**, it will trigger the call of the **`message_and_aggregate`** function, which does not make use of the **`scatter`** operation. Unfortunately, this workaround is not always feasible (e.g., DGCF [18]).

Dockerization. As in similar Python frameworks for machine and deep learning (e.g., TensorFlow), PyTorch Geometric also supports models' training and inference with CUDA technologies for NVIDIA GPUs. Setting up a suitable development environment where versions compatibility is ensured for CUDA, cuDNN, and other third-party libraries could be cumbersome in some cases, especially in scenarios where users may need different installed versions of the same frameworks and libraries on one workstation. To tackle this challenge, we decide to leverage Docker[5] to create a self-consistent and out-of-the-box environment that provides the

[3]https://hub.docker.com/r/sisinflabpoliba/demo-graph.
[4]https://github.com/sisinflab/Graph-Demo.

[5]https://www.docker.com/.



```
 /\\\\\\\\\\\\\\\   /\\\\\\     /\\\\\\                    Version: 0.3.1
 \/\\\///////////   \////\\\    \////\\\
  \/\\\                \/\\\       \/\\\     /\\\               /\\\
   \/\\\\\\\\\\\        \/\\\       \/\\\    \///      /\\\\\     /\\\\\\\\\\\
    \/\\\///////         \/\\\       \/\\\     /\\\   /\\\///\\\  \////\\\////
     \/\\\                \/\\\       \/\\\    \/\\\  /\\\  \//\\\    \/\\\
      \/\\\                \/\\\       \/\\\    \/\\\ \//\\\  /\\\     \/\\\ /\\
       \/\\\\\\\\\\\\\\\  /\\\\\\\\\  /\\\\\\\\\ \/\\\  \///\\\\\/      \//\\\\\
        \///////////////  \/////////  \/////////  \///     \/////         \/////

MODELS:

+--------------------------------------------------------------------------+----------+-----------------------------------------------------+
|                                  Paper                                   |   Name   |                        Link                         |
+--------------------------------------------------------------------------+----------+-----------------------------------------------------+
| Neural Graph Collaborative Filtering                                     | ngcf     | https://dl.acm.org/doi/10.1145/3331184.3331267      |
| Disentangled Graph Collaborative Filtering                               | dgcf     | https://dl.acm.org/doi/abs/10.1145/3397271.3401137  |
| LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation | lightgcn | https://dl.acm.org/doi/10.1145/3397271.3401063 |
| Self-supervised graph learning for recommendation                        | sgl      | https://doi.org/10.1145/3404835.3462862             |
| UltraGCN: Ultra Simplification of Graph Convolutional Networks for Recommendation | ultragcn | https://dl.acm.org/doi/10.1145/3459637.3482291 |
| How Powerful is Graph Convolution for Recommendation?                    | gfcf     | https://dl.acm.org/doi/abs/10.1145/3459637.3482264  |
+--------------------------------------------------------------------------+----------+-----------------------------------------------------+

Insert model name (ngcf, dgcf, lightgcn, sgl, ultragcn, gfcf): ngcf
DATASETS:

+---------------------+--------+--------+--------------+
|       Dataset       | Users  | Items  | Interactions |
+---------------------+--------+--------+--------------+
| Gowalla             | 29,858 | 40,981 |    1,027,370 |
| Yelp 2018           | 31,668 | 38,048 |    1,561,406 |
| Amazon Book         | 52,643 | 91,599 |    2,984,108 |
+---------------------+--------+--------+--------------+

Insert dataset name (gowalla, yelp 2018, amazon book): gowalla
Retrieving folder list
Processing file 1iLyRzmxaHCGZt76m0cGPY-XBrrwNyKsq test.tsv
Processing file 18le5_VfGo8oxQMHiHTNHfpAFxqLiBhWw train.tsv
Retrieving folder list completed
Building directory structure
Building directory structure completed
Downloading...
From: https://drive.google.com/uc?id=1iLyRzmxaHCGZt76m0cGPY-XBrrwNyKsq
To: /Graph-Demo/gowalla/test.tsv
100%|██████████████████████████████████████████████████████████████████████| 2.39M/2.39M [00:00<00:00, 147MB/s]
```

Figure 2: Screenshot of the application start, where user can select the model (e.g., NGCF) and the dataset (e.g., Gowalla).

necessary libraries already installed in a proper configuration setting. Quite conveniently, we adopt the Docker container toolkit provided by NVIDIA[6] for the creation of containers equipped with customizable versions of CUDA and cuDNN.

