Federated Unlearning for On-Device Recommendation

Wei Yuan
The University of Queensland
Brisbane, Australia
w.yuan@uq.edu.au

Hongzhi Yin*
The University of Queensland
Brisbane, Australia
h.yin1@uq.edu.au

Fangzhao Wu
Microsoft Research Asia
Beijing, China
wufangzhao@gmail.com

Shijie Zhang
Tencent
Shenzhen, China
julysjzhang@tencent.com

Tieke He
Nanjing University
Nanjing, China
hetieke@gmail.com

Hao Wang
Alibaba Cloud, Alibaba Group
Hangzhou, China
cashenry@126.com

ABSTRACT

The increasing data privacy concerns in recommendation systems have made federated recommendations attract more and more attention. Existing federated recommendation systems mainly focus on how to effectively and securely learn personal interests and preferences from their on-device interaction data. Still, none of them considers how to efficiently erase a user's contribution to the federated training process. We argue that such a dual setting is necessary. First, from the privacy protection perspective, "the right to be forgotten (RTBF)" requires that users have the right to withdraw their data contributions. Without the reversible ability, federated recommendation systems risk breaking data protection regulations. On the other hand, enabling a federated recommender to forget specific users can improve its robustness and resistance to malicious clients' attacks.

To support user unlearning in federated recommendation systems, we propose an efficient unlearning method FRU (*F*ederated *R*ecommendation *U*nlearning), inspired by the log-based rollback mechanism of transactions in database management systems. It removes a user's contribution by rolling back and calibrating the historical parameter updates and then uses these updates to speed up federated recommender reconstruction. However, storing all historical parameter updates on resource-constrained personal devices is challenging and even infeasible. In light of this challenge, we propose a small-sized negative sampling method to reduce the number of item embedding updates and an importance-based update selection mechanism to store only important model updates. To evaluate the effectiveness of FRU, we propose an attack method to disturb federated recommenders via a group of compromised users. Then, we use FRU to recover recommenders by eliminating these users' influence. Finally, we conduct extensive experiments on two real-world recommendation datasets (i.e. MovieLens-100k and Steam-200k) with two widely used federated recommenders to show the efficiency and effectiveness of our proposed approaches.

CCS CONCEPTS

• **Information systems** → **Collaborative filtering**.

KEYWORDS

Federated Recommender System, Machine Unlearning

ACM Reference Format:
Wei Yuan, Hongzhi Yin, Fangzhao Wu, Shijie Zhang, Tieke He, and Hao Wang. 2023. Federated Unlearning for On-Device Recommendation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Recommender Systems (RS) suggest the most appropriate items and services to users by analyzing the collected personal data, e.g. user-item interactions and user profiles [7, 27, 47]. With the growing awareness of privacy and the recent publishing of data privacy protection regulations such as the General Data Protection Regulation (GDPR) [36] in the European Union and the California Consumer Privacy Act (CCPA) [13] in the United States, collecting, storing, and using users' data is becoming harder. To address the above challenges, more and more researchers focus on applying federated learning [28] to recommendation systems (FedRecs), which train recommender models on client devices without sharing user data with a central server or other clients.

Since Ammad et al. [1] proposed the first federated recommendation framework, FedRecs have made great advancements recently [18]. For example, Muhammad et al. [29] proposed FedFast to accelerate training convergence. Lin et al. [23] investigated how to exploit explicit feedback. Liang et al. [22] attempted to improve the security of FedRecs via denoising techniques. Wu et al. [40] incorporated GNN into a general FedRec framework.

Despite great advancements made in this area, it has been unexplored how to forget specific users during the FedRec training process. Without the ability to erase specific users' contributions to the federated training process, FedRec might break privacy protection laws or regulations such as CCPA and GDPR that give users the right to control and withdraw their data at any time. Apart from reducing the risks of breaking privacy protection rules, implementing unlearning is also important to improve FedRec's robustness

*Corresponding author.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

and resistance to malicious attacks in an open setting where any user/device can participate in the training process of FedRec. Recent studies [33, 46] have shown that current FedRecs are still not "safe" enough when facing malicious users' attacks. After detecting such attacks, the ability to efficiently erase such malicious users' influence without retraining from scratch is essential for FedRecs.

Although some recent works [5, 21] tried to apply *machine unlearning* to recommender systems because of data privacy concerns, all of them focus on the traditional centralized recommenders. Their methods require accessing the whole training data during unlearning, which is prohibitive in FedRecs. Some works [24–26, 41] explored unlearning in federated learning, however, they are tailored for classification tasks in Computer Vision (CV) area. To erase the contributions of target clients, the most naive yet effective method is to retrain the recommender model from scratch after removing the target clients, which is infeasible in the real-world recommendation setting due to its huge time and resource costs. Another alternative is to continue the training after removing the target clients. However, such a method cannot guarantee whether and when these target users' influence on the global parameters (e.g. item embeddings) will be erased. As a result, how to effectively and efficiently erase target clients' contributions is not trivial in FedRecs.

Inspired by the log-based rollback mechanism of transactions in database management systems (DBMS), we propose to record each client's historical model updates. Once we need to erase some users' contributions, we will roll back and reconstruct the other clients' models according to their training logs (i.e. their historical model updates). To achieve that, the most intuitive way is to keep all clients' historical model updates at the central server. This method can work when the number of clients is small, such as in the federated classification setting in which there are only tens of clients [24]. However, in FedRec, the number of clients is several orders of magnitude larger than the classification settings. As the storage costs at the central server increase linearly with the number of clients, this naive method is unsustainable and impractical for FedRec. Therefore, we propose to retain historical model updates at each client's local device, and storage costs at each client device decouple with the number of clients. Still, it is non-trivial to store each client's historical model updates on the resource-constrained device, and it is infeasible to simply store all updates.

In this paper, we propose FRU (*F*ederated *R*ecommendation *U*nlearning), a simple yet effective federated recommendation unlearning method. FRU is model-agnostic and can be applied to most federated recommendation systems. The basic idea of FRU is to erase a target client's influence by revising FedRec's historical updates and leveraging the revised updates to speed up FedRec reconstruction. Compared with completely retraining (reconstructing) the FedRec from scratch, FRU requires less running time and achieves even better model performance. FRU stores each client's historical updates locally on decentralized personal devices to avoid high central server storage overhead. To efficiently utilize the limited storage space on each client's device, we design two novel components: a user-item mixed semi-hard negative sampling component and an importance-based update selection component. The user-item mixed negative sampling exploits high-quality negative samples to train FedRec, reaching comparable model performance with fewer negative samples than the traditional sampling method. Consequently, it reduces the size of item embedding updates at each client. The importance-based update selection component dynamically chooses important updates to store on each client device at each training epoch, instead of storing all parameter updates.

