SREraser: Efficient Exact Unlearning for Sequential Recommendation

Anonymous authors
Paper under double-blind review

Abstract

Sequential recommender systems ingest sensitive personal histories, necessitating machine unlearning to ensure privacy compliance and maintain system utility. Since retraining from scratch is costly and approximate unlearning cannot certify complete data removal, exact unlearning via data sharding has become the prevailing solution. However, most studies in recommendation unlearning have focused on collaborative filtering, and the application of this paradigm to sequential recommendation remains underexplored. Existing methods face two limitations: partitioning strategies scale poorly with the number of users, and aggregators assign a single scalar weight to each sub-model, ignoring how sub-model reliability varies across representation dimensions. In this paper, we propose SREraser, a framework capable of interaction-level unlearning in sequential recommendation. SREraser introduces two core designs. First, User-Centric Spherical Partitioning (UCSP) ranks shards for each user by cosine similarity and resolves shard capacity constraints locally, reducing partitioning cost by an order of magnitude. Second, Adaptive Dimension Aggregation (ADA) produces one weight per representation dimension of every sub-model, conditioned on how closely that sub-model's response aligns with its shard prototype. Experiments on three real-world datasets show that SREraser matches or outperforms other shard-based unlearning methods in recommendation utility, while maintaining comparable unlearning efficiency.

1 Introduction

Recommender systems alleviate information overload by providing personalised services, and have become increasingly influential in shaping user behaviour and decision-making across e-commerce, media streaming and social platforms (Boka et al., 2024). Within this domain, sequential recommender systems predict a user's next interaction by capturing the temporal structure of their past behaviours, and have been scaled to large industrial deployments (Pancha et al., 2022; Zhai et al., 2024). However, machine learning models have been shown to memorise training data (Hu et al., 2022; Tian et al., 2025), and recommender models also have this problem (Zhang et al., 2021b; Zhu et al., 2023; He et al., 2025). Removing data from trained sequential recommenders is called sequential recommendation unlearning. This operational requirement is driven by two complementary imperatives. First, from a privacy perspective, regulatory frameworks such as the GDPR (2016), CCPA (2018), and PIPL (2021) grant users the "right to be forgotten", allowing users to request the removal of specific interactions (e.g., a sensitive purchase or video view) while keeping their accounts. Second, from a utility perspective, harmful interactions introduced by poisoning attacks (Zhang et al., 2021a; Zhao et al., 2025) or automated scripts seriously degrade recommendation utility and must be removed from trained models once identified. This paper therefore focuses on interaction-level unlearning, where a request targets one or more interactions rather than an entire account.

The most direct response to an unlearning request is to retrain the model from scratch on the remaining data. However, full retraining on large-scale datasets incurs prohibitive computational costs (Li et al., 2026). Exact unlearning addresses this overhead through a shard-based design: splitting the training data into disjoint shards, training an isolated sub-model on each shard, and aggregating the sub-model outputs at inference. The influence of a single sample is therefore limited to one sub-model, restricting the retraining overhead to the affected shard. In recommendation, the random

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

partitioning of SISA (Bourtoule et al., 2021) disrupts collaborative signals and degrades performance, so unlearning frameworks for recommendation partition the data by similarity and aggregate the sub-model outputs with designed weights. However, existing partitioning methods, from RecEraser to its recent extensions to sequential and LLM-based recommendation (Chen et al., 2022a; Xin et al., 2024; Hu et al., 2025), adopt capacity-constrained k-means clustering, incurring a sorting overhead that scales super-linearly with the number of users. In the aggregation phase, existing scalar weighting methods (Chen et al., 2022a;b; Li et al., 2023; Xin et al., 2024; Alshabanah et al., 2025) exhibit limited expressiveness. Assigning a single scalar weight to each sub-model evaluates its output as a whole, failing to capture varying reliability across individual dimensions.

To address these limitations, we propose SREraser, a framework for sequential recommendation that achieves efficient unlearning while maintaining high recommendation utility. It consists of two novel components. UCSP (User-Centric Spherical Partitioning) partitions users by the direction of their sequence representations on the unit sphere: each user takes their most similar shard with remaining capacity, so one ranking per user over the shards replaces the global sort. ADA (Adaptive Dimension Aggregation) weights every dimension of every sub-model representation, a fine granularity that enables selectively leveraging each sub-model's individual strengths. Since SREraser is model-agnostic, we evaluate it with three sequential recommendation backbones: SASRec (Kang & McAuley, 2018), BERT4Rec (Sun et al., 2019) and GRU4Rec (Hidasi et al., 2016) on three real-world datasets, against full retraining, SISA (Bourtoule et al., 2021) and SRU (Xin et al., 2024). SREraser achieves recommendation utility superior or comparable to state-of-the-art unlearning methods while preserving the efficiency of shard-based unlearning. The main contributions of this work are summarised as follows:

- We propose SREraser, a framework for sequential recommendation that supports interaction-level unlearning and achieves both high unlearning efficiency and high recommendation utility.
- We design UCSP, a user-centric partitioning strategy to split the dataset into disjoint shards with superior scalability, and propose an adaptive dimension-wise aggregation method ADA to further improve the recommendation utility of SREraser.
- Empirical evaluations across three datasets and three backbones show that SREraser achieves a better trade-off between unlearning efficiency and recommendation utility than other state-of-the-art shard-based unlearning methods.