First, we build a Docker image derived from this NVIDIA image[7], which comes with Ubuntu 20.04, CUDA 11.6.2, and cuDNN 8. Additionally, the custom image includes other useful Linux packages (e.g., Python 3.8 and pip), a cloned version of our GitHub repository for this demonstration, and all required Python packages to run the framework. You may refer to this link for the Dockerfile we use to build the image. Finally, to pull and run a Docker container from it, we also release the docker-compose YAML file (accessible at this link). It allows all GPUs on the machine to be used within the container, creates a bind mount between the `results/` folder in the container and a homonym folder on the host (thus storing permanently all generated files), and runs the application.

Installation and running. Thanks to the core functionalities of NVIDIA-powered Docker containers, the installation requirements needed to run our application are minimal. You may refer to the official installation guide[8].

As for the pre-requisites, ensure you have the proper NVIDIA drivers installed on the host machine. Then, follow the indicated procedure to install Docker[9] and the NVIDIA Container Toolkit[10]. If everything works smoothly, you should be able to pull and run a Docker container with any CUDA and cuDNN versions (you may test the application by launching a container with the command `nvidia-smi` which shows a snapshot on the GPU usage). Finally, Docker Compose needs to be installed on the host machine[11].

[6]https://github.com/NVIDIA/nvidia-docker.
[7]https://dockr.ly/3aWAtWt. The URL has been shortened to save space.
[8]https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html.
[9]https://docs.docker.com/engine/install/ubuntu/.
[10]https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#setting-up-nvidia-container-toolkit.
[11]https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-compose-on-ubuntu-22-04.

Once everything has been correctly set up, the custom Docker image can be pulled, and a container can be instantiated from it. To do so, you may use the docker-compose YAML file we provide in the GitHub repository. When the application starts (Figure 2), the user is asked to insert the model's name and the dataset, that is downloaded on-the-fly from the cloud. After that, the selected model is trained, validated, and tested on the chosen dataset for some hyper-parameter configurations (see later). You may refer to the official Elliot's documentation[12] and GitHub page for a comprehensive presentation of the formatting of the results. We also release a video tutorial for the reader[13].

3 AN EXPERIMENTAL FLOW

Settings. We test our application on three popular recommendation datasets, namely: Gowalla, Yelp2018 and Amazon Book. These datasets can be directly linked to those employed in the LightGCN experiments. Indeed, the methodological approach adopted in this investigation seeks to replicate the outcomes associated with six state-of-the-art models for graph-based recommendation systems. Notably, all of these models (excluding SGL) harness identical datasets, maintaining a consistent partitioning between training and testing subsets.

Framework outputs. Table 1 displays the accuracy recommendation performance for the best models' configurations, considering top-20 recommendation lists. Hyperparameters used by the authors of the original papers on their respective datasets were judiciously chosen to reproduce these results. Nevertheless, the current framework facilitates the exploration of hyperparameter values either within a grid-delineated space or by employing a strategy that balances exploration and exploitation, i.e., Tree-structured Parzen Estimator (TPE).

[12]https://elliot.readthedocs.io/en/latest/.
[13]https://www.youtube.com/watch?v=_Bpgf4wnwIU.

Table 1: Accuracy performance of our six graph-based baselines on Gowalla, Yelp 2018, and Amazon Book. The performance shift between the original results and the ones measured with our framework are maximum in the magnitude of 10^{-03}.