After achieving unlearning in FedRec, evaluating the effectiveness of unlearning is neither easy because there are a large number of users in recommendation datasets, and many of them have common items and similar preferences (actually, it is the base of CF-based recommendation methods). Deleting a small portion of normal users/clients will not significantly change the performance of FedRec. In light of this, we propose an attack method to destroy the FedRec with a group of compromised clients/users (also called malicious users). An effective unlearning method should recover the destroyed FedRec quickly and achieve comparable or even better performance than training without malicious users.

To demonstrate the effectiveness of our proposed approach, we choose two commonly used recommenders [45], Neural Collaborative Filtering (NCF) [17] and LightGCN [16], with the most basic federated learning protocol [1] as our base models. Then, we conduct extensive experiments with these base models on two real-world recommendation datasets, MovieLens-100k and Steam-200k. The experimental results show that FRU can erase the influence of removed malicious users, with at least 7x speedup compared with the naive retraining from scratch.

The main contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work to investigate machine unlearning in federated recommender systems, enabling FedRecs to effectively erase the influence of specific users/clients and efficiently finish recovering.
- We propose FRU, an unlearning method tailored for FedRecs. It stores each client's historical changes locally on their devices. To improve the storage space efficiency on resource-constrained devices, we propose a novel negative sampling method and an importance-based update selection mechanism. Then, FRU rolls back FedRecs to erase the target users'/clients' influence and fast recover FedRecs by calibrating the historical model updates.
- We design an attack method to intuitively evaluate the unlearning effectiveness. The experimental results demonstrate the effectiveness and efficiency of FRU on two real-world datasets. Comprehensive ablation studies reveal the effectiveness and importance of each technical component in FRU.

2 PRELIMINARIES

2.1 Federated Recommendation

Let $\mathcal{V}$ and $\mathcal{U}$ denote the sets of items and users (clients), respectively. The sizes of items and users are $|\mathcal{V}|$ and $|\mathcal{U}|$. Each user u_i owns a local training dataset $\mathcal{D}_i$, which contains user-item interactions (u_i, v_j, r_{ij}). $r_{ij} = 1$ represents u_i has interacted with item v_j, and $r_{ij} = 0$ means no interactions exist between u_i and v_j (i.e. negative samples). We denote the set of all negative instances for u_i as $\mathcal{V}_{neg}(i)$. The federated recommender is trained to predict the score of $\hat{r}_{ij}$ between u_i and all non-interacted items. And then, according to the predicted scores $\hat{r}_{ij} \in [0, 1]$, the federated recommender

outputs an item list $\hat{\mathcal{V}}_i$ for each user u_i by selecting top K ranked items w.r.t. $\hat{r}_{ij}$.

With FedRec, users can keep their private data locally and do not need to share it with others. To achieve this goal, a central server is often employed to coordinate individual clients. At first, the central server randomly selects a batch of users/clients and dispenses the global parameters to these clients. Each client combines its received global parameters and its local user embedding as a local recommender. Then, the local recommender is optimized on local data using a certain objective function (e.g. BPR loss [32]). After multiple local training epochs, each client sends the updated global parameters (or global parameter updates) back to the server and keeps the updated private parameters locally. The server aggregates received global parameter updates with a certain aggregation strategy [28]. The above steps, namely the global training round, are repeated until the system satisfies the pre-defined requirements.

2.2 Machine Unlearning of Federated Recommendation

Machine unlearning (also referred to as data removal or data deletion [10, 12], selective forgetting [11]) aims to actively clean the influence of a specific subset of training data. In this paper, we apply machine unlearning to remove the contributions of a subset of users $\mathcal{U}'$ in FedRecs. Specifically, let f_{ori} be the original federated recommender, and after applying unlearning to f_{ori}, the new FedRec f_{unl} is expected to erase all contributions from $\mathcal{U}'$.

Evaluating the effectiveness of unlearning is a big challenge. Theoretically, retraining the federated recommender model from scratch after removing $\mathcal{U}'$ can get an ideal model f_{ret} that is guaranteed to completely forget the influence of $\mathcal{U}'$. But we cannot directly compare the model parameters between the unlearned model and the fully retrained model as there are too many stochastic processes during federated training. Therefore, the common practice is to take f_{ret} as the baseline model and evaluate the effectiveness and efficiency of the proposed unlearning approach in some attack scenarios [35, 42]. In this paper, we also follow this protocol. Specifically, we design an adversary to disturb the FedRec's training process via a group of malicious users. Then, we assess and analyze *(1) whether FRU can recover the destroyed FedRec* and *(2) the recovery efficiency (i.e., time cost and space costs) compared with some baselines.*

3 FEDERATED RECOMMENDATION UNLEARNING (FRU)

3.1 Overview

To fast unlearn target users, FRU calibrates all users' historical model updates and aggregates these updates to reconstruct FedRec models. Algorithm 1 describes the overall work flow of FRU in a general Fedrec framework. FRU contains two parts: efficient on-device update storing (Algorithm 1 Line 16, 19-20) and update revision for fast recovery (Algorithm 1 Line 3-5, 23-38). The efficient on-device update storing aims to record each client's historical updates on its resource-constrained device. It contains two components: a user-item mixed negative sampling method to reduce the size of updated parameters and an importance-based update filtering to dynamically clean less important historical updates. Based on the stored historical updates, FRU can roll back the FedRec to the point when the deleted user/client joins the FedRec and then calibrate the historical updates from that time point to recover the FedRec.

In the following subsections, we present the details of FRU. For clarity, we use $\mathcal{M}(V)_k^t$ and $\mathcal{M}(U)_k^t$ to represent the updates of global (or public) parameters (e.g. item embeddings) and local (or private) parameters (e.g. user embeddings) of user u_k at tth global round. $\mathcal{M}(U)_k^t$ is not submitted to the central server for privacy concerns, while $\mathcal{M}(V)_k^t$ is uploaded to the server and the server will update the global parameters by aggregating these clients' updates.

3.2 Base Federated Recommenders

To present the generalization ability of FRU, we choose the two most commonly used recommenders (NCF [17] and LightGCN [16]) as our base models and train them with the most typical federated learning protocol [1].

Neural collaborative filtering (NCF [17]) extends collaborative filtering (CF) by leveraging an L-layer feedforward network (FFN) to model the complex relationship between users and items.

$$\hat{r_{ij}} = \sigma(h^\top FFN([\mathrm{u}_i, \mathrm{v}_j])) \tag{1}$$

where u_i and v_i are user and item embeddings, respectively. $[\cdot]$ is concatenation operation.

