Meta-Learn to Unlearn: Enhanced Exact Machine Unlearning in Recommendation Systems with Meta-Learning

Abdulla Alshabanah
aalshaba@usc.edu
University of Southern California
Los Angeles, California, USA

Keshav Balasubramanian
keshavba@usc.edu
University of Southern California
Los Angeles, California, USA

Murali Annavaram
annavara@usc.edu
University of Southern California
Los Angeles, California, USA

Abstract

Recommendation systems are used widely to recommend items such as movies, products, or news to users. The performance of a recommendation model depends on the quality of the embeddings that are associated with users and items, which are generally learned by tracking user behavior, such as their click history. Recent legislative requirements allow users to withdraw their consent to learning from some of their behaviors, even if they have provided such a consent initially. Once a user withdraws their consent, the models are supposed to unlearn the user behavior. This requirement has led to the emergence of machine unlearning, a research area that proposes a class of privacy policy-compliant techniques aimed at maintaining good model utility after deleting user information. Machine unlearning techniques are generally divided into two categories: exact unlearning, which may be accomplished by retraining the model from scratch after removing a data point from the training data; and approximate unlearning, which approximates the model parameters that would result from removing a specific user data, without needing a complete retraining of the model to minimize computational costs.

In this work, we propose an enhanced exact machine unlearning (EEMU) strategy that leverages meta-learning to reduce the loss of recommendation performance while ensuring efficient and exact unlearnability. We demonstrate our results using four public datasets and show a significant improvement in recommendation performance over state-of-the-art baselines while preserving the privacy guarantees of exact unlearning. The source code of EEMU is available at: https://github.com/alshabae/EEMU.

Keywords

Recommendation Systems, Exact Machine Unlearning, User Privacy

1 Introduction

In today's data-driven world, machine learning (ML) applications rely heavily on sensitive user information. Recommendation models, for instance, track user behavior such as their click history, to learn user and item embeddings which are part of ML models that aim to improve recommendation performance. While behavior tracking is usually done with explicit user consent, their consent may be withdrawn at any time. In fact, regulatory bodies worldwide have enacted policies that require supporting such privacy preservation, and other data retention restrictions in ML models [4, 28, 31]. In particular, the "right to be forgotten" law requires ML models to selectively forget or unlearn user information upon request, which is known as the *Machine Unlearning* problem [3].

It is challenging to design an algorithm that allows an ML model to selectively forget specific data subsets it was trained on. The difficulty lies in the non-linear interactions between different training samples that is embedded into the model parameters. Unlearning approaches can be categorized into exact unlearning and approximate unlearning. Exact unlearning typically involves retraining the entire model from scratch after removing the desired data points, guaranteeing the removal of their influence [3, 5]. However, it incurs significant computational overhead for large-scale models. On the other hand, approximate unlearning methods aim to provide guarantees within a margin of error, offering greater practicality and lower computational overhead [11, 16, 17, 42]. However, the guarantees are provided under specific data distribution assumptions, and such margin of errors can be difficult to prove for applications that require strict adherence to the right-to-be-forgotten provisions in the European Union's General Data Protection Regulation (GDPR) [28], United States' California Consumer Privacy Act (CCPA) [4] and Canada's Personal Information Protection and Electronic Documents Act (PIPEDA) [31].

Our research specifically focuses on the problem of exact unlearning in recommendation systems, which involve training models based on user and item interaction histories. We focus on recommendation systems constructed using Deep Neural Networks (DNNs) [9, 29, 45, 46]. Although unlearning has been explored in various ML domains, its application to recommendation systems poses several challenges. Each user and item embedding is learned from a wide range of user-item interactions that are recorded in the recommendation model training data. Effective unlearning techniques require selectively removing some of the user-item interactions, as dictated by the user, while maintaining model performance and usability.

The primary challenge in achieving exact unlearning in any ML domain is the computational overhead associated with retraining the entire model. To mitigate this issue, the SISA (Sharded Isolated Sliced and Aggregated) training method was introduced [3]. SISA implements a sharded training scheme that divides the training dataset uniformly at random into disjoint shards and trains replica models using these shards. When a deletion request occurs, only the shard containing the deleted data point needs to be retrained, significantly reducing the overhead associated with retraining. During inference, the final prediction is computed via a static, permutation invariant function.

Proceedings on Privacy Enhancing Technologies 2025(4), 696–711

https://doi.org/10.56553/popets-2025-0152

In the context of recommendation models, user-item interactions are the key inputs to recommendation models that capture actions like clicks that indicate user interest. These interactions serve as the foundation for learning user and item representations. While user-user and item-item signals are not explicitly defined, models implicitly learn these patterns through embeddings, capturing relationships like similar users or frequently paired items. These relationships help improve recommendation performance by identifying complex patterns in user behavior and item relevance. Applying the SISA strategy to such recommendation models presents a challenge due to the potential loss of collaborative signal caused by random sharding of user-item interactions [19]. When data is split across multiple shards, each user's interactions with different items may be dispersed across different shards. Hence, each shard has a narrow view of the user's interactions from which it has to learn. Sharding of user-item interactions leads to incomplete user profiles and fewer interactions per shard, reducing the model's ability to learn user and item representations and ultimately hurting performance. To address this limitation, RecEraser [6] was proposed as a variant of SISA tailored to DNN-based recommendation systems. RecEraser shards interactions based on user and item similarity and uses a learnable attention-based aggregator during inference. UltraRE [25] improves over RecEraser by proposing an optimal balanced sharding algorithm and a more efficient learnable aggregator.

While RecEraser and UltraRE adaptation of the SISA method to recommendation systems has been successful, we have identified the following drawbacks that affect their applicability to exact machine unlearning:

(1) Partitioning the data based on the similarity of user and item latent representations necessitates the availability of pretrained user and item embeddings. (RecEraser and UltraRE use the model introduced in [21] to pretrain user and item embeddings). Thus, the pretrained model determines the similarity metric that is used in the sharding process. When exact unlearning is necessary, the pre-trained model must also be trained again from scratch in the event of a data point deletion, which imposes significant burden, as discussed in prior work [7]. If the pretrained model isn't retrained and the data isn't repartitioned, the unlearning process cannot be exact as the partitions themselves encode information about data points that are deleted. Hence, the pretrained model must be first trained without the sensitive data point, and then use that pretrained model to identify the similarity of users and items to reshard the data. Having to retrain a pretraining model on the entire dataset and to repartition the data again can detract from the benefit of sharding in the first place.
(2) Using a learnable aggregator also presents a challenge. While affording greater expressiveness and dynamic pooling across shards during ranking, the learnable aggregator also needs to be retrained on the entire dataset in the event of a data point deletion. This overhead diminishes the advantages of sharding in the first place. To reduce the retraining overhead, UltraRE does not retrain the learnable aggregator, while RecEraser proposes to retrain the learnable aggregator for a small number of epochs using the entire dataset. While their findings demonstrate promising empirical performance on their test data, training the aggregator even for few epochs on the entire dataset is expensive as reported in [6, 25]. The alternative strategy of keeping the learnable aggregator unchanged during unlearning would violate the exact unlearning guarantees [6, 7].

To address these challenges, we introduce a novel approach called *EEMU (**E**nhanced **E**xact **M**achine **U**nlearning)*. Unlike the approaches in previous work, EEMU adopts random balanced data partitioning and a static mean pooling aggregator, overcoming the identified drawbacks. To mitigate potential model degradation resulting from the loss of collaborative signal from random partitioning and static aggregation, EEMU leverages meta-learning to enhance the training process. To our knowledge, this is the first paper to explore meta-learning in the context of exact machine unlearning.

The summary of the contributions of our work is as follows:

(1) We propose EEMU showcasing how meta-learning can be effectively integrated with the SISA method to yield significantly improved, exactly unlearnable, recommendation models. Unlike previous approaches that either use pretrained embeddings or employ learnable aggregators, EEMU samples several training data points from across shards for collaborative signal recovery.
(2) Through extensive experimentation on four public datasets, we demonstrate that EEMU outperforms several baselines in terms of recommendation performance, while guaranteeing exact unlearning.
(3) We leverage users' task similarity in the context of meta-learning to further enhance EEMU inference efficiency and enable batch inferencing while maintaining good model utility.

Outline. The remaining parts of the paper are structured as follows. In Section 2, we introduce the preliminaries. Then, in Section 3, we discuss the related work, and in Section 4, we present our proposed model and its work flow. Section 5 presents our findings, including ablation and sensitivity analysis results. We then, in Section 6, showcase an enhancement to enable batch inferencing. We conclude with a discussion of possible future directions in Section 7. In Table 1, we present the notations used in the paper.

Symbol	Description
$\mathcal{U}$	Set of users
$\mathcal{I}$	Set of items
$\mathbf{x_u}$	User representation
$\mathbf{x_i}$	Item representation
S	Number of shards
$\mathcal{M}_s^{global}$	Global model of shard s
p_i	Sampling probability of item i for adaptation set
$\mathcal{M}_{s,u}^{special}$	Specialized model for user u in shard s
K	Number of clusters
$\mathcal{M}_{s,c}^{special}$	Specialized model for cluster c in shard s

Table 1: Summary of symbols and notations.

2 Preliminaries

2.1 Exact Recommendation Unlearning

The objective of recommendation systems is to generate a list of items that align with the user's preferences. Recommendation systems are trained using datasets consisting of user-item interactions that are gathered from users' interaction histories. Concretely, given users $u \in \mathcal{U}$, items $i \in \mathcal{I}$, we define the dataset $\mathbb{D}(\mathbb{V}, \mathbb{E})$ with $\mathbb{V} = \mathcal{U} \cup \mathcal{I}$ and $\mathbb{E} = \{(u, i) \mid u \in \mathcal{U},\ i \in \mathcal{I}\}$. In the case of exact recommendation unlearning, if a user u wishes to retract a particular interaction (u, i) from their history, then the recommendation system must be retrained from scratch with the retracted interaction removed from the dataset [43]. Hence, as introduced in [3, 6, 7, 25], the objectives of exact recommendation unlearning are as follows:

(1) The recommendation system should **completely remove the impact** of the retracted record from the trained model, thus, the record should be removed from both the training and preprocessing phases [7].
(2) It must maintain high training **efficiency** and minimize the need to be retrained using the whole dataset [3].
(3) It should have comparable **quality of recommendations** to the ones provided by the original model [6, 25].

2.2 Meta-Learning

Meta-learning, also known as learning-to-learn, aims to train models that quickly adapt to specific tasks with limited data by leveraging knowledge from previously learned tasks [30]. One effective approach to achieve this task is the Model-Agnostic Meta-Learning (MAML) algorithm [13], which is designed to optimize a model's parameters for rapid adaptation across tasks. The algorithm involves a training phase where the model is exposed to multiple tasks and learns a shared initialization. For each task, MAML performs inner-loop updates on task-specific data (called the support set), followed by an outer-loop update on a generally smaller subset of the task-specific data (called the query set) to optimize the initialization across tasks. During the inner loop, the task-specific parameters θ_i are updated using the support set for that task as follows:

$$\theta_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(\theta) \tag{1}$$

where $\mathcal{L}_i(\theta)$ is the task-specific loss.

