GNN4FR: A Lossless GNN-based Federated Recommendation Framework

Abstract

Graph neural networks (GNNs) have gained wide popularity in recommender systems due to their capability to capture higher-order structure information among the nodes of users and items. However, these methods need to collect personal interaction data between a user and the corresponding items and then model them in a central server, which would break the privacy laws such as GDPR. As a result, I will reproduce a classic GNN-based model named LightGCN. Furthermore, we will apply our lossless federated framework named GNN4FR to it for the privacy-preserving.

Keywords: Lossless, Federated Recommendation, GNN, LigthGCN

1 Introduction

Recommender systems have played an important role in our lives, which are used to help users filter out the information they are not interested in. GNNs are widely used in personalized recommendation methods as they are able to capture high-order interactions between users and items in a user-item graph, enhancing user and item representations [2, 4, 15, 16, 19, 20]. However, these methods face challenges in terms of privacy laws, such as GDPR [14] as they require the collection and modeling of personal data in a central server.

Constructing the global graph using all users' subgraphs is often not allowed. Therefore, existing works [12, 17] just expand a user's local graph to exploit high-order information.

In this paper, we propose the first lossless federated framework named GNN4FR, which can accommodate almost all existing graph neural networks (GNNs) based recommenders. The contributions of this paper are summarized as follows:

- We propose a novel lossless federated framework for GNN-based methods, which enables the training process to be equivalent to the corresponding un-federated counterpart.
- We propose an "expanding local subgraph + synchronizing user embedding" mechanism to achieve full-graph training.
- We choose LightGCN [6] as an instantiation of our framework to demonstrate its equivalence.

2 Related works

2.1 Federated Recommendation

Recommendation systems have seen significant growth in today's society. However, the training and inference processes of these models heavily rely on users' personal data, which raises concerns about privacy. With the introduction of GDPR [14], the need for privacy and security in the recommendation domain led to the emergence of federated learning [18]. This approach aims to address privacy issues through decentralized model training. In the field of recommendation, several frameworks have been developed to enable federated learning [1, 3, 5, 7–12, 17]. For instance, FCF [1] and FedMF [3] are specifically designed for factorization-based recommendation models. When it comes to GNN-based recommendation models, there are some existing frameworks such as FedGNN [17] and SemiDFEGL [12]. However, these frameworks are not able to construct a global graph without resorting to other entities or information, leading to some loss of the high-order structure information.

2.2 GNN-based Recommendation

Graph neural networks (GNNs) have gained wide popularity in recommender systems due to their capability to capture higher-order information. Notable examples include NGCF [16] and LightGCN [6]. NGCF employs a 3-hop graph neural network to learn user and item embeddings. Subsequently, LightGCN improves upon NGCF by eliminating redundant components and achieving superior results. However, these methods are centralized and rely on collecting user information through the server to construct a global graph. Yet, privacy concerns make it challenging for the server to collect user data for graph construction. To address this limitation, we propose a novel framework that enables the server to construct the global graph using distributed user data, while ensuring user privacy protection. This approach achieves equivalent performance to centralized graph model training, making it the first lossless GNN-based federated recommendation framework to date.

3 Method

3.1 LightGCN

LightGCN, a simplified Graph Convolution Network (GCN) model specifically designed for recommendation systems. The key innovation of LightGCN is its streamlined architecture, which focuses solely on neighborhood aggregation for collaborative filtering, omitting common GCN components like feature transformation and nonlinear activation. This simplification results in a more efficient model that's easier to train and implement. LightGCN shows significant improvements over the more complex Neural Graph Collaborative Filtering (NGCF) model, offering about 16 precent average improvement in performance. The paper also provides extensive empirical and analytical validation for LightGCN's design choices.