As a graph-based recommender, LightGCN [16] treats each user and item as a distinct node. The user-item interactions can be viewed as a bipartite graph. User and item embeddings are learned and updated by propagating their neighbors' embeddings:

$$\mathrm{u}_i^l = \sum_{j \in \mathcal{N}_{u_i}} \frac{1}{\sqrt{|\mathcal{N}_{u_i}|}\sqrt{|\mathcal{N}_{v_j}|}} \mathrm{v}_j^{l-1}, \mathrm{v}_j^l = \sum_{i \in \mathcal{N}_{v_j}} \frac{1}{\sqrt{|\mathcal{N}_{v_j}|}\sqrt{|\mathcal{N}_{u_i}|}} \mathrm{u}_i^{l-1} \tag{2}$$

where l is the propagation layer in LightGCN. Note that under FedRec settings, the embeddings of neighbor users are not accessible. Therefore, each client can only use the local user-item interaction bipartite graph to update user and item embeddings.

The federated learning protocol is as follows. The user embedding $\mathrm{u_i}$ is private, initialized and maintained at each client. The item embeddings V and other global model parameters are initialized and sent from a central server. After receiving item embeddings and other global parameters, each client combines its local user embedding $\mathrm{u_i}$ and these global parameters to form a local recommender model and then updates these model parameters on the local training data. After local updating, each client sends the updated global parameters back to the server and keeps the updated user embedding $\mathrm{u_i}$ locally. The server applies the average aggregation for these global parameters to gain the new global parameters.

3.3 Efficient On-Device Update Storing

Compared with retraining from scratch, FRU speeds up the reconstruction process by accurately rolling back the FedRec and then calibrating the stored model updates. FRU takes advantage of storing historical model updates (i.e., the log) to avoid retraining from scratch. As mentioned in Section 1, we choose to store historical model updates on local devices since centrally storing all clients' historical updates will incur unaffordable storage costs when the

number of clients is large [37]. Therefore, how efficiently utilizing the limited storage space for the historical model updates at each client device is the key.

Generally, the item embeddings dominate the size of a local recommender model in the FedRec. Based on this observation, we mainly focus on how to reduce the size of item embedding updates on each device. This paper proposes an importance-based update selection mechanism to store only important item embedding updates and a user-item mixed semi-hard negative sampling method to reduce the number of negative instances at each client.

3.3.1 ***Importance-based Update Selection.*** Instead of storing the updates of the whole item embedding table on each client device, the importance-based update selection mechanism only stores the embedding updates of the client's interacted items and the sampled negative items, which are a tiny portion of the whole item set. Then, FRU further reduces the storage costs by ignoring non-significant updates. Intuitively, the influences of a client/user on different items' embeddings are different. For example, some items are very popular, and their embeddings are updated by many users. In this case, a single user can only have a very limited impact on these items. Another example is that when an item's embedding is already well-trained, a user can only make an ignorable effect. For these items whose embedding updates are so small at a client, FRU will not store them to reduce the storage costs on the device. Specifically, for a client/user, FRU only stores the top α proportion of item embedding updates based on their updates' significance $\|\mathcal{M}(\mathrm{v}_i)_k^t\|$, where $\mathcal{M}(\mathrm{v}_i)_k^t = \mathrm{v}_i^t - \mathrm{v}_i^{t-1}$. v_i^{t-1} is the received embedding of item v_i from the central server, and v_i^t is its updated embedding on the local training data $\mathcal{D}_k$ of client u_k.

3.3.2 ***User-item Mixed Semi-hard Negative Sampling.*** As mentioned in Section 3.3.1, a client only needs to store the embedding updates of its associated items. The associated items include the client's interacted items and sampled negative items. In this part, we propose a more efficient negative sampling method to reduce the number of negative samples at each client without compromising the model's performance so that FRU can further alleviate the log storage space.

For each user, the traditional negative sampling method randomly selects a small portion of its non-interacted items as negative samples. However, some sampled negative items may not be informative enough to optimize the recommendation model. In a centralized recommendation system, some recent studies [39, 44] show that hard negatives contain more helpful information for model optimization and convergence. They proposed two methods to sample hard negatives based on user embeddings [44] and adversarial model [39]. However, these centralized negative sampling methods cannot be directly adopted in FedRecs. Specifically, the adversarial method brings much extra computation overhead, and each user's local data is highly sparse and biased. Sampling hard negatives based on user embedding is also unreliable in FedRecs because only a small portion of users are selected to train at each global epoch, and the frequency of updating user embeddings is lower than centralized recommenders. Consequently, how to retrieve high-quality negative samples from the item pool is still challenging for FedRecs.

In this paper, we propose a user-item mixed semi-hard negative sampling strategy. Specifically, we choose semi-hard negative samples from both user and item sides. The sampling from the user side is the same as in traditional centralized recommenders [44]. We calculate the relevance between the user's embedding and each item's embedding in the candidate pool to choose hard negative samples for each client. As mentioned before, the frequency of updating user embedding is lower than centralized recommenders, so the user embedding needs to take a relatively long time to be informative. To compensate for the unreliability of user embeddings at the early stage, we integrate sampling negative items from the item side. Specifically, we use the embedding centroid of the user's interacted items as the pseudo user embedding at the early model training stage, as item embeddings are updated more frequently than user embeddings in FedRecs to be more reliable at a very early stage of training. We adopt the element-wise average to calculate the item embedding centroid. Then, we choose hard negative items by calculating the relevance score with the centroid vector.

We first select the top $2R\%$ items as the candidate item pool based on the user-item mixed sampling strategy. Then, we randomly select $N * \beta$ $(0 < \beta < 1)$ negative samples from the candidate item pool to form semi-hard negative samples, and N is the original size of negative samples for each user in the traditional negative sampling methods. We do not select the top negative items from the candidate pool (i.e., hard negatives) to avoid sample false negatives. The sampling mechanism can be formally described as follows:

$$\begin{aligned}
\mathcal{V}_k^u &= \underset{v_i \notin \mathcal{D}_k \wedge \left|\mathcal{V}_k^u\right| = R\% * |\mathcal{V}|}{argmax} R(\mathrm{u}_k^{t-1}, \mathrm{v}_i^{t-1}) \\
\mathcal{V}_k^v &= \underset{v_i \notin \mathcal{D}_k \wedge \left|\mathcal{V}_k^v\right| = R\% * |V|}{argmax} R(\mathrm{v}_{k,cet}^{t-1}, \mathrm{v}_i^{t-1}) \\
\mathcal{V}_k^{neg} &= \underset{\left|\mathcal{V}_k^{neg}\right| = N * \beta}{Random} (\mathcal{V}_k^u \cup \mathcal{V}_k^v)
\end{aligned} \tag{3}$$

where $\mathrm{v}_{k,cet}^{t-1}$ is the average embedding of u_k's interacted items at the beginning of epoch t. $R(\cdot)$ is the relevance measure function. Here, we use Euclidean distance to measure the relevance. $N * \beta$ is the number of negative samples we finally select. The ablation study in the experiment shows that our proposed negative sampling method achieves comparable performance even with $\beta = 0.5$. Therefore, our efficient sampling mechanism significantly reduces the number of required negative samples for each client, thus leading to less storage space for item embedding updates.