Datasets	Models	Ours		Original		Performance Shift	
		Recall	nDCG	Recall	nDCG	Recall	nDCG
Gowalla	NGCF	0.1556	0.1320	0.1569	0.1327	$-1.3 \cdot 10^{-03}$	$-7 \cdot 10^{-04}$
	DGCF	0.1736	0.1477	0.1794	0.1521	$-5.8 \cdot 10^{-03}$	$-4.4 \cdot 10^{-03}$
	LightGCN	0.1826	0.1545	0.1830	0.1554	$-4 \cdot 10^{-04}$	$-9 \cdot 10^{-04}$
	SGL*	—	—	—	—	—	—
	UltraGCN	0.1863	0.1580	0.1862	0.1580	$+1 \cdot 10^{-04}$	0
	GFCF	0.1849	0.1518	0.1849	0.1518	0	0
Yelp 2018	NGCF	0.0556	0.0452	0.0579	0.0477	$-2.3 \cdot 10^{-03}$	$-2.5 \cdot 10^{-03}$
	DGCF	0.0621	0.0505	0.0640	0.0522	$-1.9 \cdot 10^{-03}$	$-1.7 \cdot 10^{-03}$
	LightGCN	0.0629	0.0516	0.0649	0.0530	$-2 \cdot 10^{-03}$	$-1.4 \cdot 10^{-03}$
	SGL	0.0669	0.0552	0.0675	0.0555	$-6 \cdot 10^{-04}$	$-3 \cdot 10^{-04}$
	UltraGCN	0.0672	0.0553	0.0683	0.0561	$-1.1 \cdot 10^{-03}$	$-8 \cdot 10^{-04}$
	GFCF	0.0697	0.0571	0.0697	0.0571	0	0
Amazon Book	NGCF	0.0319	0.0246	0.0337	0.0261	$-1.8 \cdot 10^{-03}$	$-1.5 \cdot 10^{-03}$
	DGCF	0.0384	0.0295	0.0399	0.0308	$-1.5 \cdot 10^{-03}$	$-1.3 \cdot 10^{-03}$
	LightGCN	0.0419	0.0323	0.0411	0.0315	$+8 \cdot 10^{-04}$	$+8 \cdot 10^{-04}$
	SGL	0.0474	0.0372	0.0478	0.0379	$-4 \cdot 10^{-04}$	$-7 \cdot 10^{-04}$
	UltraGCN	0.0688	0.0561	0.0681	0.0556	$+7 \cdot 10^{-04}$	$+5 \cdot 10^{-04}$
	GFCF	0.0710	0.0584	0.0710	0.0584	0	0

**Results are not provided since SGL was not originally trained and tested on Gowalla [19].*

4 CONCLUSION

This demonstration shows full experimental flows with six graph collaborative filtering models in Elliot. Differently from previous Elliot versions and similar solutions in the literature, we leverage PyTorch Geometric and address its known reproducibility issues. Additionally, we propose a novel model categorization that distinguishes between explicit and implicit message-passing schema and takes advantage of Elliot's hyper-parameter optimization to allow extended ablation studies. Noteworthy, by dockerizing our application, we create an out-of-the-box experimental environment.

Acknowledgment. This work was partially supported by the following projects: IPZS-PRJ4_IA_NORMATIVO, Codice Pratica VHRWPD7 – CUP B97I19000980007 – COR 1462424 ERP 4.0, H2020 PASSEPARTOUT, Secure Safe Apulia, Codice Pratica 3PDW2R7 SERVIZI LOCALI 2.0, Casa delle Tecnologie Emergenti Comune di Matera, PON ARS01_00876 BIO-D, CT_FINCONS_III "Smart Rights Management Platform".

REFERENCES

[1] Vito Walter Anelli, Alejandro Bellogín, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Francesco Maria Donini, and Tommaso Di Noia. 2021. Elliot: A Comprehensive and Rigorous Framework for Reproducible Recommender Systems Evaluation. In *SIGIR*. ACM, 2405–2414.

[2] Vito Walter Anelli, Alejandro Bellogín, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Francesco Maria Donini, and Tommaso Di Noia. 2021. V-Elliot: Design, Evaluate and Tune Visual Recommender Systems. In *RecSys*. ACM, 768–771.

[3] Vito Walter Anelli, Yashar Deldjoo, Tommaso Di Noia, Daniele Malitesta, Vincenzo Paparella, and Claudio Pomo. 2023. Auditing Consumer- and Producer-Fairness in Graph Collaborative Filtering. In *ECIR (1) (Lecture Notes in Computer Science, Vol. 13980)*. Springer, 33–48.

[4] Vito Walter Anelli, Yashar Deldjoo, Tommaso Di Noia, Eugenio Di Sciascio, Antonio Ferrara, Daniele Malitesta, and Claudio Pomo. 2022. How Neighborhood Exploration influences Novelty and Diversity in Graph Collaborative Filtering. In *MORS@RecSys (CEUR Workshop Proceedings, Vol. 3268)*. CEUR-WS.org.

[5] Chen Chen, Jie Guo, and Bin Song. 2021. Dual Attention Transfer in Session-based Recommendation with Multi-dimensional Integration. In *SIGIR*. ACM, 869–878.

[6] Ziwei Fan, Zhiwei Liu, Jiawei Zhang, Yun Xiong, Lei Zheng, and Philip S. Yu. 2021. Continuous-Time Sequential Recommendation with Temporal Graph Collaborative Transformer. In *CIKM*. ACM, 433–442.

[7] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yong-Dong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *SIGIR*. ACM, 639–648.