2 Related Work

Machine unlearning. Cao & Yang (2015) introduced machine unlearning as the problem of making a trained model forget specified training data. The field is broadly categorised into approximate and exact paradigms by the level of unlearning completeness (Wang et al., 2024; Xu et al., 2024; Nguyen et al., 2025). Approximate methods update trained models either by manipulating their parameters (Guo et al., 2020; Fan et al., 2024) or by fine-tuning (Chundawat et al., 2023; Shen et al., 2024); they are computationally efficient but cannot guarantee complete data removal. Exact methods guarantee completeness by construction. Their dominant realisation in deep models is the data sharding paradigm, first formalised by SISA (Bourtoule et al., 2021), which trains isolated sub-models on disjoint shards and aggregates their predictions, and has been extended to graphs (Chen et al., 2022b), class-wise partitioning (Yan et al., 2022), federated learning (Wen et al., 2026) and large models (Xu et al., 2025; Hu et al., 2025). **Recommendation unlearning.** Approximate methods have been applied to collaborative filtering (Dang et al., 2025) and sequential recommendation (Yang et al., 2025). Among exact methods, the random partitioning of SISA is unsuitable for recommendation, which relies on collaborative signals (Li et al., 2026). RecEraser (Chen et al., 2022a) therefore introduces similarity-based partitioning and an attention-based aggregator that weights the user and item embeddings of each sub-model by a single scalar. UltraRE (Li et al., 2023) adds transport weights to the partitioning and simplifies the attention aggregator to logistic regression, whereas EEMU (Alshabanah et al., 2025) returns to random partitioning and static mean pooling, compensating for the lost collaborative signal with meta-learning. These frameworks target static collaborative filtering; SRU (Xin et al., 2024) extends the RecEraser framework to sequential recommendation, retaining capacity-constrained k-means clustering for partitioning and a scalar-weighted aggregator over the sub-model representations. SREraser adopts the same shard-based exact paradigm but redesigns its two components: UCSP retains partitioning quality at reduced cost, and ADA improves recommendation utility through dimension-wise aggregation.

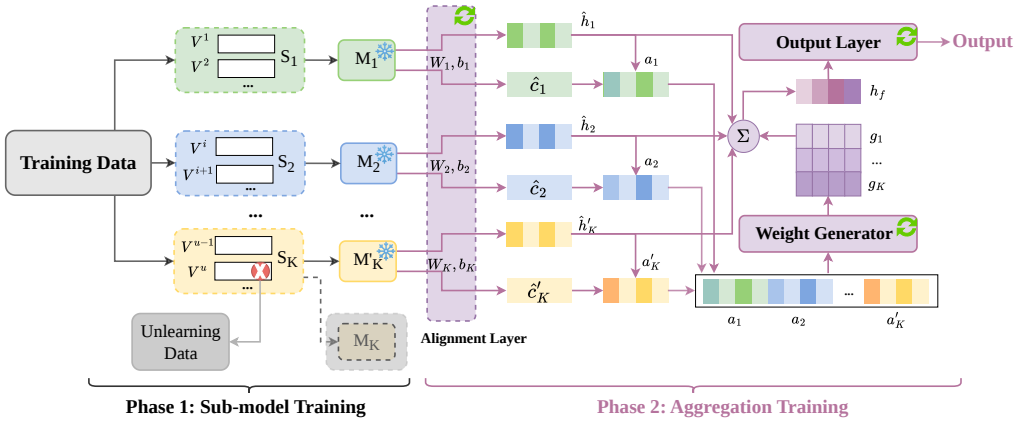

Figure 1: The SREraser framework. UCSP assigns users to disjoint shards for isolated sub-model training, ADA then generates dimension-wise weights to fuse sub-model representations for scoring.

3 PRELIMINARIES

Definition 3.1 (Sequential recommendation)**.** Let $\mathcal{V}$ denote the item set and $\mathcal{D}$ represent the set of training sequences. The sequence $V^u = [v_1^u, \ldots, v_n^u] \in \mathcal{D}$ records the chronologically ordered interactions of user u, where $v_t^u \in \mathcal{V}$ is the item interacted with at time step t, and n denotes the sequence length. Given V^u, the probability of user u interacting with a candidate item v at the next step $n+1$ is defined as

$$p_v^u = p\big(v_{n+1}^u = v \mid V^u, \mathcal{D}\big) = \mathcal{M}(v \mid V^u; \theta_0)\,, \tag{1}$$

where $\mathcal{M}$ denotes the sequential recommendation model, e.g., SASRec (Kang & McAuley, 2018), and $\theta_0 = A(\mathcal{D})$ denotes its parameters learned on $\mathcal{D}$ by the training algorithm A. The model ranks all items in $\mathcal{V}$ by p_v^u to produce the recommendation list.

An unlearning request may be user-level (remove all interactions of a user), item-level (remove all interactions involving an item), or interaction-level (remove a single interaction); this paper focuses on interaction-level unlearning.

Definition 3.2 (Interaction-level exact unlearning)**.** Suppose user u requests the removal of the interaction v_t^u from V^u, which forms the forget set $\mathcal{D}_f = \{v_t^u\}$. The removal yields the sequence $V_{\setminus t}^u = [v_1^u, \ldots, v_{t-1}^u, v_{t+1}^u, \ldots, v_n^u]$ and the remaining training set $\mathcal{D}_r = \big(\mathcal{D} \setminus \{V^u\}\big) \cup \{V_{\setminus t}^u\}$, on which retraining from scratch gives the parameters $\theta^* = A(\mathcal{D}_r)$. Following Li et al. (2026), the goal of interaction-level exact unlearning is to design an unlearning algorithm $U(\cdot)$ such that

$$U(A, \mathcal{D}_f, \mathcal{D}_r, \theta_0) = A(\mathcal{D}_r), \tag{2}$$

where the equality holds in distribution. We write $\theta_{\text{unl}} = U(A, \mathcal{D}_f, \mathcal{D}_r, \theta_0)$ for the unlearned parameters.