3.3.3 ***Storage Space Cost Analysis.*** $|\mathcal{V}|$ is the size of total items, $\left|\mathcal{V}_k^{pos}\right|$ and $\left|\mathcal{V}_k^{neg}\right|$ are the sizes of positive items and selected negative items for user u_k. Assume the number of global epochs in the FedRec is B, and $b\%$ users will be selected for model training at each global epoch. So averagely, a user will be trained $B \times b\%$ times in the whole training process. The cost for storing an item's embedding updates is C. As mentioned previously, for a local recommender, the cost of storing model updates mainly happens in item embeddings. Therefore, we focus on analyzing the space cost of storing item embedding updates here. Before applying our efficient on-device update storing method, each client keeps the whole item embedding table's updates after training. Averagely, the cost of storing update logs for a user u_k is $b\% \times B \times |\mathcal{V}| \times C$. By applying

our proposed Importance-based Update Selection method, the storage space cost can be reduced to $b\% \times B \times \alpha(\left|\mathcal{V}_k^{pos}\right| + \left|\mathcal{V}_k^{neg}\right|) \times C$. Then, by applying our proposed efficient negative sampling method, the storage space cost for each client can be further reduced to $b\% \times B \times \alpha(\left|\mathcal{V}_k^{pos}\right| + \beta\left|\mathcal{V}_k^{neg}\right|) \times C$. Generally, in most FedRecs, the negative items are sampled with a certain ratio of the positive items. Assume the ratio is $1:n$, then, the cost of our FRU's storage for each client is $b\% \times B \times \alpha(1+\beta n)\left|\mathcal{V}_k^{pos}\right| \times C$.

Take the setting of our experiment on Steam-200k as an example, where $B = 200$, $b = 10$, $\alpha = 0.5$, $\beta = 0.5$. The average interacted item size $\left|\mathcal{V}_k^{pos}\right|$ is about 30. The negative sample ratio n for NCF and LightGCN is 4 and 1, respectively. On average, the space cost of storing model updates on each client device is about $900C$ for NCF and $450C$ for LightGCN. The total item size is 5134. As a result, each client only needs to pay an extra 17.5% space cost when using NCF (8.75% when using LightGCN) compared with the FedRec without unlearning ability.

3.4 Unlearning with Updates Revision

When a group of users leave the FedRec service and request to forget their information at a certain time t, FRU will first roll back the federated recommender model to the initial state (i.e. $t = 0$) or the state when the first one of these users joined the federated training process, and then calibrate all historical model updates on the remaining clients. The basic idea of calibration is that we only calibrate the historical updates' direction while keeping the length of updates unchanged since the direction guides the model to fit the training data, which has been polluted by the removed users [30]. Our unlearning method is based on FedEraser [24], which is a unlearning method for classification tasks. But we calibrate updates based on the efficient on-device update storing since the number of clients are greatly larger than classification tasks, meanwhile, we consider how to revise private parameters (i.e. user embeddings) which only exist in Fedrec tasks. In what follows, we will present how to perform unlearning in FRU, which is corresponding to function Unlearning and ClientUnlearning in Algorithm 1. To avoid complex presentation, in this part, we directly use $\mathcal{M}(V)_k^t$ to represent the model updates at time t that are stored with our proposed efficient on-device update storing method. We use V to directly represent the global parameters since the item embedding table dominates the global parameters' part.

3.4.1 ***Search Calibrated Direction.*** At t'th global training round, a group of users are selected to train the FedRec. We denote this group of users as $\mathcal{U}_t$. We first remove the target users $\mathcal{U}'$ who request to forget their information. Then, for each remained user $u_k \in \mathcal{U}_t/\mathcal{U}'$, we run $\lambda * L$ local training epochs based on the unlearned global model $\bar{\mathrm{V}}_{t-1}$ achieved in the last round and the local user embedding u_k^t to get new global parameters' updates $\hat{\mathcal{M}}(V)_k^t$ and new user embedding updates $\hat{\mathcal{M}}(U)_k^t$. Note that the users selected to recover the FedRec are the same as in the original training at round t, except the removed users. Here, L is the original local training epochs, and λ is a *speed-up factor*. We need much fewer local training epochs because we only want to approximate the update direction that can make the model fit the remaining data at this step. It is worth mentioning that the global model V_0 is the initial global recommender model on the central server that has yet received any update from clients, therefore, when $t = 1$, we can directly get the unlearned global model through aggregating updates from the remaining users: $\bar{\mathrm{V}}_1 = \mathrm{V}_0 + agg(\mathcal{M}(V)_k^1|_{u_k \notin \mathcal{U}'})$. The above operations can be seen in Algorithm 1 Line 25, 37.

Algorithm 1 FRU (**F**ederated **R**ecommendation **U**nlearning)

Input: global epoch T; local epoch L; speed-up factor λ; embedding size e; learning rate lr, ...;
Output: global parameter V, local client embedding $\mathrm{u}_k|_{k\in\mathcal{U}}$;