In the outer loop, the shared initialization θ is updated by aggregating gradients obtained from the query set for each task:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(\theta_i) \tag{2}$$

Inference in Meta-learning: The above approach generates a global model that has learned from the data of multiple tasks. During inference, before providing any task specific prediction, few-shot examples are drawn from the task-specific data. These examples are then used to fine-tune the global model to create a transient model. The fine-tuned transient model is then queried for the prediction. After the prediction is complete the transient model may be optionally deleted or preserved for future use. This approach enables the model to generalize effectively across tasks by performing few-shot specialization on a small number of task-specific samples, without requiring extensive fine-tuning during inference.

Adapting Meta-learning: In our work, we adapt meta learning by treating each task as the objective of learning to recommend items to a specific user. Thus, if there are N users in the system then the meta learning framework has to process N tasks, each task u is focused on recommending items to a specific user u. In the context of a sharded training scheme, the task-specific data (namely per-user interaction history) is split across multiple shards. We train a global model on a per-shard basis using equations 1 and 2. For instance, each user interactions within a shard are first split into support and query sets to enable the inner and outer loop updates. Thus, the per-shard global model training does not deviate from the meta-learning approach explained earlier.

As we discuss in detail in Section 4, however, we designed a unique inference framework that is built on the meta-learning approach explained earlier but is uniquely suited for a sharded training scheme.

2.3 Recommendation Model Architecture

In this work we demonstrate our idea using the popular Two-Tower Neural Network (TTNN) [1, 29, 45] as the fundamental recommendation model. TTNNs consist of two Deep Neural Networks: the user tower and the item tower. The user tower processes both dense and sparse user features to generate a dense user representation $\mathbf{x_u}$, while the item tower performs the same function for items, producing a dense item representation $\mathbf{y_i}$. What constitutes a dense or a sparse feature is generally defined by the model designer. For instance, an item's price or a user's age may be considered a dense feature since many of these values are usually non-zero, while the item categorization information such as a shoe category, could be considered as a sparse feature. In this work we assume that these features are selected by the designer and the training dataset is appropriately labeled. The final relevance score of item i to user u is computed using a scoring function $SF : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$. SF can be a simple inner product, such as a dot product, which is the method we utilize, or a more complex function, such as a DNN.

More formally, the user network processes user feature representations, which are concatenated and passed through a multi-layer perceptron (MLP) to produce a user representation:

$$\mathbf{x_u} = \text{MLP}_u(\mathbf{e_{cat}} \oplus \mathbf{f_{cont}}) \tag{3}$$

where $\mathbf{e_{cat}}$ represents embeddings of categorical features, $\mathbf{f_{cont}}$ represents dense continuous features and $\oplus$ denotes the concatenation operator.

Similarly, the item network processes item feature representations and generates item representations:

$$\mathbf{y_i} = \text{MLP}_i(\mathbf{e_{cat}} \oplus \mathbf{f_{cont}}) \tag{4}$$

Then the relevance between a user u and an item i is calculated as the dot product of the user representation $\mathbf{x_u}$ and the item representation $\mathbf{y_i}$:

$$SF(\mathbf{x_u}, \mathbf{y_i}) = \mathbf{x_u} \mathbf{y_i}^\top \tag{5}$$

Details about the fundamental recommendation model architecture are depicted in Figure 1.

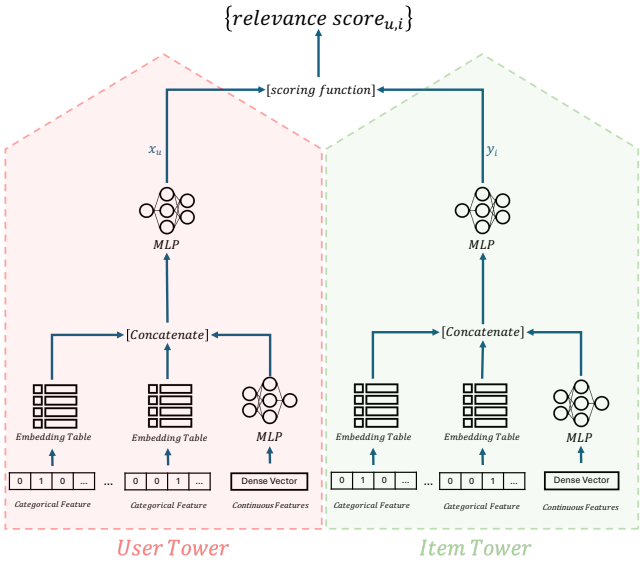

Figure 1: Fundamental Recommendation Model Architecture

3 Related Work

Recommendation System Unlearning: While research on machine unlearning in the context of recommendation systems is still relatively new, there have been recent notable attempts to apply unlearning techniques [6, 25, 38]. For instance, [38] presents a fast and exact unlearning strategy applicable to neighborhood-based recommendation systems using K-nearest neighbors indexing, where at training time the model builds an index of the top-K similar users per user. However, this approach is unsuitable for DNN-based recommendation models where the model uses non-linear functions to learn from user-item interactions. Another approach, GraphEraser proposed in [7], can be applied to recommendation model unlearning since interactions in a recommendation system form a bipartite graph. However, similar to [6, 25] mentioned in Section 1, GraphEraser also relies on pretraining to generate node embeddings for partitioning nodes, resulting in similar drawbacks that affect its applicability to exact machine unlearning. [26] groups similar data together and trains a model sequentially on each subset using curriculum learning, which is a training strategy that present data to the model in an increasing order of complexity to help the model learn more effectively. However, this approach provides two-fold speedup over retraining from scratch, which is still a significant retraining cost.

Meta-Learning in Recommendation Systems: Meta-learning techniques have proven effective in enhancing various aspects of recommendation systems [22, 24, 47]. One popular approach is to adapt the Model-Agnostic Meta-Learning (MAML) algorithm [13] to different use cases [10, 24]. The MAML algorithm, designed to be model agnostic and enable few-shot specialization [10, 13], serves as motivation for our work. We incorporate a MAML-style meta-learning framework in EEMU. EEMU treats the objective of learning to recommend items to a user within a shard as one task, and thus multiple shards would have multiple views of that task for the same user. The goal is to use meta-learning across these multiple shards to re-acquire the degraded collaborative signal due to sharding, as we describe in the next section.

There are several works in recommendation systems and other domains that exploit task similarity to modify the meta-training process, resulting in better model accuracy [44, 48, 49]. Task similarity in meta-learning refers to the measurable relationship between tasks, such as the similarity based on their shared feature representations or how a model learns them. However, we pursue a different goal: improving the inference efficiency. Therefore, we take an orthogonal approach by clustering users based on their task similarity in the learned representation space to enable batch inferencing. In particular, we use the user representations generated from the optimized models to measure how close users are to one another, enabling model serving in batches of similar users.

4 EEMU

EEMU is an exact unlearning approach that is designed for the scenario in which a user may request for deletion of a subset of their interactions that the recommendation model was trained on. At a high level, EEMU enables efficient exact unlearning by partitioning user-item interactions into isolated shards and training separate models per shard. In particular, within each shard, we consider the task of recommending an item to a given user u as a specific task. Thus, the user-item interactions in each shard are first grouped by user then each user's interactions are sliced into support and query sets. After that, in each shard, the support and query sets of all users are used to train a global model using the MAML training as described in Section 2.2. However, EEMU relies on a novel inference adapation where the few-shot examples for inference are sampled from across the shards. Note that as described earlier, MAML samples few-shots from a task-specific dataset. But in EEMU since the task-specific dataset is no longer only available in a single shard it samples these few-shot examples from across the shards. The small adaptation set, sampled across the shards, is used to fine-tune the model for each user, but these fine-tuned models are never stored, preserving exact unlearnability. Finally, predictions from all shards are aggregated using a simple mean, helping recover lost collaborative signals while maintaining shard isolation. In Figure 2, we illustrate EEMU's work flow and describe its steps below.

Sharding: In EEMU, which is a specialization of the SISA paradigm, the first step is dataset *sharding*. Sharding enables efficient exact unlearnability by creating a per-shard model rather than one trained on the whole dataset. EEMU employ random balanced partitioning of user's interactions across all shards. This approach offers the advantage of being computationally efficient as each user-item interaction is randomly assigned to only one of the shards. Thus, shards are mutually exclusive and do not share any user-item interaction. While computationally efficient, random balanced sharding poses a challenge for recommendation systems since user-item interactions contain valuable collaborative information that may be lost when sharded randomly. Each shard has only a fraction of the

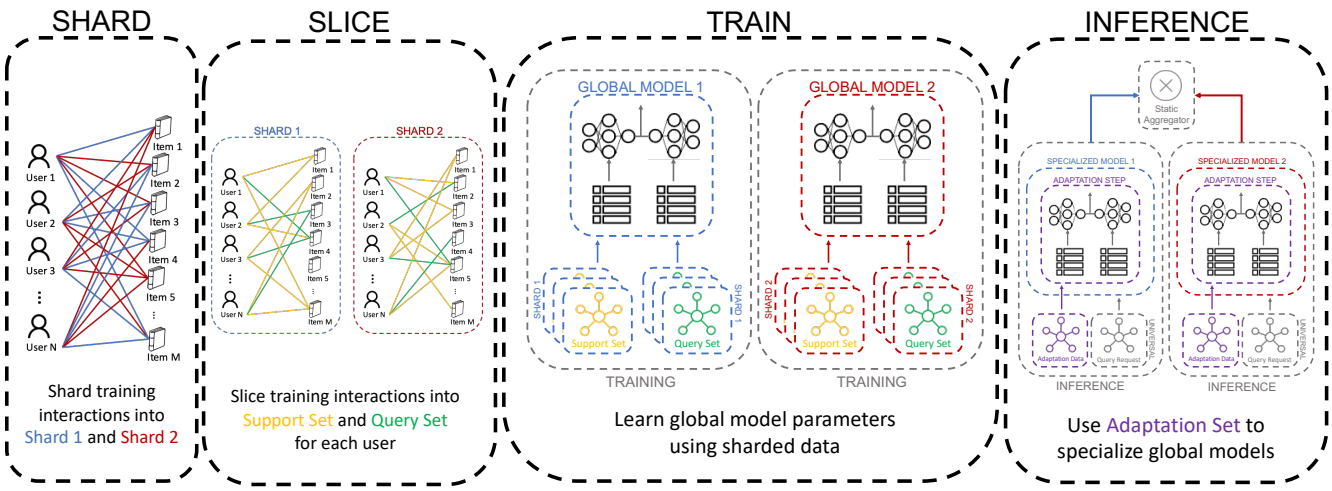

Figure 2: EEMU Work Flow.

per-user interactions thereby potentially impacting the per-shard model performance. However, EEMU addresses this concern by deferring the mitigation of collaborative signal loss to a subsequent step in the EEMU training flow.