Unlearning for sequential recommendation inherits three main objectives established in prior work (Chen et al., 2022a; Li et al., 2023; Alshabanah et al., 2025): **unlearning completeness**, fully removing the influence of $\mathcal{D}_f$; **unlearning efficiency**, obtaining θ_{unl} at lower cost than retraining $A(\mathcal{D}_r)$ from scratch; and **recommendation utility**, keeping the performance of $\mathcal{M}(\cdot\,; \theta_{\text{unl}})$ comparable to that of $\mathcal{M}(\cdot\,; \theta^*)$. Following the shard-based exact unlearning design, which attains completeness at the algorithmic level, this paper advances unlearning efficiency and recommendation utility. Table 4 in Appendix A summarises the notation used throughout the paper.

4 THE SRERASER FRAMEWORK

4.1 OVERVIEW

An overview of the proposed SREraser is illustrated in Figure 1. The architecture decouples the learning process into two phases. In the first phase, the training set $\mathcal{D}$ is divided into K disjoint

Algorithm 1 UCSP: User-Centric Spherical Partitioning algorithm

Require: Normalised user representations $\bar{\boldsymbol{E}} \in \mathbb{R}^{N\times d}$, number of shards K, maximum capacity $C_{\max}$, iteraction budget T
Ensure: Shards $\mathcal{S} = \{S_1, \ldots, S_K\}$
Initialise unit-norm centroids $\boldsymbol{Q} \leftarrow \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K\}$ and previous partition $\mathcal{P} \leftarrow \varnothing$
for $t = 1, \ldots, T$ **do**
  Compute affinity matrix $\boldsymbol{A} \leftarrow \bar{\boldsymbol{E}}\boldsymbol{Q}^\top$
  Initialise empty shards $S_k \leftarrow \varnothing$ for all $k \in \{1, \ldots, K\}$
  for each user $u \in \mathrm{RandomPermutation}(\mathcal{U})$ **do**
    $k_u \leftarrow \arg\max_{k:|S_k|<C_{\max}} \boldsymbol{A}_{u,k}$
    $S_{k_u} \leftarrow S_{k_u} \cup \{u\}$
  end for
  $\mathcal{S} \leftarrow \{S_1, \ldots, S_K\}$
  if $\mathcal{S} = \mathcal{P}$ **then break**
  end if
  $\mathcal{P} \leftarrow \mathcal{S}$
  Update centroids $\boldsymbol{\mu}_k \leftarrow \dfrac{\sum_{u\in S_k} \bar{\boldsymbol{e}_u}}{\|\sum_{u\in S_k} \bar{\boldsymbol{e}_u}\|_2}$ for all $k \in \{1, \ldots, K\}$
end for
return $\mathcal{S}$

shards $S_1, \ldots, S_K$, and one sub-model M_k is trained on each shard independently. This isolation allows sub-models to be trained in parallel, thereby accelerating the overall training process. In the second phase, all sub-models are frozen, and their outputs serve as training inputs to optimise the aggregator. During inference, an input user sequence is passed to every sub-model, each encoding the sequence into a hidden representation. The aggregator then computes fine-grained weights based on the input to combine these sub-model representations, producing the final recommendation results.

This architecture also determines how SREraser serves an unlearning request. Since UCSP assigns each user sequence to a single shard, a revoked interaction affects only the training data of the corresponding sub-model M_k. Therefore, retraining M_k on the shard without the revoked interaction and retraining the aggregator on $\mathcal{D}_r$ from scratch ensure that every parameter of SREraser is trained only on $\mathcal{D}_r$. The computational cost is restricted to the affected sub-model and the aggregator, offering higher efficiency than full model retraining.

4.2 UCSP: User-Centric Spherical Partitioning

Existing frameworks combine similarity-based grouping with balanced partitioning to protect recommendation accuracy and unlearning efficiency (Chen et al., 2022a;b; Xin et al., 2024). However, both properties are achieved by capacity-constrained k-means clustering, which introduces substantial computational costs. Inspired by spherical k-means clustering on high-dimensional document vectors (Dhillon & Modha, 2001), UCSP replaces the Euclidean metric with cosine similarity on the unit sphere and, more importantly, resolves the capacity constraint from each user's perspective, reducing sorting overhead from $O(NK \log NK)$ to $O(NK \log K)$ per iteration. This enables UCSP to retain both desirable properties at a fraction of the existing partitioning cost.

First, we deploy a sequential recommendation model (e.g., SASRec), pre-trained on the full dataset $\mathcal{D}$, as the encoder $f(\cdot)$ to map each user interaction sequence V^u into a representation vector $\boldsymbol{e}_u = f(V^u) \in \mathbb{R}^d$, where d is the hidden dimension. To optimise the computation, each representation vector is ℓ_2-normalised onto the unit sphere as $\bar{\boldsymbol{e}_u} = \boldsymbol{e}_u / \|\boldsymbol{e}_u\|_2$. Normalisation simplifies similarity computation by reducing pairwise cosine similarity to an inner product. Consequently, computing affinities between all N users and K centroids reduces to a single matrix multiplication $\boldsymbol{A} = \bar{\boldsymbol{E}}\boldsymbol{Q}^\top$, where $\bar{\boldsymbol{E}} \in \mathbb{R}^{N\times d}$ denotes the user matrix collecting all $\{\bar{\boldsymbol{e}_u}\}_{u\in\mathcal{U}}$ and $\boldsymbol{Q} \in \mathbb{R}^{K\times d}$ denotes the centroid matrix.

To enforce balance across shards, the maximum capacity is defined as $C_{\max} = \lfloor N/K \rfloor + 5$. Rather than performing global sorting over all user-shard pairs, UCSP adopts a user-centric assignment strategy. In each iteration t, users are processed according to a random permutation. Each user u

is assigned to their highest-affinity shard with remaining capacity, $k_u = \arg\max_{k\,:\,|S_k|<C_{\max}} \boldsymbol{A}_{u,k}$. This local search requires $O(K \log K)$ operations per user. After assigning all users, each centroid is updated as $\boldsymbol{\mu}_k \leftarrow \sum_{u\in S_k} \bar{\boldsymbol{e}_u} / \| \sum_{u\in S_k} \bar{\boldsymbol{e}_u} \|_2$. The algorithm terminates when the shard assignments stabilise between consecutive iterations or until exhausting the iteration budget T, as summarised in Algorithm 1.