1: Initializing global parameter V_0;
2: **for** each round t = 0, 1, ..., T **do**
3: **if** request unlearning users $\mathcal{U}'$ **then**
4: $\mathrm{V}_t \leftarrow$ Unlearning$(t, \mathcal{U}', \mathrm{V}_0)$;
5: $t \leftarrow t-1$;
6: **else**
7: sampling a fraction of clients $\mathcal{U}_t$;
8: **for** $u_k \in \mathcal{U}_t$ **do**
9: $\mathcal{M}(V)_k^{t+1} \leftarrow$ ClientUpdate(u_k, V_t, L);
10: **end for**
11: $\mathrm{V}_{t+1} \leftarrow \mathrm{V}_t + agg(\mathcal{M}(V)_k^{t+1})$;
12: **end if**
13: **end for**
14: **function** ClientUpdate(u_k, V_t, L)
15: downloading V_t from the server;
16: $\mathcal{V}_k^{neg} \leftarrow$ sampling negative items using E.q. 3;
17: $\mathcal{M}(U)_k^{t+1}, \mathcal{M}(V)_k^{t+1} \leftarrow$ training L epochs on $\mathcal{V}_k^{neg} \cup \mathcal{V}_k^{pos}$;
18: $\mathrm{u}_k^{t+1} = \mathrm{u}_k^t + \mathcal{M}(U)_k^{t+1}$;
19: $\mathcal{M}(V)_k^{t+1} \leftarrow$ selecting updates based on $\|\mathcal{M}(V)_k^{t+1}\|$;
20: storing $\mathcal{M}(V)_k^{t+1}$ and user embedding u_k^{t+1} locally;
21: **return** $\mathcal{M}(V)_k^{t+1}$
22: **end function**
23: **function** Unlearning$(\mathrm{T}, \mathcal{U}', \mathrm{V}_0)$
24: enquiring $\mathcal{M}(V)_k^1$ from clients;
25: $\bar{\mathrm{V}}_1 \leftarrow \mathrm{V}_0 + agg(\mathcal{M}(V)_k^1|_{u_k \notin \mathcal{U}'})$;
26: **for** t=2, ..., T **do**
27: **for** $u_k \in \mathcal{U}_t/\mathcal{U}'$ **do**
28: $\hat{\mathcal{M}}(V)_k^{t+1} \leftarrow$ ClientUnlearning$(u_k, \bar{\mathrm{V}}_{t-1}, \lambda, L)$;
29: **end for**
30: enquiring $\mathcal{M}(V)_k^t$ from clients;
31: $\bar{\mathcal{M}}(V)^t \leftarrow$ calculate with E.q. 4;
32: $\bar{\mathrm{V}}_t \leftarrow$ global parameter update with E.q. 5;
33: **end for**
34: **return** $\bar{\mathrm{V}}_T$
35: **end function**
36: **function** ClientUnlearning$(u_k, \bar{\mathrm{V}}_t, \lambda, L)$
37: $\hat{\mathcal{M}}(V)_k^{t+1} \leftarrow$ ClientUpdate$(u_k, \bar{\mathrm{V}}_t, \lambda * L)$;
38: **return** $\hat{\mathcal{M}}(V)_k^{t+1}$
39: **end function**

3.4.2 ***Aggregate and Modify Updates.*** After the first step, the client computed new global parameter updates $\hat{\mathcal{M}}(V)_k^t$ and user embedding updates $\hat{\mathcal{M}}(U)_k^t$. For the global parameter updates, clients at first upload their new computed updates $\hat{\mathcal{M}}(V)_k^t$ and the stored

updates $\mathcal{M}(V)_k^t$ to the central server. Then, the central server aggregates these updates and combines the new updates' direction with the original updates' length to construct the new calibrated updates $\bar{\mathcal{M}}(V)_k^t$, since the original updates' direction guides model to fit old training data.

$$\bar{\mathcal{M}}(V)^t = \left\| agg(\mathcal{M}(V)_k^t)|_{u_k \notin \mathcal{U}'} \right\| \frac{agg(\hat{\mathcal{M}}(V)_k^t)|_{u_k \notin \mathcal{U}'}}{\left\| agg(\hat{\mathcal{M}}(V)_k^t)|_{u_k \notin \mathcal{U}'} \right\|} \tag{4}$$

$agg(\cdot)$ is aggregation strategy.

Then, the global parameters can be recovered at the central server based on the above calibrated updates, as follows:

$$\bar{\nabla}_t = \bar{\nabla}_{t-1} + \bar{\mathcal{M}}(V)^t \tag{5}$$

For user embedding updates, we directly use $\hat{\mathcal{M}}(U)_k^t$ to update user embedding: $\bar{u}_k^t = \bar{u}_k^{t-1} + \hat{\mathcal{M}}(U)_k^t$, because the updating frequency of user embeddings is much lower than item embeddings, and the previous user embedding updates may not be reliable.

FRU repeats the above process at each global round to achieve the unlearned model.

3.4.3 ***Time Complexity Analysis.*** One of the key advantages of FRU is that it can accelerate the reconstruction of the FedRec, compared with retraining from scratch. The speed-up ratio is mainly related to the speed-up factor λ, which allows clients to perform fewer rounds of local training. Note that the local training time costs dominate the time complexity of the whole federated unlearning process. In our experiments, we set $\lambda = 0.1$. Under this setting, FRU can achieve up to 10x speedup compared with retraining from scratch. Empirically, FRU is 7x faster than retraining from scratch.

4 EXPERIMENTS

4.1 Experimental Settings

Datasets. We adopt two commonly used datasets for the federated recommendation: MovieLens-100k[1] [14] and Steam-200k [9]. MovieLens-100k contains 100,000 interactions, 943 users, and 1,682 movies, while Steam-200k includes 3,753 users, 5,134 steam games, and 114,713 user-game interactions. Following [15, 33, 46], all the observed interactions are converted to $r_{ij} = 1$. 80% data and 20% data are treated as train and test set.

Baselines. As there are no federated unlearning works for the on-device recommendation, we construct the following two baselines: **Retrain** and **FedRemove**. Retrain means deleting the target users that we need to forget and then retraining the federated recommender model from scratch with the remaining users. We take 'Retrain' as the reference method that can lead to ideal unlearning results (e.g., ground-truth results). FedRemove is the method that simply removes the target users' global parameter updates and directly aggregates the remaining clients' updates during each global round when recovering the FedRec. In other words, compared with FRU, FedRemove does not include calibrating operations. Therefore, its recovery speed is very fast.

Evaluation Metrics. We employ the widely used Hit Ratio at rank 10 (HR@10) and Normalized Discounted Cumulative Gain at rank 10 (NDCG@10) to measure the recommendation performance.

[1]https://grouplens.org/datasets/movielens/100k/

Evaluating the effectiveness of the federated unlearning is not an easy task, as analyzed in Section 1. In this paper, we evaluate the unlearning effectiveness in the following attack scenario. We assume a group of users are compromised. The malicious users attempt to perturb the FedRec's training process by uploading poisoning gradients to the server. Specifically, the malicious user/client uploads flipped gradients to the server. Simply flipping the gradients cannot severely destroy the FedRec, as many normal users may counteract the flipped gradients with correct gradients. Therefore, we further add slight noise in the flipped gradients and use a randomly generated factor γ to control the flipped gradients' magnitude.

$$\widetilde{\mathcal{M}}(V)_k^t = -\gamma \mathcal{M}(V)_k^t + \mu \tag{6}$$

where $\widetilde{\mathcal{M}}(V)_k^t$ is poisoning updates generated by malicious user u_k. γ is a random factor sampled from a uniform distribution. μ is a noise sampled from a normal distribution where the mean and standard deviation are calculated based on $\mathcal{M}(V)_k^t$.

To erase the influence of these malicious clients/users, we first delete these users from the FedRec and then apply our proposed FRU and the two baselines to recover the FedRec. In this scenario, we only need to evaluate whether the malicious clients' poisoning influence can be erased.