Slicing: Recall that EEMU adapts the MAML algorithm by creating multiple per-user recommendation tasks in each shard. Thus, the per-shard model has to be trained using MAML's two-stage optimization process, where subsets of the user-specific data, known as the *support* and *query* sets, are used for local (Equation 1) and global (Equation 2) model updates. Hence, to train the model in each shard, we first group the training samples per user and then slice the per-user training samples in a shard into support and query sets. To ensure balance, we allocate a fixed number of interactions to the query set, ensuring that each user has an equal number of interactions in their query sets across all shards. The remaining interactions are used for the support set. In our current research, we place interactions into support and query sets randomly. However, we acknowledge the potential for further enhancements by intelligently selecting the the support and query sets in future work.

Figure 2 illustrates the sharding and slicing steps from the perspective of all users. In the sharding step, all the interactions are placed into shard1 and shard2 (color coded in the figure). In the slicing step, the data in each shard is grouped by user and their interactions are sliced into support and query sets.

Training: We map the two-stage optimization process of the MAML algorithm presented in Section 2.2 to the non-collaborative sharded training. As shown in the training box in Figure 2, each shard trains its own global model. As described earlier, a key concept of the MAML algorithm is the notion of tasks. Specifically, the model is trained on multiple tasks and then adapted to perform on a specific task. In our framework, we refer to the objective of learning to recommend items within a shard to a user as a task.

Algorithm 1 shows the detailed training procedure of EEMU. First, the global model parameters θ for each shard's model $\mathcal{M}_s^{global}$ are randomly initialized (line 2). Next, a batch of users is sampled from the training data of shard s (line 4). For each user u in the batch, the model computes gradients based on the user's support set, then updates the user-specific parameters θ_u (lines 6-7). The model then evaluates these updated parameters on the user's query set to compute gradients for the global update (line 8). Finally, the global model parameters θ are updated using the aggregated gradients from all users in the batch (line 10). EEMU's training process in Algorithm 1 is similar to MAML's training process. The difference between the two training methods is that EEMU uses multiple shards where each shard is trained in isolation as shown in the For loop in line 1.

Algorithm 1 Training Procedure of EEMU

Input: number of shards S; step size α, β

1: **for** each shard s in S **do**
2: Randomly initialize global model $\mathcal{M}_s^{global}$ parameters θ
3: **while** not converged **do**
4: Sample batch of users B
5: **for** each user u in B **do**
6: Evaluate $\nabla_\theta L(\theta)$ with user u support set
7: Update: $\theta_u = \theta - \alpha \nabla_\theta L(\theta)$
8: Evaluate $\nabla_{\theta_u} L(\theta_u)$ with user u query set
9: **end for**
10: Global update: $\theta \leftarrow \theta - \beta \nabla_{\theta_u} L(\theta_u)$
11: **end while**
12: **end for**

The parameters of the models are trained using a max-margin-based ranking loss function. The objective is to maximize the score of positive examples and at the same time minimize the score of negative examples. Positive examples are items a user interacted with, while negative examples are items the user did not interact with. This is typically done by ensuring that the score of positives is larger than the score of negatives by a fixed margin Δ. Thus, given a positive example (u, i^+) and a negative example (u, i^-), with item

representations $\mathbf{y}_{\mathbf{i}^+}$ and $\mathbf{y}_{\mathbf{i}^-}$, respectively, and user representation $\mathbf{x}_\mathbf{u}$, the loss function is computed as follows:

$$\mathcal{L}(\mathbf{x_u}, \mathbf{y}_{\mathbf{i}^+}) = max(0, SF(\mathbf{x_u}, \mathbf{y}_{\mathbf{i}^-}) - SF(\mathbf{x_u}, \mathbf{y}_{\mathbf{i}^+}) + \Delta) \tag{6}$$

Once the per-shard global model is trained, EEMU can be deployed in any recommendation pipeline. EEMU then uses a novel inference framework to recapture the collaborative signal loss. We describe the inference next.

EEMU Inference: The first step during inference is called the adaptation step in which the model achieves few-shot specialization on a user-by-user basis. Specialization within a shard is achieved by finetuning $\mathcal{M}_s^{global}$ using a small set of adaptation data. Specifically, in EEMU, the adaptation set is constructed by sampling items from the user interaction history from across all shards. Thus, while the global model was trained using only the per-shard data, during inference few shot examples are drawn randomly from across shards. This random sampling from across the shards is one key element of the EEMU inference deployment in a sharded training setting. Since we prioritize efficiency, we sample very few items (a hyperparameter of EEMU) and aim to choose them wisely to better model user preferences.

One of the challenges in recommendation systems is the long-tail distribution of interactions where some items have very few interactions and hence they do not get recommended well. In our prior work, we demonstrated that sampling interactions from the long-tail items is beneficial for improving model utility [1]. Inspired by this work, we create an adaptation set by sampling interactions from across shards with a bias toward items that have received few interactions, as they contain more information about user preferences. For user u, the sampling probability distribution over each item i in user u's interaction history is given by the following softmax distribution:

$$p_i \sim softmax\left(\frac{1}{d_i}\right) \tag{7}$$

where d_i is the number of interactions received by item i.

The adaptation set for each user contains significantly fewer interactions than the support set used during their training. This is why the adaptation process is referred to as the few-shot specialization. We denote the specialized model for a given user u in a shard s by $\mathcal{M}_{s,u}^{special}$. A key thing to note here is that even though the adaptation step produces $\mathcal{M}_{s,u}^{special}$ for each user u in every shard s, $\mathcal{M}_{s,u}^{special}$ is never required to be stored. This ensures that any model trained using EEMU is indeed efficiently and exactly unlearnable. EEMU employs a static aggregator to combine scores from each shard during inference. Among various choices of static, permutation invariant functions, EEMU adopts a straightforward approach of computing the mean across all shards to derive the final score. The inference box in Figure 2 illustrates the process of performing inference on a single incoming query request from a single user.

Algorithm 2 shows the inference procedure of EEMU. First, an adaptation set is created for user u by sampling a few interactions, where the probability for each item i appearing in the sample is given by Equation 7 (line 1). Next, for each shard s in S, the global model $\mathcal{M}_s^{global}$ is specialized for user u using their adaptation set to produce $\mathcal{M}_{s,u}^{special}$ (line 3). Using this specialized model, the scores for an incoming query are computed for each shard (line 4). Finally, the final score is calculated by averaging the scores from all shards (lines 6). EEMU employs a two-stage optimization process similar to MAML within each shard, enabling better generalization across users without extensive fine-tuning during inference. However, unlike MAML, EEMU selects samples for the adaptation set with a probability inversely proportional to the item degree, as shown in Equation 7. Additionally, EEMU computes scores within each shard before aggregating them across all shards, with each shard having access to samples from the whole dataset.

It is important to emphasize that few-shot specialization during inference enables each shard s to access data from all other shards without compromising the isolation guarantees, as the $\mathcal{M}_{s,u}^{special}$ is never required to be stored. As EEMU accesses user-item interactions from multiple shards during inference, it recaptures some of the lost collaborative signal, allowing for improved recommendation performance.

Algorithm 2 Inference Procedure of EEMU

Input: Global models $\mathcal{M}_s^{global}$ for each shard s; number of shards S; user u

1: $AdaptSet_u \leftarrow$ sample few interactions $p_i \sim softmax(\frac{1}{d_i})$
2: **for** each shard s in S **do**
3: $\quad \mathcal{M}_{s,u}^{special} \leftarrow$ Specialize $\mathcal{M}_s^{global}$ using $AdaptSet_u$
4: $\quad SF_{s,u} \leftarrow$ compute scores for incoming query using $\mathcal{M}_{s,u}^{special}$
5: **end for**
6: $FinalScore \leftarrow \frac{1}{S}\sum_{s=1}^{S} SF_{s,u}$
7: **Return:** $FinalScore$

Efficient Exact Unlearnability: To achieve efficient exact unlearning in recommendation systems, minimizing the retraining overhead is crucial. EEMU is structured to minimize the components that require retraining when a deletion request is made. Only the shard containing the deleted interaction needs to be retrained due to the following reasons: 1) partitioning is performed at random with no associated clustering, learnable parameters or pretrained models, 2) aggregation is performed using a non-learnable static function and 3) only $\mathcal{M}_s^{global}$ is part of the stored state in the EEMU work flow. As mentioned in Section 1, this is unlike the frameworks proposed in [6, 7, 25] which require the retraining of the pretrained model used for partitioning and/or the learnable aggregator to ensure exact unlearnability. Moreover, to minimize the retraining overhead, employing a balanced sharding scheme is also crucial. EEMU employes a random balanced sharding scheme and have the following lower bound on the expected retraining overhead:

$$\mathbb{E}(OH) = \sum_{i=1}^{S} t_i \cdot r_i \geq \frac{T}{S}, \tag{8}$$

where t_i is the retraining overhead of shard s_i, r_i is the probability of a retracted interaction being in shard s_i and T is the overhead of retraining all S shards. This lower bound is theoretically validated in [25] assuming no prior knowledge about unlearning request distribution [3]. Thus, EEMU is practically and theoretically a very

efficient model of exact recommendation system unlearning. Adaptation is a crucial component of this framework (as we will show in Section 5.5), and the fact that adapted model parameters are not required to be stored makes EEMU an ideal choice for a system that supports exact unlearning.

5 Experiments

Sharding effectively avoids the necessity to retrain a single centralized model on the entire dataset in response to a deletion request. However, it introduces a critical challenge: the loss of collaborative signal. This collaborative signal loss is an inherent consequence of data fragmentation across shards. What further complicates the matter is that the lost signal cannot be recovered during training through information exchange between the shards. Attempting such an exchange would require full retraining of all shards in response to a single deletion request, rendering the process inefficient. Consequently, there is an anticipated decline in model performance within the sharded training scheme. In this section, we start with comparing the degradation experienced by different sharded training strategies to that observed with EEMU. Following that, we analyze the impact of each sharded training strategy on both retraining and inference complexity. Then, we perform ablation and sensitivity studies to understand the effect of some of the key parameters in our strategy. We also present three case studies in Appendix A to show how EEMU can generalize to another fundamental recommendation model, support user-wise exact unlearning and perform while employing another meta-learning algorithm.