3.2 GNN4FR

Algorithm 1 GNN4FR

Initialize(), i.e., Algorithm 2
ExpandLocalGraph(), i.e., Algorithm 3
for $t = 1, 2, \ldots, T$ do
  ForwardPropagation(), i.e., Algorithm 4
  for each client u in parallel do
    constructs the local loss function
    computes the gradient of the local loss w.r.t. the nodes of the final layer
  end for
  BackwardPropagation(), i.e., Algorithm 5
  $\theta = \theta - \gamma \nabla_{\theta}$
end for=0

This section describes our proposed GNN4FR in detail, i.e., Algorithm 1. The training process contains five major parts. Firstly, do pre-training preparations, including initializing item embeddings, etc. Secondly, expand the subgraphs of all clients. Thirdly, use an "expanding local subgraph + synchronizing user embedding" mechanism for forward propagation. Fourthly, pass back neighboring users' embedding gradients. Fifthly, use secret sharing to aggregate gradients. In order to make it more comprehensive, we illustrate this process using a specific example to enhance clarity.

In the example, there are three clients and four items, and their interactions are shown in Figure 1. Besides, the number of GNN convolution layers is three. For doing pre-training preparations, we show

Algorithm 2 Initialize

for each client u in parallel do
  constructs the local subgraph $\mathcal{G}_u$
  initializes parameters (i.e., user embedding and item embeddings) using the same seed
  generates key pairs including the public key PB_u and the private key PI_u
  sends PB_u to the server
end for
the server collects PB_u from each client u, and sends to the client u_c (i.e., the user randomly chosen by the server)
the client u_c receives the public keys from the server, encrypts the shared key S with each PB_u, and sends them to the server
the server receives the ciphertext and forwards them to the corresponding clients
for each client u in parallel do
  receives the ciphertext of S
  decrypts it with PI_u and obtains the shared key S
end for=0

the process in Algorithm 2. Firstly, each client constructs his or her local subgraph and initializes the

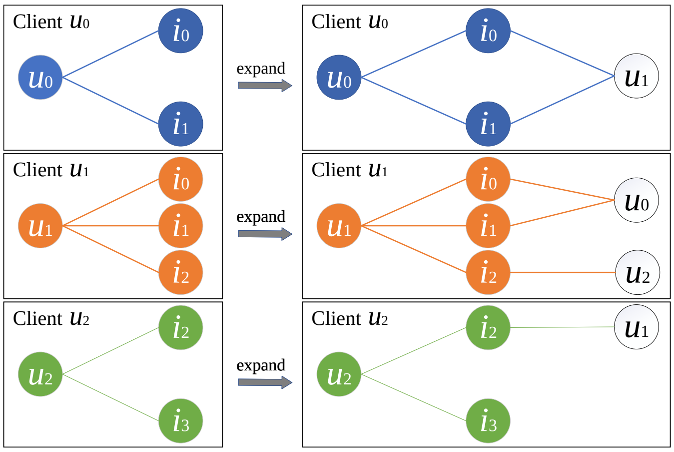

Figure 1. Illustration of the process that each client expands the local subgraph in the example.

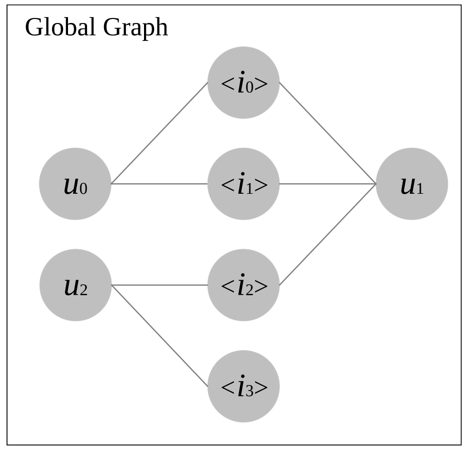

Figure 2. Illustration of the process that the global graph constructed by the server. Notice that $< i_1 >$ means the ciphertext of i_1.

user embedding and item embeddings using the same seed, which means that the embedding of item i_0 in client u_0 is equal to the embedding of item i_0 in client u_1. Secondly, each client generates a key pair (i.e., a public key and a private key) and sends the public key to the server. Thirdly, the server collects the public keys of all clients, then randomly chooses one client u_c from all clients (here we suppose that u_c is u_1), and sends all the public keys to it (i.e., u_1). Fourthly, the client u_1 generates a shared key S, encrypts it with all the public keys received from the server (i.e., public keys of u_2 and u_3), and sends all the ciphertext to the server. Fifthly, the server receives all the ciphertext and forwards them to the corresponding clients. Finally, each client receives the ciphertext of the shared key S and decrypts it with his or her own private key. This means that the clients u_1, u_2, and u_3 would hold the common shared keys S, but the server did not know it.