Parameter Settings For both NCF, the dimension of user and item embedding is 64. We adopt 4 neural layers with dimensions 128, 256, 128, and 64 to process the concatenated user and item embeddings. For LightGCN, the dimension of user and item embedding is 64. We use 1 layer for the graph propagation. The original negative sampling ratio is $1:4$ and $1:1$ for NCF and LightGCN, following previous studies [1, 16, 46]. All other hyperparameters for these two models are the same. Specifically, the learning rate, batch size, local epoch, and the maximum global epoch are set to 0.001, 64, 20, and 200, respectively. α, β, and λ are 0.5, 0.5, 0.1.

4.2 Main Results and Analysis

In this part, we present and discuss the main experimental results on erasing the poisoning influence from malicious users. We evaluate the performance of our proposed FRU from two aspects: (1) whether the FedRec is recovered; (2) the efficiency of unlearning (i.e., recovering the FedRec).

4.2.1 ***Removing Malicious Users' Influence.*** Table 1 shows the comparison of recommendation accuracy. "Attacked" means that the FedRecs are attacked by malicious users. "Retrain" removes malicious users and then retrains FedRecs from scratch. Table 1 shows that FRU can recover the attacked FedRecs and achieve even better performance than Retrain in all cases, demonstrating FRU's effective and generic unlearning ability for different FedRec models. FedRemove cannot work well with NCF or LightGCN, indicating the importance of calibrating updates. In addition, with importance-based update selection, FRU can perform even better. This observation indicates that only important updates should be calibrated, and over-calibration can negatively affect model performance.

4.2.2 ***Efficiency of Unlearning.*** Compared with retraining from scratch, one of the most important advantages of FRU is that it can speed up FedRec's recovery. Fig. 1 shows the time costs of different unlearning methods on different datasets. Under our settings of

Table 1: Comparison of recovering attacked federated recommenders. "IUS" is short for importance-based update selection. "Attacked" shows the federated recommender's performance destroyed by the attack method. "Retrain" train Fedrec on the data without malicious users, whose performance can be referred as the performance of FedRec that has not been attacked.

		MovieLens-100k				Steam-200k			
		malicious 10%		malicious 20%		malicious 10%		malicious 20%	
Base Model	Method	hit	ndcg	hit	ndcg	hit	ndcg	hit	ndcg
Fed-NCF	Attacked	55.34	30.10	49.02	25.15	81.05	47.58	76.28	44.34
	FedRemove	51.06	24.12	45.28	28.22	76.77	45.33	49.30	27.72
	Retrain	57.24	31.33	58.33	31.84	85.64	49.72	87.05	49.73
	FRU w/o IUS	57.96	30.57	58.61	**32.77**	88.39	50.76	91.19	51.87
	FRU	**58.43**	**31.26**	**60.28**	32.38	**88.42**	**50.86**	**91.76**	**52.33**
Fed-LightGCN	Attacked	59.91	29.92	45.32	24.19	88.56	51.95	86.52	50.84
	FedRemove	57.48	27.17	56.67	31.41	87.22	50.68	90.14	51.75
	Retrain	61.99	32.31	62.22	32.15	92.14	53.85	93.64	54.31
	FRU w/o IUS	62.47	33.25	62.50	**33.29**	92.07	53.12	93.26	53.85
	FRU	**63.90**	**34.61**	**63.06**	31.53	**93.57**	**55.07**	**93.79**	**54.28**

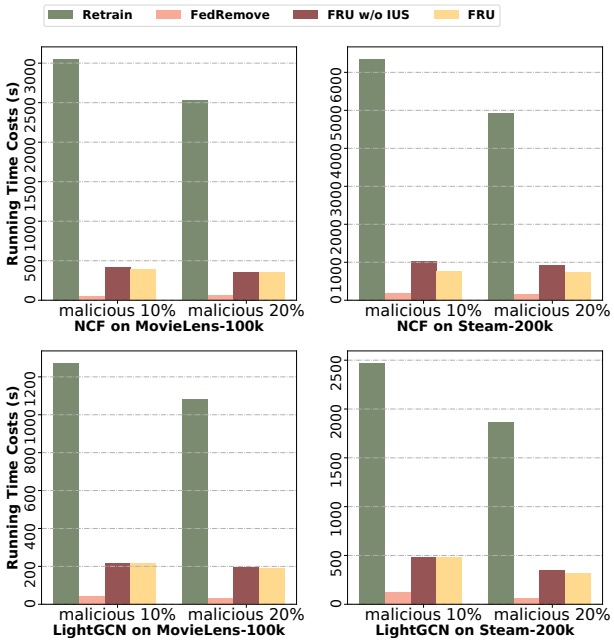

Figure 1: Time comparison of recovering FedRecs on MovieLens-100k and Steam-200k.

FRU (i.e. $\lambda = 0.1$), FRU is 7x faster than Retrain for both NCF and LightGCN models. FRU is also a little bit faster than FRU w/o IUS because each client in FRU has fewer item embedding updates to calibrate. As expected, FedRemove is the most efficient unlearning method as it does not need to calibrate updates. However, its unlearning effectiveness is the worst, as shown in Table 1.

4.3 Ablation Study and Hyperparameter Analysis

In this part, we investigate α, β, and λ's influence, respectively. Experiments are conducted on MovieLens-100k and Steam-200k with NCF and LightGCN, but we only present the results on MovieLens-100k due to the space limitation, and similar results and trends are observed on Steam-200k. When investigating one component, the hyperparameters in other components keep unchanged.

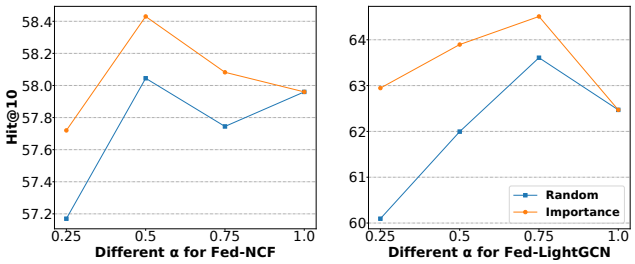

Figure 2: Different values of α for NCF and LightGCN on MovieLens-100k when recovering FedRecs from 10% malicious users' attack. "Random" (blue line) means randomly selecting updates to store, while "Importance" (orange line) selects updates based on their importance.

4.3.1 ***Effect of Importance-based Update Selection.*** We explore different values of α for importance-based update selection and compare the results with randomly selecting updates. The results are shown in Fig. 2. When $\alpha = 1.0$, FRU degrades to FRU w/o IUS since all updates will be stored. In Fig. 2, our importance-based selection method outperforms the random selection strategy with all different α values. In addition, the trends also show that FRU's performance first increases to a peak point and then decreases with continually increasing α value (i.e., keeping more item embedding updates). This phenomenon indicates that storing and calibrating too many unimportant updates will hurt FRU's performance.