4.3 ADA: Adaptive Dimension Aggregation

By utilising UCSP to partition data into coherent behavioural groups, each sub-model captures distinct aspects of preferences and specialises in its specific representation space. To fully utilise these specialised representations for higher performance, aggregation must be conducted at a fine-grained level. To this end, ADA produces a weight for every *dimension* of every sub-model.

Before detailing the aggregation mechanism, we first define its two primary inputs, which capture both the individual input features and the global shard behaviour. Let $\boldsymbol{h}_k \in \mathbb{R}^d$ denote the hidden representation emitted by the sub-model M_k for an input sequence. We define the prototype of shard k, denoted as $\boldsymbol{c}_k \in \mathbb{R}^d$, as the mean hidden representation produced by M_k across all users within its training shard.

Alignment Layer. Since independently trained sub-models produce hidden representations in distinct representation spaces, we apply a shared affine mapping to each sub-model's hidden representation and prototype:

$$\hat{\boldsymbol{h}}_k = \boldsymbol{W}_k \boldsymbol{h}_k + \boldsymbol{b}_k, \qquad \hat{\boldsymbol{c}}_k = \boldsymbol{W}_k \boldsymbol{c}_k + \boldsymbol{b}_k, \qquad k = 1, \ldots, K, \tag{3}$$

where $\boldsymbol{W}_k \in \mathbb{R}^{d\times d}$ and $\boldsymbol{b}_k \in \mathbb{R}^d$ are learnable parameters.

Weight Generator. The weight generator uses the dimension-wise agreement between each sub-model's aligned representation and its aligned prototype to generate input-adaptive weights for dimension-wise aggregation. For sub-model k, an agreement signal is obtained as

$$\boldsymbol{a}_k = \hat{\boldsymbol{h}}_k \odot \hat{\boldsymbol{c}}_k \in \mathbb{R}^d. \tag{4}$$

The element-wise product is large on the dimensions along which the input drives M_k in the same direction as its own prototype. The agreement signals from all sub-models are concatenated into a single vector

$$\boldsymbol{z} = [\boldsymbol{a}_1; \ldots; \boldsymbol{a}_K] \in \mathbb{R}^{Kd}, \tag{5}$$

which is passed through the weight generator to produce the weight matrix:

$$\boldsymbol{G} = \boldsymbol{W}^{(2)} \mathrm{ReLU}\big(\boldsymbol{W}^{(1)} \boldsymbol{z} + \boldsymbol{b}^{(1)}\big) + \boldsymbol{b}^{(2)} \in \mathbb{R}^{Kd}, \tag{6}$$

where $\boldsymbol{W}^{(1)} \in \mathbb{R}^{d\times Kd}$ and $\boldsymbol{W}^{(2)} \in \mathbb{R}^{Kd\times d}$ are learnable weight matrices, with $\boldsymbol{b}^{(1)} \in \mathbb{R}^d$ and $\boldsymbol{b}^{(2)} \in \mathbb{R}^{Kd}$ as bias terms. This output is reshaped into a $K \times d$ matrix, so that $G_{k,j}$ is the raw weight of sub-model k on dimension j. Because the weight generator takes the concatenated vector $\boldsymbol{z}$ as input, the generator jointly captures the relative contributions of all sub-models. To obtain valid aggregation coefficients from this raw matrix, we apply a softmax function across all K sub-models independently for each feature dimension. For a given dimension $j \in \{1, \ldots, d\}$, the normalised weight $g_{k,j}$ assigned to sub-model k is computed as:

$$g_{k,j} = \frac{\exp(G_{k,j})}{\sum_{k'=1}^{K} \exp(G_{k',j})}. \tag{7}$$

Let $\boldsymbol{g}_k \in \mathbb{R}^d$ denote the resulting normalised weight vector for sub-model k.

Dimension-wise Aggregation and Output. Given the aligned representations and their corresponding fine-grained weights, the final fused representation $\boldsymbol{h}_f$ is computed as the dimension-wise combination across all sub-models:

$$\boldsymbol{h}_f = \sum_{k=1}^{K} \boldsymbol{g}_k \odot \hat{\boldsymbol{h}}_k. \tag{8}$$

As shown in Figure 1, the fused representation $\boldsymbol{h}_f$ is then passed to a two-layer feed-forward network with ReLU activation that maps $\boldsymbol{h}_f$ into the full candidate set to produce the final recommendation scores.

Table 1: Statistics of datasets.

Dataset	#User	#Item	#Interaction	Density	Avg. length
Amazon Games	24303	10672	231780	0.09%	9.5
Amazon Beauty	22363	12101	198502	0.07%	8.9
MovieLens-1M	6040	3416	999611	4.84%	165.5

Table 2: Performance comparison between our SREraser and other efficient unlearning methods (SISA, SRU) after serving unlearning requests affecting 10% of the users. All values are reported as the mean over five seeds, scaled by 10^{-3}. The best results are bolded.