For expanding the local subgraph, we show the process in Algorithm 3. Firstly, each client encrypts the ID of interacted items with the shared key S and sends them to the server. Secondly, the server receives the ciphertext from all clients, then constructs the global graph by comparing them, which is shown in Figure 2. For example, suppose there is a common item i_0 which is interacted with by two clients u_0 and u_1. Encrypting the same content with the same key will result in identical output. Therefore, the server knows that the clients u_0 and u_1 have a common item i_0, but could not know what item i_0 is due to just knowing the ciphertext of the item ID of i_0, which protects the privacy. Thirdly, for each client, the server sends the neighboring users-IDs and exclusive items (i.e., items that are only interacted with by the client, for instance, item i_3 is the exclusive item of client u_2) to them and informs the connectivity between neighboring users and common items. Finally, each client expands the local subgraph, as shown in Figure 1.

For forward propagation, we show the process in Algorithm 4. Firstly, synchronize users' embeddings. As shown in step 1 in Figure 3, each client sends the user embedding to the neighboring users through the server. The transmission policy is that the sender encrypts and the receiver decrypts with the same shared key S. Since the server can receive the ciphertext in the process, but does not have

Algorithm 3 ExpandLocalSubgraph

```
for each client $u$ in parallel do
    encrypts $\mathcal{I}_u$ (i.e., the set of item-IDs which are interacted by user $u$) with $S$
    sends $<\mathcal{I}_u>$ (i.e., the cyphertext of $\mathcal{I}_u$) to the server
end for
//Server
receives the ciphertext from all clients
constructs the global graph by comparing the ciphertext
for $u \in \mathcal{U}$ do
    sends the neighboring users $\mathcal{N}_u$ and $\mathcal{I}_u^e$ (i.e., the set of items which are only interacted by user $u$) to the client $u$
    informs the client $u$ of the connectivity between neighboring users and common items
end for
for each client $u$ in parallel do
    receives the information from the server
    expands the local subgraph
end for=0
```

the key to decrypt it, the privacy is protected. Secondly, each client convolves the local subgraph to get the user embedding and item embeddings of layer 1. Notice that we do not get the neighboring users' embeddings by convolution, but receive them from the corresponding clients. i.e., step 3 in Figure 4. Thirdly, each client convolves the local subgraph to get the user embedding and item embeddings of layer 2, i.e., the last layer.

For constructing the local loss and backward propagation, we show the process in Algorithm 5. Firstly, each client constructs a local loss with user embedding and item embeddings, which does not require the embedding of the neighboring users. Secondly, backward propagation. As shown in step 1 in Figure 4, each client backpropagates to get the user embedding and item embedding gradients. Then

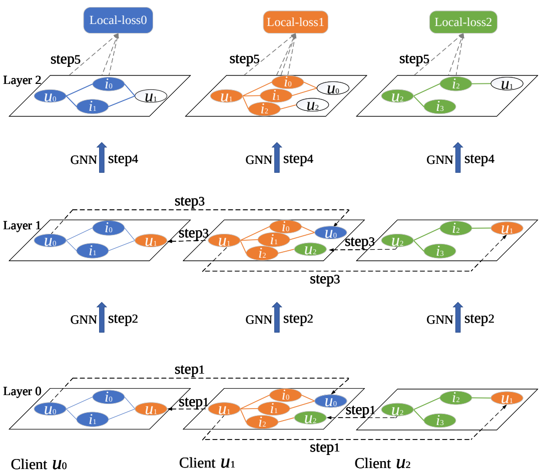

Figure 3. Illustration of forward propagation.