5.1 Datasets

We evaluate the performance of EEMU on four benchmark datasets: MovieLens-1M (ML1M)[**?**], BookCrossing (BX)[**?**], Gowalla[**?**] and Epinions[**?**]. Because we're interested in the top k recommendation task, we use a common pre-processing method to convert these datasets into implicit feedback. As in [1, 19, 47], we divide all the datasets into training, validation and test datasets according to the leave-one-out setting. In the presence of interaction time stamps, we use the latest interaction for each user for testing, the penultimate interaction for validation and the rest for training. The original crawled BookCrossing dataset is extremely noisy, thus, we follow [51] recommendations in processing the dataset to obtain more meaningful results. The same applies to Gowalla and Epinions dataset, therefore we process them using a 20-core setting to get close to the number of interactions in [37]. In MovieLens-1M, the available user features include ID, age, gender and occupation, while item features include ID, genre, year and title. In BookCrossing, user features consist of ID, location and age, and item features include ID, author, publisher, year and title. In Gowalla, there is only one feature for each user and each item which is ID. Finally, in Epinions, users have only IDs as their features, while item features are ID and category. For all users across each dataset, a random training interaction is selected for the query set, while the remaining training interactions are used for the support set. Detailed statistics about the datasets used in our experiments can be found in Table 2.

	ML1M	BX	Gowalla	Epinions
# Users	6040	2297	5992	3666
# Items	3883	4717	5639	3369
# Interactions	1000209	160560	287404	151096
Sparsity	99.957%	99.985%	99.991%	99.988%
Avg User Degree	163.598	67.900	45.965	39.215
Avg Item Degree	254.476	33.065	48.842	42.673
# User Features	4 features	3 features	1 feature	1 feature
# Item Features	4 features	5 features	1 feature	2 features

Table 2: Statistics of the datasets used in the experiments.

5.2 Baselines

We compare EEMU against a number of sharded training strategies that support exact machine unlearning which can have static or trainable aggregation methods. EEMU and all baselines use the same fundamental recommendation model architecture specifications. The baselines are outlined below:

- **RETRAIN** - An unsharded centralized implementation of the recommendation system. To perform exact unlearning on this model, the model has to be retrained on the entire dataset after the deletion of a datapoint.

Static Aggregation

- **MEAN** - Mean aggregation across the shards is utilized by this baseline to calculate the final score for each item.
- **MRA** - A sharded training strategy that uses Median rank aggregation (MRA) [12] across the shards to get the final rank for all items . MRA is shown to be effective in aggregating item ranks from different recommenders in [32].
- **ItemC** - The final score for each item in this baseline is computed by taking the weighted mean of the item scores across shards, with the weights being proportional to the number of items each shard has for a user.
- **JS_Div** - This baseline uses weighted mean aggregation across the shards to compute the final score for each item. The weights are proportional to the similarity between the probability distribution of the unsharded data and the data in each shard. The probability distribution similarity is measured using Jensen–Shannon divergence.

Trainable Aggregation

- **GraphEraser [7]** - Utilizes an aggregation layer trained with data from all shards to obtain a set of fixed weights for aggregating the final item scores across all users. The aggregation layer, however, must be retrained following a deletion request.
- **RecEraser [6]** - Implements an attention aggregation layer to compute the final score for each item, with weights that vary across users and items. As with *GraphEraser*, retraining the aggregation layer is necessary after a deletion request.
- **UltraRE [25]** - Employs a logistic regression-based aggregator to compute the final score for each item, using weights obtained from training data across all shards. Like *GraphEraser*, the weights of the aggregation layer remains fixed for all users and requires retraining after a deletion request.

	MovieLens-1M		BookCrossing		Gowalla		Epinions	
Baseline	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
RETRAIN	0.07235	0.03555	0.03135	0.01446	0.05391	0.02531	0.03219	0.01518
MRA	0.05927 (18%)	0.02837 (20%)	0.02003 (36%)	0.00988 (32%)	0.01218 (77%)	0.00609 (76%)	0.01937 (40%)	0.00909 (40%)
ItemC	0.05977 (17%)	0.02934 (17%)	0.02481 (21%)	0.01230 (15%)	0.01736 (68%)	0.00874 (65%)	0.01991 (38%)	0.00904 (40%)
JS_Div	0.06026 (17%)	0.02876 (19%)	0.02525 (19%)	0.01204 (17%)	0.01636 (70%)	0.00756 (70%)	0.01882 (42%)	0.00915 (40%)
Mean	0.05993 (17%)	0.02908 (18%)	0.02537 (19%)	0.01317 (9%)	0.01643 (70%)	0.00761 (70%)	0.01964 (39%)	0.00906 (40%)
GraphEraser	0.06093 (16%)	0.02930 (18%)	0.02569 (18%)	0.01236 (15%)	0.01752 (68%)	0.00841 (67%)	0.01909 (41%)	0.00886 (42%)
RecEraser	0.05613 (22%)	0.02684 (25%)	0.01872 (40%)	0.00871 (40%)	0.02170 (60%)	0.01103 (56%)	0.02182 (32%)	0.00953 (37%)
UltraRE	0.05894 (19%)	0.02863 (19%)	0.02481 (21%)	0.01294 (11%)	0.01652 (70%)	0.00789 (69%)	0.01909 (41%)	0.00896 (41%)
EEMU	0.06589 **(9%)**	0.03299 **(7%)**	0.03962 **(0%)**	0.01985**(0%)**	0.05674 **(0%)**	0.02691 **(0%)**	0.02591 **(20%)**	0.01218 **(20%)**

Table 3: Comparison against baselines by HR@10 and NDCG@10.

	MovieLens-1M		BookCrossing		Gowalla		Epinions	
Baseline	HR@20	NDCG@20	HR@20	NDCG@20	HR@20	NDCG@20	HR@20	NDCG@20
RETRAIN	0.11738	0.04738	0.05877	0.02067	0.09162	0.03713	0.05728	0.02220
MRA	0.10050 (14%)	0.03900 (18%)	0.03222 (45%)	0.01218 (41%)	0.02086 (77%)	0.00746 (80%)	0.03573 (38%)	0.01291 (42%)
ItemC	0.10695 (9%)	0.04092 (14%)	0.03891 (33%)	0.01581 (24%)	0.02954 (68%)	0.01174 (68%)	0.03328 (42%)	0.01185 (47%)
JS_Div	0.10629 (9%)	0.04038 (15%)	0.03962 (33%)	0.01668 (19%)	0.02804 (69%)	0.01004 (73%)	0.03464 (40%)	0.01211 (45%)
Mean	0.10679 (9%)	0.04041 (15%)	0.04052 (31%)	0.01693 (18%)	0.02870 (69%)	0.01070 (71%)	0.03544 (39%)	0.01261 (43%)
GraphEraser	0.10695 (9%)	0.04031 (15%)	0.04049 (31%)	0.01643 (21%)	0.02887 (68%)	0.01104 (70%)	0.03519 (39%)	0.01210 (45%)
RecEraser	0.09586 (18%)	0.03702 (22%)	0.03439 (41%)	0.01138 (45%)	0.03471 (62%)	0.01298 (65%)	0.03710 (35%)	0.01308 (41%)
UltraRE	0.10232 (13%)	0.03938 (17%)	0.04179 (29%)	0.01762 (15%)	0.02837 (69%)	0.01089 (71%)	0.03519 (39%)	0.01217 (45%)
EEMU	0.11474 **(2%)**	0.04539 **(4%)**	0.05529 **(6%)**	0.02350 **(0%)**	0.07977 **(13%)**	0.03287 **(11%)**	0.04092 **(29%)**	0.01594 **(28%)**

Table 4: Comparison against baselines by HR@20 and NDCG@20.

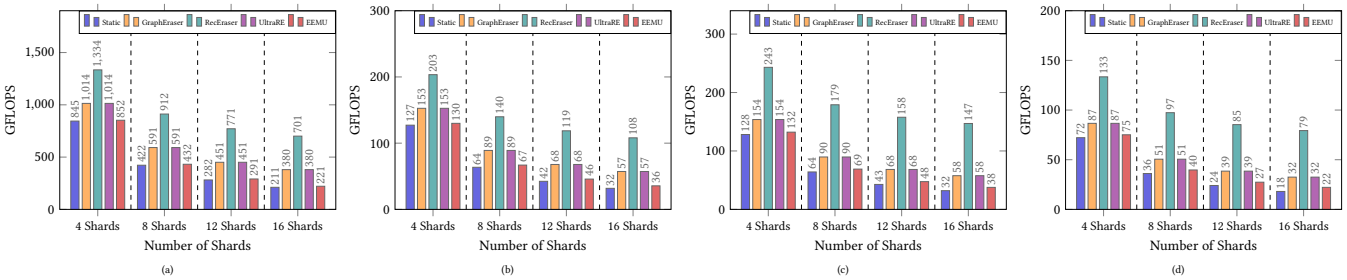

Figure 3: Floating point operations (GFLOPs) for one epoch of retraining for: (a) MovieLens-1M (b) BookCrossing (c) Gowalla (d) Epinions. Retraining without sharding (RETRAIN) takes 3378.92, 508.72, 512.52, and 288.72 GFLOPS for MovieLens-1M, BookCrossing, Gowalla, and Epinions, respectively, and these values are omitted from the figure for clarity.

Dataset / Baseline	Static	GraphEraser	RecEraser	UltraRE	EEMU
MovieLens-1M	0.5817	0.5818	2.6816	0.5818	0.5823
BookCrossing	0.8224	0.8225	3.3733	0.8225	0.8231
Gowalla	0.4286	0.4287	3.4780	0.4287	0.4290
Epinions	0.3390	0.3391	2.1610	0.3391	0.3394

Table 5: Floating point operations (GFLOPs) for one inference request.

With exception of *RETRAIN*, all the included baselines use a standard sharded implementation of the recommendation system, similar to the one used in SISA [3] that employs random balanced sharding to target exact machine unlearning. Also, we are not suggesting the use of trainable aggregators in an exact unlearning setting; in fact, this goes against the motivation we laid out in the introduction. However, we are including *GraphEraser*, *RecEraser* and *UltraRE* as baselines to have a fair comparison against a variety of state-of-the-art recommendation unlearning methods.

5.3 Evaluation Criteria

Ranking. To evaluate the ranking performance of each model we employ two metrics: Hit Rate @ k (HR@k) and Normalized Discounted Cumulative Gain @ k (NDCG@k). The former determines if a candidate item is among the top k recommendations, while the latter measures how closely the recommended item is placed to the top of the list. Since retraining a centralized model serves as the gold standard in terms of model performance, we also include the percentage degradation in parentheses for each baseline compared to RETRAIN as the objective of enhanced sharded training is to minimize this degradation despite sharding. A value of 0% in parentheses indicates no degradation (including when the baseline outperforms RETRAIN).