		SASRec				BERT4Rec				GRU4Rec			
		Retrain	SISA	SRU	SREraser	Retrain	SISA	SRU	SREraser	Retrain	SISA	SRU	SREraser
Games	NDCG@5↑	59.8	36.0	36.4	**41.7**	46.6	36.3	39.0	**40.9**	51.0	24.9	31.0	**32.2**
	NDCG@10↑	74.1	43.0	45.5	**52.1**	60.5	45.9	50.6	**52.8**	62.6	29.6	38.4	**40.6**
	NDCG@20↑	89.0	50.5	55.5	**63.8**	76.3	56.2	63.2	**66.3**	74.2	34.7	46.4	**49.7**
	Recall@5↑	85.9	51.4	54.3	**62.1**	72.3	55.1	59.7	**63.0**	74.5	35.2	45.6	**48.2**
	Recall@10↑	130.7	73.0	82.7	**94.4**	115.7	85.1	96.0	**100.1**	110.5	50.0	68.4	**74.4**
	Recall@20↑	189.8	103.0	122.3	**141.1**	178.2	126.1	146.2	**154.1**	156.4	70.3	100.1	**110.6**
Beauty	NDCG@5↑	69.6	47.4	48.8	**51.9**	54.8	42.4	43.9	**46.3**	63.9	43.5	43.3	**46.4**
	NDCG@10↑	80.9	54.9	56.5	**61.0**	66.6	50.9	52.9	**55.7**	73.5	49.3	50.4	**54.5**
	NDCG@20↑	91.9	62.3	64.3	**69.9**	78.0	59.6	62.5	**65.8**	82.9	55.0	57.9	**62.5**
	Recall@5↑	96.9	64.4	66.3	**71.8**	80.6	61.6	62.9	**66.8**	87.3	57.9	59.5	**63.9**
	Recall@10↑	132.1	87.5	90.1	**100.2**	117.2	88.0	91.0	**95.9**	116.9	76.0	81.6	**89.1**
	Recall@20↑	175.7	116.9	121.4	**135.3**	162.5	122.4	129.2	**136.2**	154.1	98.4	111.4	**121.0**
MovieLens-1M	NDCG@5↑	112.3	107.3	105.1	**108.2**	92.8	89.9	**90.7**	90.6	101.6	95.7	93.7	**96.3**
	NDCG@10↑	139.9	131.8	132.1	**135.8**	120.5	115.9	**118.1**	117.9	126.9	118.9	119.1	**122.1**
	NDCG@20↑	168.6	157.7	160.5	**164.2**	150.6	144.9	148.1	**148.4**	153.8	144.0	146.6	**149.8**
	Recall@5↑	167.7	159.1	158.9	**163.3**	144.4	138.6	**140.5**	140.4	152.4	142.2	142.4	**146.5**
	Recall@10↑	253.5	235.4	242.9	**249.1**	230.7	219.7	**225.8**	225.7	231.3	214.3	221.6	**226.6**
	Recall@20↑	367.4	338.0	355.6	**361.8**	350.2	335.1	345.2	**346.8**	338.1	314.0	331.0	**336.9**

Training the Aggregator. ADA is trained on the full training set to ensure the aggregation parameters capture the overall data distribution. The training objective combines the recommendation loss with ℓ_2 regularisation on all aggregator parameters to mitigate overfitting:

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \lambda \|\Theta_{\text{agg}}\|_2^2, \tag{9}$$

where λ controls the regularisation strength and Θ_{agg} denotes the collective parameters of the alignment layers, the weight generator and the output layer. All sub-models remain frozen throughout the aggregation training phase, restricting gradient updates exclusively to Θ_{agg}. Since each frozen sub-model acts as a deterministic feature extractor, $\{\boldsymbol{h}_k\}$ remains constant across epochs and can be pre-computed and cached prior to aggregation training. Subsequent epochs read directly from the cache instead of executing K sub-model forward passes per batch, which accelerates aggregation training. When retraining the aggregator after unlearning requests, the cache requires only a partial refresh targeting stale entries: representations produced by the retrained sub-model and the affected users' representations from the remaining sub-models. Appendix B.2 provides a detailed analysis of the cache refresh mechanism alongside ablation results for unlearning time when caching is disabled.

5 Experiments

5.1 Experimental Setup

Datasets and models. We evaluate SREraser on three public datasets that differ in scale and sparsity (Table 1): Amazon Games, Amazon Beauty (McAuley et al., 2015) and MovieLens-1M (Harper & Konstan, 2015). We adopt three sequential recommendation backbones covering self-attention and recurrent architectures: SASRec (Kang & McAuley, 2018), BERT4Rec (Sun et al., 2019) and GRU4Rec (Hidasi et al., 2016). The number of shards is $K = 8$ by default. **Baselines.** We compare SREraser against Retrain, which retrains from scratch on the remaining data, and two shard-based exact unlearning methods, SISA (Bourtoule et al., 2021) and SRU (Xin et al., 2024). **Unlearning**

Table 3: Unlearning time for a single unlearning request across Retrain, SREraser and SRU.

Time (s) ↓		Games			Beauty			MovieLens-1M		
		SASRec	BERT4Rec	GRU4Rec	SASRec	BERT4Rec	GRU4Rec	SASRec	BERT4Rec	GRU4Rec
Retrain		474.64	385.44	123.23	444.9	406.82	115.01	4655.33	6237.41	1536.64
SRU	Sub-model	60.55	54.22	23.29	48.55	36.19	22.75	569.31	671.69	139.38
	Aggregation	18.55	19.05	18.18	17.07	15.72	15.73	136.12	117.58	101.73
	Total	79.1	73.27	41.47	65.62	51.91	38.48	705.43	789.27	241.11
SREraser	Sub-model	60.55	53.37	22.96	46.27	38.26	23	570.95	641.91	141.18
	Aggregation	18.53	18.96	17.8	17.39	15.93	15.63	132.67	113.55	96.82
	Total	79.08	72.33	40.76	63.66	54.19	38.63	703.62	755.46	238

setting and metrics. Following SRU, we define the *unlearning ratio* as the percentage of users issuing an unlearning request, where each request removes one randomly chosen interaction; the ratio is 10% by default. We report Recall@$\{5, 10, 20\}$ and NDCG@$\{5, 10, 20\}$, with the target item ranked against the full candidate set. Appendix A details data preprocessing and training settings.