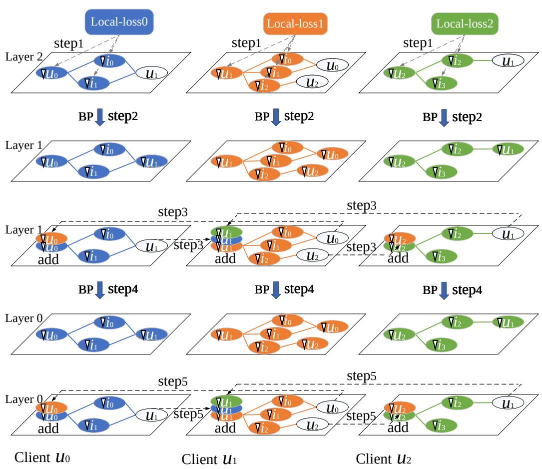

Figure 4. Illustration of backward propagation.

Algorithm 4 ForwardPropagation

for $l = 0, 1, \ldots, L-1$ do
  for each client u in parallel do
    encrypts the user embedding of user u in l-th layer $U_u^{[l]}$ with S
    sends them to the server
  end for
  the server receives the ciphertext from all clients and forwards them to the corresponding clients
  for each client u in parallel do
    receives the ciphertext
    decrypts $U_u^{[l]}$ with S
    updates the l-th layer neighboring user embeddings in the local subgraph
    convolves the local subgraph at l-th layer to get the $(l+1)$-th layer user embedding and item embeddings(except for the neighboring users' embeddings)
  end for
end for=0

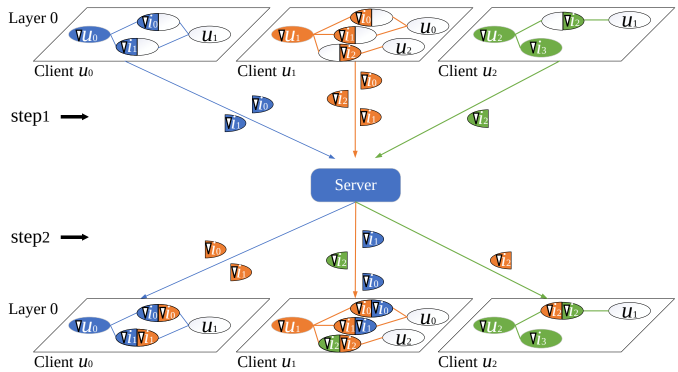

Figure 5. Illustration of sending one part of gradients.

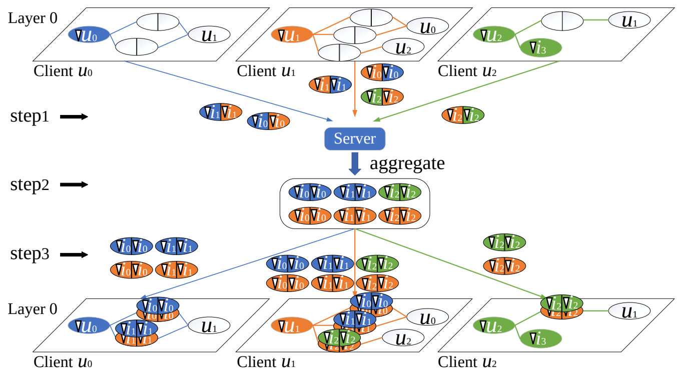

Figure 6. Illustration of aggregating gradients.

backpropagate again to get the gradient of all nodes at layer 1. The gradient of the neighboring users would be sent back to the corresponding client, i.e., step 3 in Figure 4. For example, the gradient of user embedding in client u_1 consists of three parts, one from client u_0, one from client u_2, and the final one from itself. Similarly, continue to backpropagate and finally get the embedding gradient of the user at layer 0 and the embedding gradient of the items.