Retraining Complexity. The complexity of training a model dictates the overhead incurred to retrain in the event of a deletion request. Thus, we compare the number of GFLOPS required on average per epoch to retrain a shard of EEMU to baselines. The choice of using GFLOPS to report the complexity is made following the recommendation of [39].

Inference Complexity. Inference complexity is a crucial factor as it affects serving latency. In a sharded setup, in addition to the computations needed to perform inference in each shard, there is an additional overhead incurred by aggregation across all shards. Thus, we compare EEMU to all baselines in inference complexity for a single request (also in GFLOPS).

5.4 Results

In this section, we summarize our key findings from our main experiments in alignment with our evaluation criteria.

Ranking: Our primary ranking results for Top-k recommendation are presented in Tables 3 and 4 for k=10 and k=20, respectively. The biggest takeaway from both tables is that EEMU significantly outperforms other sharded strategies in an exact unlearning setting, including those based on learnable aggregators trained using data from all shards. We postulate that this superiority stems from the localized parameterization of each shard within different subspaces, facilitated by optimization on distinct tasks — a phenomenon akin to Mixture-of-Experts models [40]. These findings serve to validate the efficacy of meta-learning in addressing the loss of collaborative signal resulting from random interaction sharding. What is more interesting, however, is that EEMU also outperforms RETRAIN in some situations. While we leave the detailed understanding of this phenomenon to follow-up work, we hypothesize that this is due to the power of meta-learning to effectively learn in a few-shot setting [15, 41], in conjunction with the specialization afforded by the adaptation step.

Training Complexity: So far, we have established the improved model performance with EEMU over other baseline methods and showcased the effectiveness of meta-learning within a sharded system. Nevertheless, the practicality of this solution hinges on ensuring that the added complexity stemming from meta-learning does not outweigh the advantages gained in reducing the retraining complexity, which is a core benefit of sharding. Encouragingly, the data presented in Figure 3 illustrate that EEMU is almost as efficient as baselines that employ the standard sharding scheme with static aggregation (Static) in terms of the average number of floating-point operations necessary to retrain a single shard. What is even more promising is that this number can be further diminished by scaling up the number of shards, rendering EEMU an even more efficient retraining option. It is also evident from the figure that employing trainable aggregation layers (GraphEraser, UltraRE and UltraRE) hurts the retraining efficiency, as the aggregation layers will also be retrained. It is important to note that augmenting the number of shards does introduce the risk of increased loss of collaborative signal. However, as we will demonstrate in Section 5.5, EEMU exhibits remarkable resilience to the expansion of shard count, suffering only from minor additional performance degradation.

Inference Complexity: To further reinforce the computational efficiency of EEMU, Table 5 presents the number of floating-point operations necessary, on average, to fulfill a single inference request. Here, a single inference request entails generating a Top-K recommendation list for a single user. Notably, despite the additional complexity introduced by the adaptation step during inference, EEMU demonstrates a minimal increase in the number of floating-point operations required to serve a single inference request in comparison to the baselines that do not involve an adaptation step. This efficiency is partly attributed to the fact that the number of items used to adapt each user during inference can be just one, as we will examine in Section 5.5. This makes EEMU highly scalable for inference.

5.5 Ablation and Sensitivity Analysis

In this section, we present abltation studies on some key parameters and components of our proposed strategy. Particularly, we would like to answer the following questions: (i) How significant is the adaptation step during inference to the overall performance of EEMU? (ii) Can adaptation benefit other sharded models that do not employ meta-learning during training? (iii) Does biasing toward unpopular items during adaptation set creation positively improve EEMU's ranking performance (iv) If adaptation is crucial, how large should the adaptation set be for each user? (v) What is the effect of increasing the number of shards on model performance? And finally, (vi) Given that meta-learning performs well in few-shot learning settings, is one shard of data sufficient to achieve good performance, or is aggregating predictions from all shards necessary?

Importance of adaptation: To comprehend the significance of the adaptation step during inference, we conduct a comparative analysis of the outcomes produced by a trained EEMU model with and without the inclusion of an adaptation step prior to ranking. The results are visualized in Figure 4. It is evident that, across all datasets, the incorporation of the adaptation step significantly enhances the recommender's performance. To provide a deeper analysis of the results in Figure 4 and to further confirm the importance of adaptation to EEMU, we report EEMU's shard-level recommendation performance (HR@10) with and without adaptation in Table 6. It is clear that adaptation plays a vital role in EEMU, as it enhances recommendation performance compared to EEMU without adaptation across all shards and datasets included in the experiment. Interestingly, however, the impact of adaptation on a model not initially trained with meta-learning (Standard Sharded Training) is more nuanced, resulting in a less effective recommender on half of the datasets and a slightly improved one on the remaining half. This

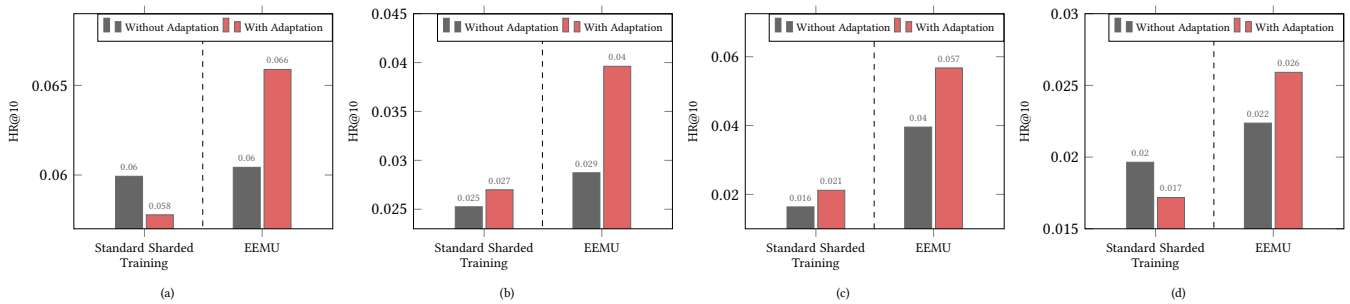

Figure 4: Ablation on adaptation step importance showing HR@10 for: (a) MovieLens-1M. (b) BookCrossing. (c) Gowalla. (d) Epinions.

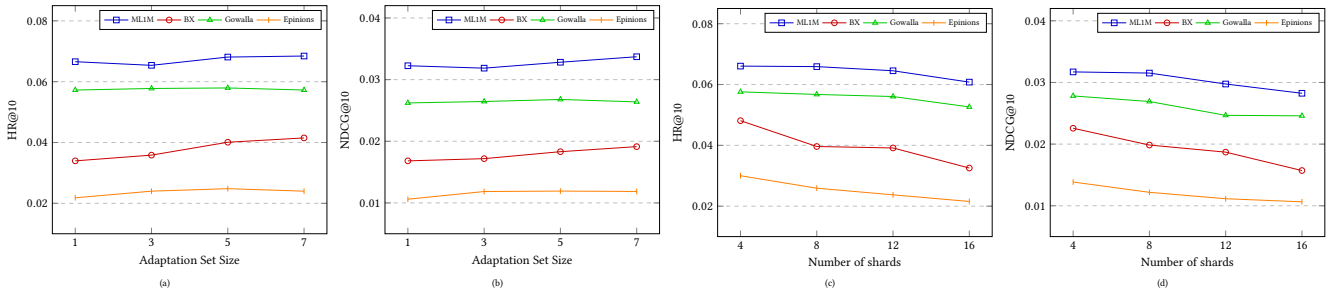

Figure 5: (a) Ablation on adaptation set size showing HR@10 for all datesets. (b) Ablation on adaptation set size showing NDCG@10 for all datesets. (c) Ablation on number of shards showing HR@10 for all datesets. (d) Ablation on number of shards showing NDCG@10 for all datesets.

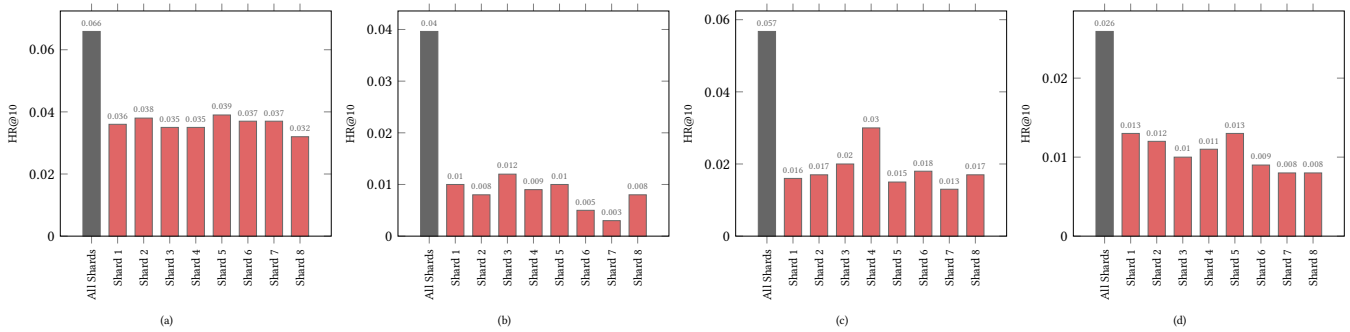

Figure 6: Ablation on aggregation importance showing HR@10 for: (a) MovieLens-1M. (b) BookCrossing. (c) Gowalla. (d) Epinions.

variability in recommendation performance suggests that models not trained with meta-learning might not be inherently optimized for generalization to user preferences with limited data, in contrast to the performance seen in EEMU. To understand the reason for this variability, we analyzed the entropy of user degrees in the four datasets. MovieLens-1M and Epinions have larger entropy values than BookCrossing and Gowalla, indicating more diverse user interactions. Adaptation degraded performance in high-entropy datasets (MovieLens-1M and Epinions), potentially because user preferences are spread across many items, making small updates destabilizing. In contrast, lower-entropy datasets (BookCrossing and Gowalla) benefited from adaptation, as users interact with a more concentrated set of items, likely making their preferences more predictable. This observation underscores the interdependency between the effectiveness of inference-time adaptation and a training procedure based on meta-learning.

Importance of bias during adaptation set creation: Unpopular items have been theoretically proven in [1] to provide more information about user preferences than other items. Therefore, in EEMU, the adaptation set is constructed with a bias toward these unpopular items. We expect EEMU to perform better when the adaptation set is constructed using biased sampling rather than

	MovieLens-1M		BookCrossing		Gowalla		Epinions	
EEMU	w/o Adapt	w/ Adapt	w/o Adapt	w/ Adapt	w/o Adapt	w/ Adapt	w/o Adapt	w/ Adapt
Shard 1	0.03477	0.0368	0.01001	0.01027	0.01469	0.01647	0.01146	0.01366
Shard 2	0.03626	0.03818	0.00871	0.009	0.01769	0.01793	0.00955	0.0125
Shard 3	0.03079	0.03546	0.01045	0.01288	0.02019	0.021	0.00682	0.01054
Shard 4	0.03146	0.03515	0.0064	0.00947	0.03037	0.031	0.00873	0.01135
Shard 5	0.03642	0.03966	0.00653	0.0109	0.01235	0.0155	0.01037	0.01396
Shard 6	0.03775	0.03792	0.00479	0.00581	0.01669	0.01897	0.00655	0.00919
Shard 7	0.03526	0.03788	0.00218	0.00361	0.01285	0.01315	0.00791	0.00888
Shard 8	0.03013	0.03241	0.00784	0.00899	0.01469	0.0176	0.00846	0.00885

Table 6: Ablation on adaptation step importance at the shard level showing HR@10.