5.2 Recommendation Performance

We evaluate recommendation performance of all unlearning methods after serving unlearning requests at the default 10% ratio. Table 2 shows that Retrain achieves the best performance across all configurations. We analyse its full-dataset computational bottleneck in §5.3. Among the three shard-based methods, SREraser achieves the strongest recommendation performance overall, achieving the highest Recall@20 in all nine combinations of datasets and backbones, with a paired t-test yielding $p < 0.05$ against SRU and $p < 0.001$ against SISA. Compared with SRU, SREraser improves Recall@20 on the Games dataset by 15.4%, 10.5% and 5.4% under SASRec, GRU4Rec and BERT4Rec respectively; the corresponding improvements over SISA are 37.0%, 57.3% and 22.2%. We attribute this gain to the adaptive dimension-wise aggregation design, which the ADA ablation in §5.4 validates in detail. For completeness, we provide the corresponding pre-unlearning results in Table 5 of Appendix B.1.

Beyond the aggregation design, the effective information available to each sub-model is also critical for shard-based unlearning performance. This information is determined by intrinsic dataset density (Table 1) and the data volume per shard. Denser datasets like MovieLens-1M equip each shard with richer behavioural signals, mitigating partition-induced information loss and allowing SREraser to closely match Retrain. On MovieLens-1M, SREraser recovers over 98% of Retrain performance across all three backbones. The second factor, the data volume per sub-model, is determined by the number of shards K, introducing an inherent trade-off between unlearning efficiency and recommendation performance that the shard-number analysis in §5.4 verifies.

Resilience to Data Removal. To evaluate the robustness of SREraser against extensive data removal, we vary the unlearning ratio from 0 to 50% on Games under SASRec, averaging results across ten random seeds to mitigate the impact of randomness in user selection and interaction selection. Figure 5 shows that after erasing interactions from half of the users, SREraser retains 95.7% of its Recall@20, while SRU retains 94.5%. This minimal performance degradation reflects SREraser's resilience to data removal.

5.3 Unlearning Efficiency

The unlearning cost is reported per request, defined as the removal of a single interaction for a specific user. Since SREraser and SRU assign each user to an exclusive shard, an unlearning request invalidates exactly one sub-model and requires retraining only the affected sub-model and the aggregator. Given that the specific shard affected by an unlearning request varies, we report the sub-model retraining time as the mean retraining time across all K sub-models. To accelerate aggregator training, SREraser pre-computes and caches the hidden representations generated by the frozen sub-models (§4.3). For a fair comparison, we also apply this caching strategy to SRU. Table 3 summarises the unlearning cost for SREraser, SRU and Retrain, with all experiments evaluated under an identical environment configuration (detailed in Appendix A.3). As shown in Table 3, SREraser and SRU incur comparable unlearning time across all configurations. However, SREraser

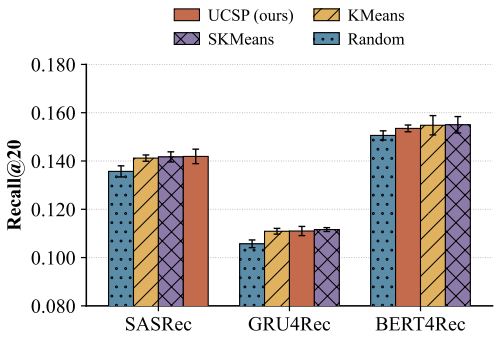

Figure 2: Performance comparison of different data partition methods on Games: Recall@20, mean over five seeds with ± 1 std error bars.

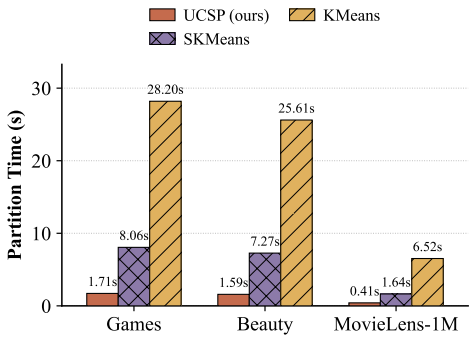

Figure 3: Partitioning time of UCSP, SKMeans and KMeans across three datasets with $K = 8$, mean over five seeds.

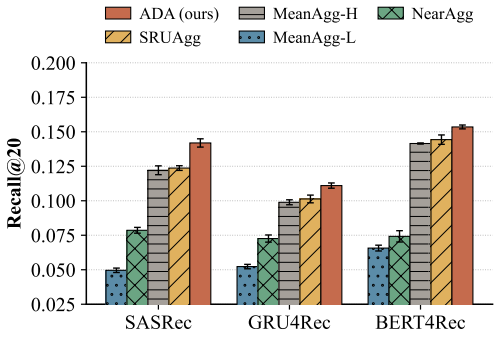

Figure 4: Performance comparison of different aggregation methods on Games: Recall@20, mean over five seeds with ± 1 std error bars.

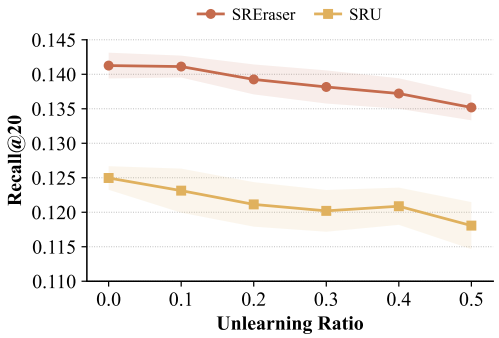

Figure 5: SREraser and SRU with various unlearning ratios on Games/SASRec, mean over ten seeds with ± 1 std bands.

achieves a 3.0–8.3× speedup over Retrain, with the largest gain on MovieLens-1M (6.5–8.3×), where full retraining is most computationally expensive. SREraser therefore achieves a superior performance-efficiency trade-off to SRU.