Notice that only the gradients of the item embeddings (except the exclusive items and user embedding of each client) need to be aggregated. Here we use the secret sharing technology [13] to protect privacy. Firstly, each client splits the aggregated gradients into two parts randomly, chooses one to encrypt with the shared key S, and sends them to the one of neighboring clients by the server, as shown in Figure 5. Secondly, each client decrypts the ciphertext and adds them to the corresponding nodes. Finally, as shown in Figure 6, each client sends the gradients to the server for aggregation.

We now introduce how to make a prediction (i.e., Algorithm 6). Firstly, for each item, the server

Algorithm 5 BackwardPropagation

for $l = L, L-1, \ldots, 1$ do
  for each client u in parallel do
    computes $\nabla_u^{[l-1]}$ with $\nabla_u^{[l]}$ (i.e., the gradient of $U_u^{[l]}$)
    encrypts $\nabla_{\mathcal{N}_u}^{[l-1]}$ (i.e., the gradient of $U_{\mathcal{N}_u}^{[l]}$) with S
    sends them to the server
  end for
  the server receives the ciphertext from all clients and forwards them to the corresponding clients
  for each client u in parallel do
    receives the ciphertext
    decrypts them with S
    adds them to $\nabla_u^{[l-1]}$ (i.e., the gradient of all nodes w.r.t. user u in the l-th layer)
  end for
end for=0

Algorithm 6 Predict

for each client u in $\mathcal{U}_I^{'}$ (i.e., the set of $u_i^{'}$, $i \in \mathcal{I}$) ($u_i^{'}$ means the user randomly chosen by the server in the set of users who interact with item i) do
  encrypts V_u (i.e., the set of item embeddings that need to be uploaded by the client u) with S
  sends them to the server
end for
the server collects the ciphertext of all item embeddings and then sends the negative item embeddings (i.e., the items which are not interacted with by the user) to all clients
for each client u in parallel do
  receives the ciphertext of negative item embeddings
  decrypts them with the shared key S
  calculates the predicted scores with user embedding and item embeddings
end for=0

randomly selects one user from those who have interacted with it to send its embedding encrypted with the shared key S. Secondly, the server collects the ciphertext of all item embeddings. Then for each client, the server sends the embeddings of the negative items, which refer to the items that the client has not previously interacted with. Finally, each client calculates the predicted scores with user embedding and item embeddings.

4 Implementation details

4.1 Comparing with the released source codes

This section must be filled. If no related source codes are available, please indicate clearly. If there are any codes referenced in the process, please list them all and describe your usage in detail, highlighting your own work, creative additions, noticeable improvements and/or new features. The differences and advantages must be dominant enough to demonstrate your contribution. The source code of LightGCN is available. However, we replicated the code with another dataset named Movielens 100k. Besides, in order to protect the privacy of users, we federated LightGCN with GNN4FR which is first introduced by us.

4.2 Dataset and Evaluation Metrics

We use the files u1.base and u1.test of MovieLens 100K as our training data and test data, respectively. MovieLens 100K contains 943 users and 1682 items. The u1.base file contains 80000 rating records, and its density is 5.04%. The u1.test file contains 20000 rating records. We follow the common practice and keep the (user, item) pairs with ratings 4 or 5 in u1.base and u1.test as preferred (user, item) pairs, and remove all other records.

We use two commonly used evaluation metrics, i.e., Precision@N and Recall@N, where N is the number of recommended items.

Table 1. Experimental results of the proposed federated method GNN4FR and its un-federated version LightGCN.

Model	Precision@5	Recall@5
LightGCN	0.3816	0.1257
GNN4FR	0.3816	0.1257

5 Results and analysis

We use LightGCN to instantiate an example of our framework and report the results in Table 1. The training process and test results are equivalent to the corresponding un-federated counterpart. Notice that we do not include more datasets and baselines because the purpose is to show the equivalence between the proposed framework and the un-federated counterpart, which is different from that of empirical studies in most works.