4.3.2 ***Effect of User-item Mixed Semi-hard Negative Sampling.*** To analyze our proposed negative sampling method's effectiveness, we train FedRecs without attackers and explore three different values of β. $\beta = 1.0$ represents keeping the original sampling ratio in the traditional sampling method. $\beta = 0.5$ means our negative sampling rate is 50% of the original one. In Fig. 3, our user-item mixed semi-hard negative sampling method outperforms all baselines under the same β values. Another important observation is that our proposed sampling method with $\beta = 0.5$ achieves

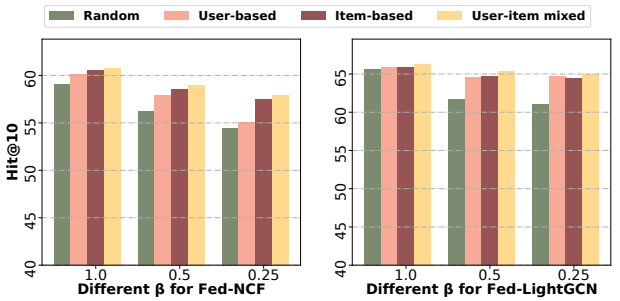

Figure 3: Different values of β for NCF and LightGCN on MovieLens-100k without attacks. "Random" means randomly selecting negative samples, "User-based" and "Item-based" select negatives based on the relevance with the user embedding and the centroid of interacted items' embeddings, respectively.

almost the same performance as the random sampling method with $\beta = 1.0$. As a result, using our proposed negative sampling method, we can reduce the negative sample size and further reduce the number of item embedding updates at each iteration without comprising the recommendation performance.

Table 2: The comparison of different λ values to recover NCF and LightGCN destroyed by 10% malicious users on MovieLens-100k.

	Fed-NCF		Fed-LightGCN	
λ	Hit	Time Costs (s)	Hit	Time Costs (s)
0.1	**58.43**	**396.9**	63.90	**217.3**
0.25	57.04	894.6	61.28	394.1
0.5	56.06	1675.45	**64.24**	730.3
0.75	53.44	2270.61	62.27	1029.2

4.3.3 ***Impact of Speedup Factor.*** The speedup of recovering FedRecs mainly depends on the speedup factor λ. In this part, we explore different values of λ to show the unlearning recommendation results and the recovery time costs. From Table 2 we can observe that when we increase λ, NCF's performance will drop. For LightGCN, the relationship between the speedup factor and its performance is a little complex, but overall, it obtains lower performance with a larger λ value. For all FedRecs, the recovery time costs consistently increase with increasing λ values.

5 RELATED WORK

Federated Recommendation. Ammad et al. [1] propose the first general framework of FL with implicit feedback. Muhammad et al. [29] ameliorate the user sampling and aggregation strategy to accelerate the convergency process. Besides utilizing implicit feedback, Lin et al [23] and Liang et al. [22] attempt to exploit explicit feedback with FL. Kharitonov et al. [20] adopt online learning to learn online feedback. Wu et al. [40] incorporate GNN to improve federated recommendation performance further. Aside from improving the system's performance, many works also focus on attacking federated recommendations, such as [33, 38, 43, 46]

Machine Unlearning. The work of machine unlearning can be classified into two categories: *Exact Unlearning* and *Approximate Unlearning* [31]. For exact unlearning, methods are designed to theoretically guarantee that models can totally remove the influence from certain data. These methods usually require complicated mathematical calculations. Therefore, they can only be used for specific simple machine learning models [2, 4, 8, 10, 19, 34]. More recently, to achieve unlearning in more complex models (e.g. deep learning), Bourtoule et al. [3] propose SISA. This method accelerates retraining through splitting the dataset into multiple partitions and training sub-models on these data shards. Based on SISA, some research takes steps into applying unlearning in deep learning models [6]. However, all these methods have to utilize all data information, which is forbidden in FedRecs.

Unlike exact unlearning, approximate unlearning methods relax the requirements of certifiably erasing data contributions. Otherwise, it only provides a statistical assurance that the unlearned model forgets the targeted data to a large extent. Until now, all federated learning-based unlearning approaches belong to approximate unlearning. For example, [25] and [24] delete samples' influence via gradients. Wu et al. [41] introduce knowledge distillation to unlearn federated models. Liu et al. [26] adopt the Newton methods to speed up retraining. However, most of these methods are applied to classification tasks. Compared with FedRecs, these tasks have much fewer clients (users) and do not have private model parameters. Therefore, these methods cannot be directly used in FedRecs.

6 CONCLUSION

This paper proposes the first federated unlearning framework for on-device recommendation, called FRU. Specifically, FRU stores all users' historical updates and reconstructs FedRecs by calibrating these updates, which is similar to the log-based rollback mechanism in database management systems. We propose an importance-based update selection strategy and a novel negative sampling method to efficiently store historical updates on resource-constrained devices. To evaluate FRU's unlearning ability, we propose an attack method that employs a group of compromised clients to perturb FedRecs' training process. Then, we apply FRU to erase these malicious users' influence. FRU is model-agnostic and can be incorporated in most FedRecs. We conduct experiments with two popular recommenders on two real-world recommendation datasets. The results show that FRU can eliminate specific users' influence and efficiently recover the FedRecs with 7x speedup. Finally, ablation studies are investigated to show the contributions of FRU's different components.

ACKNOWLEDGMENTS

This work is supported by Australian Research Council Future Fellowship (Grant No. FT210100624), Discovery Project (Grant No. DP190101985). This work contributes to industry knowledge engineering in OpenTrek which is a technical brand of industry intelligence in Alibaba Cloud.

REFERENCES

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

[2] Thomas Baumhauer, Pascal Schöttle, and Matthias Zeppelzauer. 2020. Machine unlearning: Linear filtration for logit-based classifiers. *arXiv preprint arXiv:2002.02730* (2020).

[3] Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 141–159.

[4] Jonathan Brophy and Daniel Lowd. 2021. Machine unlearning for random forests. In *International Conference on Machine Learning*. PMLR, 1092–1104.

[5] Chong Chen, Fei Sun, Min Zhang, and Bolin Ding. 2022. Recommendation Unlearning. *arXiv preprint arXiv:2201.06820* (2022).

[6] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2021. Graph Unlearning. *arXiv preprint arXiv:2103.14991* (2021).

[7] Tong Chen, Hongzhi Yin, Hongxu Chen, Lin Wu, Hao Wang, Xiaofang Zhou, and Xue Li. 2018. Tada: trend alignment with dual-attention multi-task recurrent neural networks for sales prediction. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 49–58.

[8] Yuantao Chen, Jie Xiong, Weihong Xu, and Jingwen Zuo. 2019. A novel online incremental and decremental learning algorithm based on variable support vector machine. *Cluster Computing* 22, 3 (2019), 7435–7445.