5.4 Ablation Study

Effect of UCSP. To isolate the effectiveness of the partitioning strategies, we hold the aggregator fixed at ADA and vary the partitioner. Random shards users uniformly at random. KMeans uses Euclidean distance with global sorting for capacity-constrained assignment, whereas SKMeans replaces Euclidean distance with cosine similarity (Dhillon & Modha, 2001) under the same capacity constraint. UCSP retains cosine similarity but performs capacity-aware assignment from each user's perspective. Figure 2 illustrates the performance impact of different partitioning strategies. The three clustering strategies outperform random partitioning for all three backbones on Recall@20. Similarity-based clustering preserves essential collaborative information within each shard, which inherently maximises sub-model diversity. However, the three clustering strategies perform comparably, with their error bars overlapping on every backbone (details in Appendix Table 7). Recommendation utility therefore does not distinguish them, and partitioning cost becomes the deciding criterion. Figure 3 shows that on Games, UCSP partitioning takes 1.71 s, compared with 8.06 s for SKMeans and 28.20 s for KMeans. UCSP thus achieves lower partitioning overhead without sacrificing recommendation utility. Appendix B.3 shows that this cost advantage holds across shard numbers from 4 to 20.

Effect of ADA. To evaluate ADA, we fix the partitioner to UCSP and compare against four baselines. Specifically, NearAgg routes inputs to the nearest sub-model via prototype similarity, whereas SRUAgg aggregates aligned hidden representations using learned per-shard scalar weights. MeanAgg-L

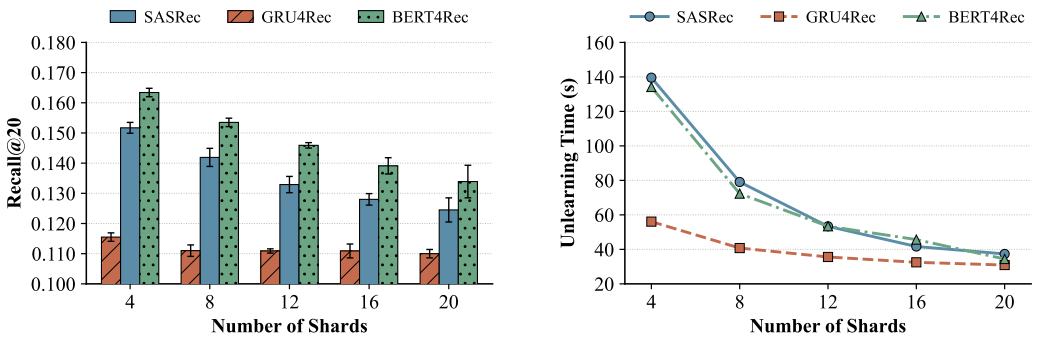

Figure 6: Impact of the shard number on Games. Left: recommendation performance measured by Recall@20, averaged over five seeds with ± 1 std. Right: unlearning efficiency.

and MeanAgg-H both apply a uniform weight of $1/K$, fusing at the logit level and the aligned hidden representation level, respectively. Figure 4 demonstrates that ADA consistently outperforms all baselines. Taking SASRec as an example, ADA achieves a Recall@20 of 0.1419, compared to 0.1237, 0.1221, 0.0786, and 0.0496 for SRUAgg, MeanAgg-H, NearAgg and MeanAgg-L, respectively. The massive improvement of MeanAgg-H over MeanAgg-L and NearAgg affirms the efficacy of the shared architectural foundation. By averaging representations mapped into a common space via the alignment layer and decoding them through the output layer trained on the full dataset, MeanAgg-H preserves representational capacity and avoids severe signal degradation. More importantly, ADA demonstrates the necessity of fine-grained aggregation weighting within representation space. While learned scalar weights (SRUAgg) offer only marginal gains over uniform averaging (MeanAgg-H), ADA achieves a 16.2% relative improvement. By performing dimension-wise aggregation over the hidden representations of specialised sub-model experts, ADA successfully exploits distinct preference patterns at a much finer granularity, fusing complementary signals to achieve superior recommendation performance. Detailed metrics are provided in Appendix Table 8.

Impact of the Shard Number. To investigate the impact of the shard number K, we vary K from 4 to 20 on the Games dataset. Figure 6 (right) demonstrates that unlearning time drops consistently across all three backbones as K increases. Because the retraining cost of the aggregator varies marginally (detailed in Appendix Table 9), this total unlearning time is primarily dominated by the sub-model retraining overhead. Increasing K reduces the data volume per shard and therefore lowers the sub-model retraining cost. Figure 6 (left) shows that recommendation performance decreases as K increases. Finer partitioning reduces the behavioural information available to each sub-model and causes greater loss of collaborative signals, both of which degrade final recommendation performance. The shard number therefore balances unlearning efficiency against recommendation performance.

6 Conclusion

This paper addresses interaction-level unlearning for sequential recommendation, where existing methods suffer from the computational inefficiency of similarity-based balanced partitioning and the limited expressive power of scalar aggregation strategies. To overcome these challenges, we propose SREraser, a framework consisting of two core components. First, UCSP adopts user-centric ranking based on normalised cosine affinity, achieving comparable recommendation performance at a fraction of the partitioning cost. Second, ADA produces adaptive dimension-wise weights that leverage distinct preference patterns from specialised sub-model experts, thereby achieving superior recommendation performance. SREraser achieves the highest Recall@20 among shard-based methods across all evaluated datasets and backbones, at an unlearning cost comparable to SRU. Ablation studies attribute this gain to dimension-wise weighting, demonstrate that UCSP lowers partitioning cost without a utility penalty, and confirm that performance remains robust when interactions are revoked from half of the users.

AI USE STATEMENT

During the preparation of this work, the authors used Claude in order to improve language. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

All datasets used in the experiments are publicly available. Implementation details including hyperparameters, random seeds and hardware specifications are provided in Appendix A. The source code is available at https://anonymous.4open.science/r/SREraser-C43F.