Dataset	Random Sample	Biased Sample	*Improv.*
ML1M	0.06324	0.06589	+%4.9
BX	0.03657	0.03962	+%8.3
Gowalla	0.05491	0.05674	+%3.3
Epinions	0.02282	0.02591	+%13.5

Table 7: Ablation on sampling bias importance during adaptation set creation showing HR@10 for all datasets. *Improv.* highlights the percentage improvement of biased sampling over random sampling.

random sampling. To evaluate the effect of biased sampling, we conducted two EEMU experiments differing in how the adaptation set was created: one used a random sample of items, while the other used a sample biased toward unpopular items. The results of these experiments across all datasets are reported in Table 7. From the results, we can conclude that biasing toward unpopular items during adaptation set creation positively improve EEMU's ranking performance.

Size of adaption set: Intuitively, one might expect that a larger adaptation set for each user would lead to improved recommender performance. However, the drawback of having a larger adaptation set is the increased demand for floating-point operations during inference. Fortunately, as demonstrated in Figure 5(a), even with an adaptation set of just one edge, the performance remains quite robust. Therefore, depending on the available computational resources for inference, utilizing a single edge for user adaptation during inference can yield significant improvements. Naturally, when there is a more generous computational budget, expanding the size of the adaptation set becomes advantageous.

Effect of shard count: Once more, intuition might suggest that an increase in shard count would correspond to a decline in model performance, primarily due to the heightened loss of collaborative signal. Interestingly, as observed in Figure 5(c), while the expected trend is present, the actual magnitude of performance degradation stemming from an increased shard count is remarkably minimal. Therefore, there is a compelling argument to be made for employing a larger number of shards, especially when the primary objective is to minimize retraining time.

Importance of aggregation: Previous studies [15, 41] have demonstrated that meta-learning models excel in few-shot learning settings. Building on this, EEMU has shown strong performance in adapting the sharded model to mitigate the loss of collaboration caused by sharding. However, an important question remains: is one shard of data sufficient to achieve good performance, or is aggregating predictions from all shards necessary? To investigate this, we evaluated the ranking performance of each shard in EEMU using the same adaptation set for all shards and compared it to the performance achieved by aggregating predictions from all shards, which is the normal operation of EEMU. The results are presented in Figure 6. The results clearly show that, despite meta-learning's effectiveness in few-shot settings, aggregating predictions from all shards is necessary, as it significantly improves ranking performance across all datasets compared to adapting a single shard.

6 Batch Inferencing

In Section 5.4, we demonstrated the efficiency of EEMU in fulfilling single-user inference requests. We are also interested in making EEMU more efficient in fulfilling multiple user inference requests using the same computing fabric. As explained earlier, EEMU fine-tunes a global model $\mathcal{M}_s^{global}$ for a given user u in a shard s to get $\mathcal{M}_{s,u}^{special}$ before it can generate a recommendation list for that user. Thus, to generate recommendation lists for multiple users, EEMU can either adapt users serially using one computing node or in parallel using multiple nodes. We are interested, however, in making EEMU more efficient and enabling it to adapt multiple users in parallel on the same node. We accomplish this by clustering users based on their task similarity into K clusters and then performing an adaptation step on $\mathcal{M}_s^{global}$ for all users that fall in the same cluster within a shard. This adaption step would generate $\mathcal{M}_{s,c}^{special}$ for each cluster $c \in K$ in shard $s \in S$, which would be used to fulfill inference requests for all users in the cluster. It is important to note that we cluster users using $\mathcal{M}_s^{global}$ which is stored within each shard. Next, we will explain our clustering methodology and discuss our findings.

6.1 Clustering based on task similarity

We use the dense user representation $\mathbf{x_u}$, which is generated by the user tower in the optimized $\mathcal{M}_s^{global}$, as a proxy to assess users'

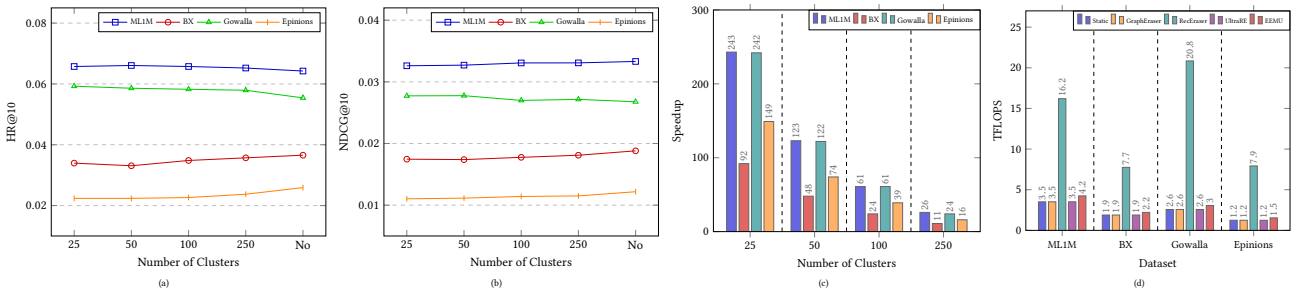

Figure 7: (a) HR@10 of EEMU in batch inferencing mode for all datesets. (b) NDCG@10 of EEMU in batch inferencing mode for all datesets. (c) Speedup of EEMU in batch inferencing mode compared to single inference mode for all datesets. (d) Floating point operations (TFLOPs) required for serving all users of EEMU and baselines in batch inferencing mode. (For (a) and (b) "No" represents EEMU in single inference mode)

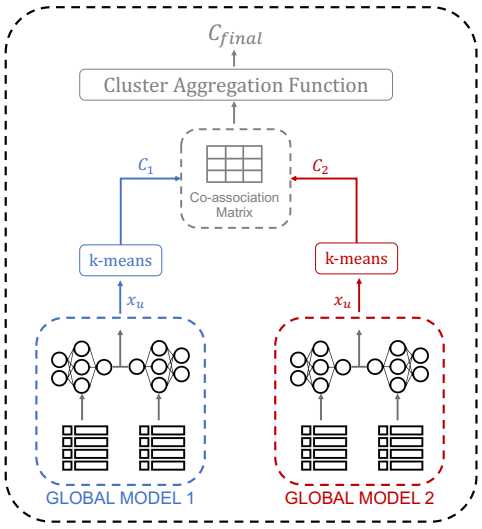

Figure 8: Clustering based on task similarity

task similarities and feed it to the chosen clustering algorithm. Considering user representations as input to the clustering algorithm will lead to having multiple cluster labels for the same user, each one belonging to a shard, thereby creating a cluster ensemble problem [2, 14]. Thus, we use the simple yet effective k-means clustering algorithm [27] to cluster users within a shard. Then, we aggregate the user cluster labels from each shard by generating a co-association matrix and feeding it to a cluster ensemble aggregation function. This procedure is depicted in Figure 8. Each entry in the co-association matrix represents the similarity between two users, calculated as the frequency of their co-assignment to clusters divided by the total number of shards, which is defined as follows:

$$\mathbf{CO}[u_i, u_j] = \frac{1}{S}\sum_{s=1}^{S} \mathbb{I}(\mathbf{C_s}[u_i] == \mathbf{C_s}[u_j]), \tag{9}$$

where C_s is the cluster labels for shard s and $u_i, u_j \in \mathcal{U}$.

Algorithm 3 shows the clustering procedure based on task similarity. First, for each shard s, user representations $\mathbf{x}_u$ are extracted from the corresponding global model $\mathcal{M}_s^{global}$ (line 2). These representations are then clustered into K clusters using the k-means algorithm, producing shard-specific cluster labels C_s (line 3). Next, a co-association matrix **CO** is initialized to aggregate clustering results from all shards (line 5). For each pair of users (u_i, u_j), the matrix is updated by counting the number of shards where the users are assigned to the same cluster (lines 6-7). Finally, the aggregated cluster labels C_{final} are determined by applying a cluster aggregation function to the co-association matrix **CO** (line 9).

While one could simply avoid generating the co-association matrix in favor of efficiency and aggregate the cluster labels based on majority voting, we have chosen to generate the co-association matrix and use k-means as an aggregation function due to its effectiveness, based on our initial results. The aggregated cluster labels will then be used by each shard to perform the adaptation step and generate $\mathcal{M}_{s,c}^{special}$. Upon receiving a data deletion request, it's noteworthy that we only need to execute the clustering algorithm on the retrained shard and the cluster ensemble aggregation function. This efficient process is possible because we run the k-means clustering algorithm on each shard independently, without needing to involve other shards.

Algorithm 3 Clustering based on task similarity

Input: Global optimized models $\mathcal{M}_s^{global}$; number of clusters K; number of shards S
Output: Aggregated cluster labels $\mathbf{C}_{final}$

1: **for** each shard s in S **do**
2: $\quad \mathbf{x_u} \leftarrow$ user representations from $\mathcal{M}_s^{global}$
3: $\quad \mathbf{C_s} \leftarrow$ shard labels by running **k-means**$(K, \mathbf{x_u})$
4: **end for**
5: Initialize co-association matrix $\mathbf{CO} \leftarrow \mathbf{0}$
6: **for** each pair of users (u_i, u_j) **do**
7: $\quad \mathbf{CO}[u_i, u_j] \leftarrow \sum_s \mathbb{I}(\mathbf{C_s}[u_i] == \mathbf{C_s}[u_j])/S$
8: **end for**
9: $\mathbf{C}_{final} \leftarrow$ **cluster_aggregation_function**$(\mathbf{CO})$
10: **return** $\mathbf{C}_{final}$

6.2 Effect on ranking performance and efficiency

We ran EEMU in batch inferencing mode with a varying number of clusters and reported the results in Figure 7. It is reasonable to expect that a reduction in the number of clusters could result in a significant decline in recommender performance, as more users would share the same $\mathcal{M}_{s,c}^{special}$. However, as shown in Figure 7(a) and Figure 7(b), even with just 25 clusters, the performance remains quite stable. We hypothesize that this occurs because clustering enables EEMU to finetune $\mathcal{M}_{s}^{global}$ using the adaptation sets of similar users within a cluster.