[8] Zihan Lin, Changxin Tian, Yupeng Hou, and Wayne Xin Zhao. 2022. Improving Graph Collaborative Filtering with Neighborhood-enriched Contrastive Learning. In *WWW*. ACM, 2320–2329.

[9] Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. 2019. Disentangled Graph Convolutional Networks. In *ICML (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 4212–4221.

[10] Daniele Malitesta, Claudio Pomo, Vito Walter Anelli, Tommaso Di Noia, and Antonio Ferrara. 2023. An Out-of-the-Box Application for Reproducible Graph Collaborative Filtering extending the Elliot Framework. In *UMAP (Adjunct Publication)*. ACM, 12–15.

[11] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. 2021. UltraGCN: Ultra Simplification of Graph Convolutional Networks for Recommendation. In *CIKM*. ACM, 1253–1262.

[12] memory-efficient-aggregation-pytorch-geometric 2022. PyTorch Geometric Documentation. MEMORY-EFFICIENT AGGREGATIONS. https://pytorch-geometric.readthedocs.io/en/latest/notes/sparse_tensor.html. Accessed online on 15-05-2023.

[13] pytorch-geometric-message-passing 2022. PyTorch Geometric Documentation. Creating Message Passing Networks. https://pytorch-geometric.readthedocs.io/en/latest/notes/create_gnn.html. Accessed online on 15-05-2023.

[14] scatter-reproducibility-pytorch-geometric 2021. GitHub. PyG Team. PyTorch_Geometric. Issues. How to make scatter (Just CUDA) results repeatable. https://github.com/pyg-team/pytorch_geometric/issues/2788. Accessed online on 15-05-2023.

[15] Yifei Shen, Yongji Wu, Yao Zhang, Caihua Shan, Jun Zhang, Khaled B. Letaief, and Dongsheng Li. 2021. How Powerful is Graph Convolution for Recommendation?. In *CIKM*. ACM, 1619–1629.

[16] Jinbo Song, Chao Chang, Fei Sun, Xinbo Song, and Peng Jiang. 2020. NGAT4Rec: Neighbor-Aware Graph Attention Network For Recommendation. *CoRR* abs/2010.12256 (2020).

[17] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*. ACM, 165–174.

[18] Xiang Wang, Hongye Jin, An Zhang, Xiangnan He, Tong Xu, and Tat-Seng Chua. 2020. Disentangled Graph Collaborative Filtering. In *SIGIR*. ACM, 1001–1010.

[19] Jiancan Wu, Xiang Wang, Fuli Feng, Xiangnan He, Liang Chen, Jianxun Lian, and Xing Xie. 2021. Self-supervised Graph Learning for Recommendation. In *SIGIR*. ACM, 726–735.

[20] 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 *KDD*. ACM, 974–983.

[21] Junliang Yu, Hongzhi Yin, Min Gao, Xin Xia, Xiangliang Zhang, and Nguyen Quoc Viet Hung. 2021. Socially-Aware Self-Supervised Tri-Training for Recommendation. In *KDD*. ACM, 2084–2092.

[22] 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*. ACM, 1294–1303.

[23] Jinghao Zhang, Yanqiao Zhu, Qiang Liu, Shu Wu, Shuhui Wang, and Liang Wang. 2021. Mining Latent Structures for Multimedia Recommendation. In *ACM Multimedia*. ACM, 3872–3880.

[24] Wayne Xin Zhao, Yupeng Hou, Xingyu Pan, Chen Yang, Zeyu Zhang, Zihan Lin, Jingsen Zhang, Shuqing Bian, Jiakai Tang, Wenqi Sun, Yushuo Chen, Lanling Xu, Gaowei Zhang, Zhen Tian, Changxin Tian, Shanlei Mu, Xinyan Fan, Xu Chen, and Ji-Rong Wen. 2022. RecBole 2.0: Towards a More Up-to-Date Recommendation Library. In *CIKM*. ACM, 4722–4726.

[25] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, Yingqian Min, Zhichao Feng, Xinyan Fan, Xu Chen, Pengfei Wang, Wendi Ji, Yaliang Li, Xiaoling Wang, and Ji-Rong Wen. 2021. RecBole: Towards a Unified, Comprehensive and Efficient Framework for Recommendation Algorithms. In *CIKM*. ACM, 4653–4664.

[26] Jieming Zhu, Quanyu Dai, Liangcai Su, Rong Ma, Jinyang Liu, Guohao Cai, Xi Xiao, and Rui Zhang. 2022. BARS: Towards Open Benchmarking for Recommender Systems. In *SIGIR*. ACM, 2912–2923.