6 Conclusion and future work

In this paper, we propose the first lossless GNN-based federated recommendation framework named GNN4FR, which uses an "expanding local subgraph + synchronizing user embedding" mechanism to achieve full-graph training, enabling the training process to be equivalent to the corresponding un-federated approach. In addition, we leverage secret sharing to protect privacy while aggregating the gradients. Finally, we use LightGCN to instantiate an example of our framework and show its feasibility and equivalence. For future work, we aim to develop diverse models using GNN4FR that can accommodate specific algorithms while achieving a good balance between accuracy and efficiency.

References

[1] Muhammad Ammad-Ud-Din, Elena Ivannikova, Suleiman A Khan, Were Oyomno, Qiang Fu, Kuan Eeik Tan, and Adrian Flanagan. Federated collaborative filtering for privacy-preserving personalized recommendation system. arXiv preprint arXiv:1901.09888, 2019.

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

[3] Di Chai, Leye Wang, Kai Chen, and Qiang Yang. Secure federated matrix factorization. IEEE Intelligent Systems, 36(5):11–20, 2020.

[4] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In 2019 World Wide Web Conference, WWW 2019, pages 417–426. Association for Computing Machinery, Inc, 2019.

[5] Adrian Flanagan, Were Oyomno, Alexander Grigorievskiy, Kuan E Tan, Suleiman A Khan, and Muhammad Ammad-Ud-Din. Federated multi-view matrix factorization for personalized recommendations. In Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part II, pages 324–347. Springer, 2021.

[6] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 639–648, 2020.

[7] István Hegedűs, Gábor Danner, and Márk Jelasity. Decentralized recommendation based on matrix factorization: A comparison of gossip and federated learning. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 317–332. Springer, 2019.

[8] Junyi Li and Heng Huang. Fedgrec: Federated graph recommender system with lazy update of latent embeddings. arXiv preprint arXiv:2210.13686, 2022.

[9] Zhiwei Liu, Liangwei Yang, Ziwei Fan, Hao Peng, and Philip S Yu. Federated social recommendation with graph neural network. ACM Transactions on Intelligent Systems and Technology (TIST), 13(4):1–24, 2022.

[10] Sichun Luo, Yuanzhang Xiao, and Linqi Song. Personalized federated recommendation via joint representation learning, user clustering, and model adaptation. In Proceedings of the 31st ACM International Conference on Information and Knowledge Management, pages 4289–4293, 2022.

[11] Tao Qi, Fangzhao Wu, Chuhan Wu, Yongfeng Huang, and Xing Xie. Privacy-preserving news recommendation model training via federated learning. arXiv preprint arXiv:2003.09592, 2020.

[12] Liang Qu, Ningzhi Tang, Ruiqi Zheng, Quoc Viet Hung Nguyen, Zi Huang, Yuhui Shi, and Hongzhi Yin. Semi-decentralized federated ego graph learning for recommendation. arXiv preprint arXiv:2302.10900, 2023.

[13] Adi Shamir. How to share a secret. Communications of the ACM, 22(11):612–613, 1979.

[14] Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). A Practical Guide, 1st Ed., Cham: Springer International Publishing, 10(3152676):10–5555, 2017.

[15] Xiang Wang, Xiangnan He, Yixin Cao, Meng Liu, and Tat-Seng Chua. Kgat: Knowledge graph attention network for recommendation. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 950–958, 2019.

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

[17] Chuhan Wu, Fangzhao Wu, Lingjuan Lyu, Tao Qi, Yongfeng Huang, and Xing Xie. A federated graph neural network framework for privacy-preserving personalization. Nature Communications, 13(1):3091, 2022.

[18] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. ACM Transactions on Intelligent Systems and Technology (TIST), 10(2):1–19, 2019.

[19] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 974–983, 2018.

[20] Jiani Zhang, Xingjian Shi, Shenglin Zhao, and Irwin King. Star-gcn: Stacked and reconstructed graph convolutional networks for recommender systems. arXiv preprint arXiv:1905.13129, 2019.