This has meaningful practical implications as can be inferred from the serving time speedup and the floating points required to serve all users in the datasets, reported in Figure 7(c) and Figure 7(d), respectively. Moreover, a drawback of MAML and therefore EEMU, is that the adaptation step adds inference latency if requests are served serially. On the other hand, fully parallelizing inference requests can lead to a blow-up of memory as each user would concurrently require an $\mathcal{M}_{s,u}^{special}$. For instance, inference batch sizes in large scale recommendation systems can exceed 64k [29], in which case naive parallelization of inference will result in over 64k copies of the model, which is intractable. However, our approach only requires as many specialized models as there are clusters. Thus, our scheme not only represents a much more efficient way to perform inference in EEMU but also meaningfully moves the needle in making MAML more practical for recommendation systems.

Hence, depending on the number of computing nodes available for inference, using EEMU in batch inferencing mode can be a viable option to increase the efficiency of inference requests for users that fall in the same cluster. Nonetheless, it would be valuable to explore various clustering algorithms or alternative cluster ensemble aggregation functions; however, we defer this exploration to future research work.

7 Future Work and Conclusion

In this work, we explore the utilization of meta-learning for exact machine unlearning in the context of deep learning-based recommendation systems. To render exact unlearning feasible, a central dataset is typically split into multiple shards, each containing a replica of the model. In this setup, only a single shard necessitates retraining in response to a data deletion request. However, this sharding approach, while effective in this regard, introduces a challenge by causing a loss of collaborative signal in recommendation systems, consequently leading to reduced recommendation quality.

To address this issue, we introduce EEMU, a meta-learning approach designed to alleviate the adverse effects of collaborative signal loss resulting from sharding. EEMU excels in situations involving random sharding and does not rely on complex sharding strategies, which are common in prior work within this domain. Our results affirm that EEMU significantly outperforms various sharded models, even those incorporating trainable parameters during aggregation, all while maintaining computational efficiency. Consequently, our work provides an accessible pathway for integrating meta-learning into the realm of machine unlearning.

However, despite the strength of our findings, we identify two unanswered questions that merit attention in future research. The first question revolves around understanding the effect of the choice of the clustering algorithm and the cluster ensemble aggregation function on recommendation performance in the batch inferencing setting. The second question focuses on determining the most suitable choices for the support and query sets within the meta-learning algorithm in the sharded setting to optimize the quality of the recommendation system. These questions represent areas that warrant further exploration and investigation, some of which we are actively pursuing.

Acknowledgments

We sincerely thank all the reviewers for their time and constructive comments. This material is based upon work supported by Defense Advanced Research Projects Agency (DARPA) under Contract Nos. HR001120C0088, NSF award number 2224319, REAL@USC-Meta center, and VMware gift. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. The authors used ChatGPT4o to revise the text in the paper to correct typos and grammatical errors.

References

[1] Keshav Balasubramanian, Abdulla Alshabanah, Elan Markowitz, Greg Ver Steeg, and Murali Annavaram. 2024. Biased User History Synthesis for Personalized Long-Tail Item Recommendation. In *Proceedings of the 18th ACM Conference on Recommender Systems* (Bari, Italy) *(RecSys '24)*. Association for Computing Machinery, New York, NY, USA, 189–199. https://doi.org/10.1145/3640457.3688141

[2] Tossapon Boongoen and Natthakan Iam-On. 2018. Cluster ensembles: A survey of approaches with recent extensions and applications. *Computer Science Review* 28 (2018), 1–25.

[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] California State Legislature. 2018. California Consumer Privacy Act (CCPA). California Civil Code. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375 Assembly Bill No. 375, Title 1.81.5, §§ 1798.100-1798.199.

[5] Yinzhi Cao and Junfeng Yang. 2015. Towards Making Systems Forget with Machine Unlearning. In *2015 IEEE Symposium on Security and Privacy*. 463–480. https://doi.org/10.1109/SP.2015.35

[6] Chong Chen, Fei Sun, Min Zhang, and Bolin Ding. 2022. Recommendation unlearning. In *Proceedings of the ACM Web Conference 2022*. 2768–2777.

[7] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. 2022. Graph unlearning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 499–513.

[50]]Gowalla-DS Eunjoon Cho, Seth A. Myers, and Jure Leskovec. [n. d.]. *Gowalla Dataset*. https://snap.stanford.edu/data/loc-gowalla.html

[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA.

[10] Manqing Dong, Feng Yuan, Lina Yao, Xiwei Xu, and Liming Zhu. 2020. Mamo: Memory-augmented meta-optimization for cold-start recommendation. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 688–697.

[11] Thorsten Eisenhofer, Doreen Riepel, Varun Chandrasekaran, Esha Ghosh, Olga Ohrimenko, and Nicolas Papernot. 2023. Verifiable and Provably Secure Machine Unlearning. arXiv:2210.09126 [cs.LG]

[12] Ronald Fagin, Ravi Kumar, and Dandapani Sivakumar. 2003. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 301–312.

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*. PMLR, 1126–1135.

[14] Ana LN Fred and Anil K Jain. 2005. Combining multiple clusterings using evidence accumulation. *IEEE transactions on pattern analysis and machine intelligence* 27,

6 (2005), 835–850.
[15] Hassan Gharoun, Fereshteh Momenifar, Fang Chen, and Amir H. Gandomi. 2023. Meta-learning approaches for few-shot learning: A survey of recent advances. arXiv:2303.07502 [cs.LG]
[16] Laura Graves, Vineel Nagisetty, and Vijay Ganesh. 2021. Amnesiac machine learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11516–11524.
[17] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. 2020. Certified data removal from machine learning models. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 359, 11 pages.
[18] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. *CoRR* abs/1703.04247 (2017). arXiv:1703.04247 http://arxiv.org/abs/1703.04247
[19] 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.
[20] Dan Hendrycks and Kevin Gimpel. 2016. Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *CoRR* abs/1606.08415 (2016). arXiv:1606.08415 http://arxiv.org/abs/1606.08415
[21] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*. 263–272. https://doi.org/10.1109/ICDM.2008.22
[22] Minseok Kim, Hwanjun Song, Yooju Shin, Dongmin Park, Kijung Shin, and Jae-Gil Lee. 2022. Meta-learning for online update of recommender systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 4065–4074.
[23] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).
[24] Hoyeop Lee, Jinbae Im, Seongwon Jang, Hyunsouk Cho, and Sehee Chung. 2019. Melu: Meta-learned user preference estimator for cold-start recommendation. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1073–1082.
[25] Yuyuan Li, Chaochao Chen, Yizhao Zhang, Weiming Liu, Lingjuan Lyu, Xiaolin Zheng, Dan Meng, and Jun Wang. 2024. Ultrare: Enhancing receraser for recommendation unlearning via error decomposition. *Advances in Neural Information Processing Systems* 36 (2024).
[26] Yuyuan Li, Chaochao Chen, Xiaolin Zheng, Junlin Liu, and Jun Wang. 2024. Making recommender systems forget: Learning and unlearning for erasable recommendation. *Knowledge-Based Systems* 283 (2024), 111124. https://doi.org/10.1016/j.knosys.2023.111124
[27] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.
[28] Alessandro Mantelero. 2013. The EU Proposal for a General Data Protection Regulation and the roots of the 'right to be forgotten'. *Computer Law & Security Review* 29, 3 (2013), 229–235. https://doi.org/10.1016/j.clsr.2013.03.010
[29] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *CoRR* abs/1906.00091 (2019). arXiv:1906.00091 http://arxiv.org/abs/1906.00091
[30] A Nichol. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).
[31] Office of the Privacy Commissioner of Canada. 2018. Announcement: Privacy commissioner seeks Federal Court determination on key issue for Canadians' online reputation. https://www.priv.gc.ca/en/opc-news/news-and-announcements/2018/an_181010/
[32] Samuel E. L. Oliveira, Victor Diniz, Anisio Lacerda, Luiz Merschmanm, and Gisele L. Pappa. 2020. Is Rank Aggregation Effective in Recommender Systems? An Experimental Analysis. *ACM Trans. Intell. Syst. Technol.* 11, 2, Article 16 (jan 2020), 26 pages. https://doi.org/10.1145/3365375
[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
[34] Steffen Rendle. 2010. Factorization Machines. In *2010 IEEE International Conference on Data Mining*. 995–1000. https://doi.org/10.1109/ICDM.2010.127
[50]]ml1m-DS GroupLens Research. [n. d.]. *MovieLens 1M Dataset*. https://grouplens.org/datasets/movielens/1m/
[50]]Epinions-DS Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. [n. d.]. *Epinions Social Network Dataset*. https://snap.stanford.edu/data/soc-Epinions1.html
[37] Matthew Richardson, Rakesh Agrawal, and Pedro Domingos. 2003. Trust Management for the Semantic Web. In *The Semantic Web - ISWC 2003*, Dieter Fensel, Katia Sycara, and John Mylopoulos (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 351–368.
[38] Sebastian Schelter, Mozhdeh Ariannezhad, and Maarten de Rijke. 2023. Forget Me Now: Fast and Exact Unlearning in Neighborhood-based Recommendation. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 2011–2015. https://doi.org/10.1145/3539618.3591989
[39] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. 2020. Green AI. *Commun. ACM* 63, 12 (nov 2020), 54–63. https://doi.org/10.1145/3381831
[40] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*.
[41] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. 2019. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 403–412.
[42] Anvith Thudi, Gabriel Deza, Varun Chandrasekaran, and Nicolas Papernot. 2022. Unrolling SGD: Understanding factors influencing machine unlearning. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 303–319.
[43] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. 2022. On the necessity of auditable algorithmic definitions for machine unlearning. In *31st USENIX Security Symposium (USENIX Security 22)*. 4007–4022.
[44] Jieyu Yang, Zhaoxin Huan, Yong He, Ke Ding, Liang Zhang, Xiaolu Zhang, Jun Zhou, and Linjian Mo. 2022. Task Similarity Aware Meta Learning for Cold-Start Recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management* (Atlanta, GA, USA) *(CIKM '22)*. Association for Computing Machinery, New York, NY, USA, 4630–4634. https://doi.org/10.1145/3511808.3557709
[45] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 269–277.
[46] 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 *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 974–983.
[47] Yin Zhang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Lichan Hong, and Ed H Chi. 2021. A model of two tales: Dual transfer learning framework for improved long-tail item recommendation. In *Proceedings of the web conference 2021*. 2220–2231.
[48] Pan Zhou, Yingtian Zou, Xiao-Tong Yuan, Jiashi Feng, Caiming Xiong, and Steven Hoi. 2021. Task similarity aware meta learning: theory-inspired improvement on MAML. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (Proceedings of Machine Learning Research, Vol. 161)*, Cassio de Campos and Marloes H. Maathuis (Eds.). PMLR, 23–33. https://proceedings.mlr.press/v161/zhou21a.html
[49] Zhipeng Zhou, Wei Gong, and Haoquan Zhou. 2023. Target-oriented Few-shot Transferring via Measuring Task Similarity. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management* (<conf-loc>, <city>Birmingham</city>, <country>United Kingdom</country>, </conf-loc>) *(CIKM '23)*. Association for Computing Machinery, New York, NY, USA, 4465–4469. https://doi.org/10.1145/3583780.3615149
[50]]BX-DS Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. [n. d.]. *Book-Crossing Dataset*. http://www.informatik.uni-freiburg.de/~cziegler/BX/
[51] Cai-Nicolas Ziegler, Sean M McNee, Joseph A Konstan, and Georg Lausen. 2005. Improving recommendation lists through topic diversification. In *Proceedings of the 14th international conference on World Wide Web*. 22–32.