[9] Germán Cheuque, José Guzmán, and Denis Parra. 2019. Recommender systems for online video game platforms: The case of steam. In *Companion Proceedings of The 2019 World Wide Web Conference*. 763–771.

[10] Antonio Ginart, Melody Guan, Gregory Valiant, and James Y Zou. 2019. Making ai forget you: Data deletion in machine learning. *Advances in Neural Information Processing Systems* 32 (2019).

[11] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9304–9312.

[12] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. 2019. Certified data removal from machine learning models. *arXiv preprint arXiv:1911.03030* (2019).

[13] Elizabeth Liz Harding, Jarno J Vanto, Reece Clark, L Hannah Ji, and Sara C Ainsworth. 2019. Understanding the scope and impact of the California Consumer Privacy Act of 2018. *Journal of Data Protection & Privacy* 2, 3 (2019), 234–253.

[14] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[15] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 355–364.

[16] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. 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*. 639–648.

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

[18] Mubashir Imran, Hongzhi Yin, Tong Chen, Nguyen Quoc Viet Hung, Alexander Zhou, and Kai Zheng. 2022. ReFRS: Resource-efficient Federated Recommender System for Dynamic and Diversified User Preferences. *ACM Transactions on Information Systems (TOIS)* (2022).

[19] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. 2021. Approximate data deletion from machine learning models. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2008–2016.

[20] Eugene Kharitonov. 2019. Federated online learning to rank with evolution strategies. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 249–257.

[21] Yuyuan Li, Xiaolin Zheng, Chaochao Chen, and Junlin Liu. 2022. Making Recommender Systems Forget: Learning and Unlearning for Erasable Recommendation. *arXiv preprint arXiv:2203.11491* (2022).

[22] Feng Liang, Weike Pan, and Zhong Ming. 2021. Fedrec++: Lossless federated recommendation with explicit feedback. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4224–4231.

[23] Guanyu Lin, Feng Liang, Weike Pan, and Zhong Ming. 2020. Fedrec: Federated recommendation with explicit feedback. *IEEE Intelligent Systems* 36, 5 (2020), 21–30.

[24] Gaoyang Liu, Xiaoqiang Ma, Yang Yang, Chen Wang, and Jiangchuan Liu. 2021. FedEraser: Enabling Efficient Client-Level Data Removal from Federated Learning Models. In *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*. IEEE, 1–10.

[25] Yang Liu, Zhuo Ma, Ximeng Liu, and Jianfeng Ma. 2020. Learn to forget: User-level memorization elimination in federated learning. *arXiv preprint arXiv:2003.10933* 1 (2020).

[26] Yi Liu, Lei Xu, Xingliang Yuan, Cong Wang, and Bo Li. 2022. The Right to be Forgotten in Federated Learning: An Efficient Realization with Rapid Retraining. *arXiv preprint arXiv:2203.07320* (2022).

[27] Jing Long, Tong Chen, Nguyen Quoc Viet Hung, and Hongzhi Yin. 2022. Decentralized Collaborative Learning Framework for Next POI Recommendation. *arXiv preprint arXiv:2204.06516* (2022).

[28] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[29] Khalil Muhammad, Qinqin Wang, Diarmuid O'Reilly-Morgan, Elias Tragos, Barry Smyth, Neil Hurley, James Geraci, and Aonghus Lawlor. 2020. Fedfast: Going beyond average for faster training of federated recommender systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1234–1242.

[30] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 739–753.

[31] Thanh Tam Nguyen, Thanh Trung Huynh, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. 2022. A Survey of Machine Unlearning. *arXiv preprint arXiv:2209.02299* (2022).

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

[33] Dazhong Rong, Shuai Ye, Ruoyan Zhao, Hon Ning Yuen, Jianhai Chen, and Qinming He. 2022. FedRecAttack: Model Poisoning Attack to Federated Recommendation. *arXiv preprint arXiv:2204.01499* (2022).

[34] Sebastian Schelter, Stefan Grafberger, and Ted Dunning. 2021. Hedgecut: Maintaining randomised trees for low-latency machine unlearning. In *Proceedings of the 2021 International Conference on Management of Data*. 1545–1557.

[35] Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan Kankanhalli. 2021. Fast Yet Effective Machine Unlearning. *arXiv preprint arXiv:2111.08947* (2021).

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

[37] Qinyong Wang, Hongzhi Yin, Tong Chen, Zi Huang, Hao Wang, Yanchang Zhao, and Nguyen Quoc Viet Hung. 2020. Next point-of-interest recommendation on resource-constrained mobile devices. In *Proceedings of the Web conference 2020*. 906–916.

[38] Qinyong Wang, Hongzhi Yin, Tong Chen, Junliang Yu, Alexander Zhou, and Xiangliang Zhang. 2022. Fast-adapting and privacy-preserving federated recommender system. *The VLDB Journal* 31, 5 (2022), 877–896.

[39] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural memory streaming recommender networks with adversarial training. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2467–2475.

[40] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. 2021. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *arXiv preprint arXiv:2102.04925* (2021).

[41] Chen Wu, Sencun Zhu, and Prasenjit Mitra. 2022. Federated Unlearning with Knowledge Distillation. *arXiv preprint arXiv:2201.09441* (2022).

[42] Ga Wu, Masoud Hashemi, and Christopher Srinivasa. 2022. PUMA: Performance Unchanged Model Augmentation for Training Data Removal. *arXiv preprint arXiv:2203.00846* (2022).

[43] Jingwei Yi, Fangzhao Wu, Bin Zhu, Yang Yu, Chao Zhang, Guangzhong Sun, and Xing Xie. 2022. UA-FedRec: Untargeted Attack on Federated News Recommendation. *arXiv preprint arXiv:2202.06701* (2022).

[44] Hongzhi Yin, Hongxu Chen, Xiaoshuai Sun, Hao Wang, Yang Wang, and Quoc Viet Hung Nguyen. 2017. SPTF: a scalable probabilistic tensor factorization model for semantic-aware behavior prediction. In *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 585–594.

[45] Shijie Zhang, Hongzhi Yin, Tong Chen, Zi Huang, Lizhen Cui, and Xiangliang Zhang. 2021. Graph embedding for recommendation against attribute inference attacks. In *Proceedings of the Web Conference 2021*. 3002–3014.

[46] Shijie Zhang, Hongzhi Yin, Tong Chen, Zi Huang, Quoc Viet Hung Nguyen, and Lizhen Cui. 2022. PipAttack: Poisoning Federated Recommender Systems for Manipulating Item Promotion. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 1415–1423.

[47] Shijie Zhang, Hongzhi Yin, Tong Chen, Quoc Viet Nguyen Hung, Zi Huang, and Lizhen Cui. 2020. Gcn-based user representation learning for unifying robust recommendation and fraudster detection. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 689–698.