A Case Studies

In this section, we present three case studies to show how EEMU can generalize to another fundamental recommendation model, support user-wise exact unlearning and perform while employing another meta-learning algorithm.

A.1 Case Study 1: EEMU on another fundamental recommendation model

EEMU is a model-agnostic framework that can be applied to various fundamental recommendation models. To demonstrate this, we extended our approach to another popular recommendation

model, DeepFM [18]. DeepFM combines the strengths of Factorization Machines [34] for efficiently capturing second-order feature interactions and Deep Neural Networks for learning complex, non-linear, high-order feature interactions. We conducted experiments using all four datasets studied in this paper and reported the results in Table 8. The results show that EEMU significantly outperforms all baselines, maintaining the trends observed in the main results of the paper (reported in Table 3). The consistent improvement across datasets highlights the generalizability of EEMU, not only to diverse data distributions but also to complex models like DeepFM. This case study reinforces EEMU's ability to adapt to different fundamental recommendation models while preserving excellent performance, further demonstrating its practical applicability to a wide range of recommendation scenarios.

A.2 Case Study 2: User-wise exact unlearning

Although EEMU is designed for scenarios where deletion requests occur at the interaction level, we believe that a robust machine unlearning framework should also support user-wise deletion requests, allowing users to request the removal of all their data. To asses how EEMU generalizes to such scenarios, we followed [25] by running experiments where 5% and 10% of the users were randomly chosen for deletion to mimic user-wise unlearning requests. The results of the user-wise unlearning are reported in Table 9, where HR@10 results are presented while varying the q% of users deleted at random. The percentages in parentheses represent the degradation for each baseline compared to RETRAIN. We observe that EEMU still outperforms the baselines and that the same conclusions for interaction level unlearning still hold.

A.3 Case Study 3: Reptile

To further demonstrate the effectiveness of meta-learning in our setting, we conducted additional experiments using Reptile [30] as the meta-learning algorithm. Reptile updates the model by performing multiple gradient steps on sampled tasks and taking a first-order approximation of the model update. Our results show that Reptile achieves performance comparable to MAML across all datasets that it outperforms all baselines, as reported in Table 10. This similarity suggests that the core advantage of meta-learning in our setting stems from the ability to quickly adapt to new tasks rather than the specific optimization strategy used.

B Implementation Details

Our model is implemented entirely in PyTorch [33] and all experiments are conducted on a server consisting of eight Nvidia RTX 5000 GPU and an AMD EPYC 7502 32-Core CPU. For a fair comparison we use the same fundamental recommendation model architecture specifications for EEMU and all baselines, and use the GeLU activation function [20] and layernorms between layers. Specifically, we adopt a common tower structure, wherein higher-layer hidden dimensions contain half the number of neurons compared to their lower layers. The Adaptive Moment Estimation (Adam) [23] is chosen as the optimizer. Learning rate and batch size are determined via grid search within the ranges of {0.2, 0.02, 0.002, 0.0002, 0.00002} and {32, 64, 128, 512, 1024}, respectively. The number of shards used for all experiments in this paper is set to 8 shards except for the experiments conducted to produce the results shown in Figure 3, Figure 5(c) and Figure 5(d), where we vary the number of shards to study its effect on efficiency and recommendation performance degradation. Similarly, an adaption set of size 7 is used for all experiments in this paper except for the experiments conducted to produce the results shown in Figure 5(a) and Figure 5(b), where we vary the size of the adaptation set to study its effect on the recommendation performance degradation. For the results in Figure 7(d), we set the number of clusters K to 25.

	MovieLens-1M		BookCrossing		Gowalla		Epinions	
Baseline	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
RETRAIN	0.06970	0.03360	0.02873	0.01340	0.05090	0.02348	0.03082	0.01480
MRA	0.06076 (12%)	0.02967 (11%)	0.01698 (41%)	0.00755 (43%)	0.00918 (81%)	0.00408 (82%)	0.01964 (36%)	0.00899 (39%)
ItemC	0.06192 (11%)	0.02950 (12%)	0.01741 (39%)	0.00825 (38%)	0.01202 (76%)	0.00542 (76%)	0.01855 (39%)	0.00908 (38%)
JS_Div	0.06242 (10%)	0.02988 (11%)	0.01829 (36%)	0.00913 (31%)	0.01218 (76%)	0.00573 (75%)	0.01964 (36%)	0.00953 (35%)
Mean	0.06258 (10%)	0.02966 (11%)	0.01785 (37%)	0.00876 (34%)	0.01235 (75%)	0.00587 (74%)	0.02046 (33%)	0.00947 (36%)
GraphEraser	0.06325 (9%)	0.02972 (11%)	0.01741 (39%)	0.00902 (32%)	0.01218 (76%)	0.00569 (75%)	0.02019 (34%)	0.00946 (36%)
RecEraser	0.05795 (16%)	0.02819 (16%)	0.01959 (31%)	0.00965 (27%)	0.01986 (60%)	0.00897 (61%)	0.01882 (38%)	0.00875 (40%)
UltraRE	0.06341 (9%)	0.02984 (11%)	0.01872 (34%)	0.00920 (31%)	0.01152 (77%)	0.00498 (78%)	0.01855 (39%)	0.00922 (37%)
EEMU	0.06490 **(6%)**	0.03199 **(4%)**	0.03396 **(0%)**	0.01756 **(0%)**	0.05189 **(0%)**	0.02426 **(0%)**	0.02319 **(24%)**	0.01236 **(16%)**

Table 8: Comparison against baselines by HR@10 and NDCG@10 for DeepFM as a fundamental recommendation model.

	MovieLens-1M		BookCrossing		Gowalla		Epinions	
Baseline	$q = 5\%$	$q = 10\%$	$q = 5\%$	$q = 10\%$	$q = 5\%$	$q = 10\%$	$q = 5\%$	$q = 10\%$
RETRAIN	0.07235	0.07114	0.03108	0.03103	0.05371	0.05334	0.03209	0.03163
MRA	0.05877 (19%)	0.05856 (18%)	0.01989 (36%)	0.01980 (36%)	0.01216 (77%)	0.01194 (78%)	0.01929 (40%)	0.01906 (40%)
ItemC	0.05966 (17%)	0.05862 (18%)	0.02466 (21%)	0.02452 (21%)	0.01723 (68%)	0.01710 (68%)	0.01975 (38%)	0.01963 (38%)
JS_Div	0.05985 (17%)	0.05934 (17%)	0.02524 (19%)	0.02492 (20%)	0.01636 (70%)	0.01606 (70%)	0.01879 (41%)	0.01860 (41%)
Mean	0.05950 (18%)	0.05884 (17%)	0.02531 (19%)	0.02487 (20%)	0.01636 (70%)	0.01612 (70%)	0.01945 (39%)	0.01925 (39%)
GraphEraser	0.06074 (16%)	0.06013 (15%)	0.02561 (18%)	0.02520 (19%)	0.01745 (68%)	0.01724 (68%)	0.01899 (41%)	0.01872 (41%)
RecEraser	0.05578 (23%)	0.05539 (22%)	0.01868 (40%)	0.01844 (41%)	0.02170 (60%)	0.02139 (60%)	0.02177 (32%)	0.02153 (32%)
UltraRE	0.05851 (19%)	0.05811 (18%)	0.02458 (21%)	0.02449 (21%)	0.01639 (69%)	0.01620 (70%)	0.01908 (41%)	0.01884 (40%)
EEMU	0.06557 **(9%)**	0.06466 **(9%)**	0.03949 **(0%)**	0.03917 **(0%)**	0.05653 **(0%)**	0.05564 **(0%)**	0.02578 **(20%)**	0.02560 **(20%)**

Table 9: Comparison against baselines by HR@10 after randomly removing $q\%$ of users.

	MovieLens-1M		BookCrossing		Gowalla		Epinions	
Baseline	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
RETRAIN	0.07235	0.03555	0.03135	0.01446	0.05391	0.02531	0.03219	0.01518
MRA	0.05927 (18%)	0.02837 (20%)	0.02003 (36%)	0.00988 (32%)	0.01218 (77%)	0.00609 (76%)	0.01937 (40%)	0.00909 (40%)
ItemC	0.05977 (17%)	0.02934 (17%)	0.02481 (21%)	0.01230 (15%)	0.01736 (68%)	0.00874 (65%)	0.01991 (38%)	0.00904 (40%)
JS_Div	0.06026 (17%)	0.02876 (19%)	0.02525 (19%)	0.01204 (17%)	0.01636 (70%)	0.00756 (70%)	0.01882 (42%)	0.00915 (40%)
Mean	0.05993 (17%)	0.02908 (18%)	0.02537 (19%)	0.01317 (9%)	0.01643 (70%)	0.00761 (70%)	0.01964 (39%)	0.00906 (40%)
GraphEraser	0.06093 (16%)	0.02930 (18%)	0.02569 (18%)	0.01236 (15%)	0.01752 (68%)	0.00841 (67%)	0.01909 (41%)	0.00886 (42%)
RecEraser	0.05613 (22%)	0.02684 (25%)	0.01872 (40%)	0.00871 (40%)	0.02170 (60%)	0.01103 (56%)	0.02182 (32%)	0.00953 (37%)
UltraRE	0.05894 (19%)	0.02863 (19%)	0.02481 (21%)	0.01294 (11%)	0.01652 (70%)	0.00789 (69%)	0.01909 (41%)	0.00896 (41%)
EEMU (Reptile)	0.06581 (9%)	0.03256 (8%)	0.03528 (0%)	0.01644 (0%)	0.05624 (0%)	0.02661 (0%)	0.02557 (21%)	0.01205 (21%)
EEMU (MAML)	0.06589 **(9%)**	0.03299 **(7%)**	0.03962 **(0%)**	0.01985**(0%)**	0.05674 **(0%)**	0.02691 **(0%)**	0.02591 **(20%)**	0.01218 **(20%)**

Table 10: Comparison against baselines by HR@10 and NDCG@10 showing EEMU that employs Reptile as meta-learning algorithm.