REFERENCES

Abdulla Alshabanah, Keshav Balasubramanian, and Murali Annavaram. Meta-learn to unlearn: Enhanced exact machine unlearning in recommendation systems with meta-learning. *Proceedings on Privacy Enhancing Technologies*, 2025(4):696–711, 2025. 2, 3

David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027–1035, 2007. 14

Tesfaye Fenta Boka, Zhendong Niu, and Rama Bastola Neupane. A survey of sequential recommendation systems: Techniques, evaluation, and future directions. *Information Systems*, 125:102427, 2024. 1

Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 141–159, 2021. 2, 6

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 463–480, 2015. 2

Chong Chen, Fei Sun, Min Zhang, and Bolin Ding. Recommendation unlearning. In *Proceedings of the ACM Web Conference*, pp. 2768–2777, 2022a. 2, 3, 4

Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph unlearning. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 499–513, 2022b. 2, 4

Vikram S Chundawat, Ayush K Tarun, Murari Mandal, and Mohan Kankanhalli. Can bad teaching induce forgetting? Unlearning in deep networks using an incompetent teacher. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 7210–7217, 2023. 2

Yizhou Dang, Yuting Liu, Enneng Yang, Guibing Guo, Linying Jiang, Jianzhe Zhao, and Xingwei Wang. Efficient and adaptive recommendation unlearning: A guided filtering framework to erase outdated preferences. *ACM Transactions on Information Systems*, 43(2):49, 2025. 2

Hangyu Deng and Jinglu Hu. SSE4Rec: Sequential recommendation with subsequence extraction. *Knowledge-Based Systems*, 285:111364, 2024. 13

Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, 2001. 4, 8

European Union. General data protection regulation (GDPR). https://gdpr-info.eu/, 2016. 1

Chongyu Fan, Jiancheng Liu, Yihua Zhang, Eric Wong, Dennis Wei, and Sijia Liu. SalUn: Empowering machine unlearning via gradient-based weight saliency in both image classification and generation. In *Proceedings of the International Conference on Learning Representations*, volume 2024, pp. 53643–53673, 2024. 2

Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. Certified data removal from machine learning models. In *Proceedings of the International Conference on Machine Learning*, pp. 3832–3842, 2020. 2

F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):19, 2015. 6

Jiajie He, Xintong Chen, Xinyang Fang, Min-Chun Chen, Yuechun Gu, and Keke Chen. Membership inference attacks on recommender system: A survey. *arXiv preprint arXiv:2509.11080*, 2025. 1

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *Proceedings of the International Conference on Learning Representations*, 2016. 2, 6

Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, Philip S. Yu, and Xuyun Zhang. Membership inference attacks on machine learning: A survey. *ACM Computing Surveys*, 54(11s):235, 2022. 1

Zhiyu Hu, Yang Zhang, Minghao Xiao, Wenjie Wang, Fuli Feng, and Xiangnan He. Exact and efficient unlearning for large language model-based recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 37(10):5866–5877, 2025. 2

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *Proceedings of the IEEE International Conference on Data Mining*, pp. 197–206, 2018. 2, 3, 6, 13

Yuyuan Li, Chaochao Chen, Yizhao Zhang, Weiming Liu, Lingjuan Lyu, Xiaolin Zheng, Dan Meng, and Jun Wang. UltraRE: Enhancing RecEraser for recommendation unlearning via error decomposition. In *Advances in Neural Information Processing Systems*, volume 36, pp. 12611–12625, 2023. 2, 3

Yuyuan Li, Xiaohua Feng, Chaochao Chen, and Qiang Yang. A survey on recommendation unlearning: Fundamentals, taxonomy, evaluation, and open questions. *IEEE Transactions on Knowledge and Data Engineering*, 38(2):781–799, 2026. 1, 2, 3

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52, 2015. 6

Thanh Tam Nguyen, Thanh Trung Huynh, Zhao Ren, Phi Le Nguyen, Alan Wee-Chung Liew, Hongzhi Yin, and Quoc Viet Hung Nguyen. A survey of machine unlearning. *ACM Transactions on Intelligent Systems and Technology*, 16(5):108, 2025. 2

Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. PinnerFormer: Sequence modeling for user representation at Pinterest. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3702–3712, 2022. 1

Shaofei Shen, Chenhao Zhang, Yawen Zhao, Alina Bialkowski, Weitong Chen, and Miao Xu. Label-agnostic forgetting: A supervision-free unlearning in deep models. In *Proceedings of the International Conference on Learning Representations*, volume 2024, pp. 54217–54235, 2024. 2

Standing Committee of the National People's Congress. Personal information protection law of the People's Republic of China (PIPL). https://en.spp.gov.cn/2021-12/29/c_948419.htm, 2021. 1

State of California Department of Justice. California consumer privacy act (CCPA). https://oag.ca.gov/privacy/ccpa, 2018. 1

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pp. 1441–1450, 2019. 2, 6, 13

Hanling Tian, Yuhang Liu, Mingzhen He, Zhengbao He, Zhehao Huang, Ruikai Yang, and Xiaolin Huang. Simulating training dynamics to reconstruct training data from deep neural networks. In *Proceedings of the International Conference on Learning Representations*, volume 2025, pp. 58968–58994, 2025. 1

Weiqi Wang, Zhiyi Tian, Chenhan Zhang, and Shui Yu. Machine unlearning: A comprehensive survey. *arXiv preprint arXiv:2405.07406*, 2024. 2

Siyuan Wen, Meng Zhang, Yang Yang, and Ningning Ding. FedShard: Federated unlearning with efficiency fairness and performance fairness. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 26841–26848, 2026. 2

Xin Xin, Liu Yang, Ziqi Zhao, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. On the effectiveness of unlearning in session-based recommendation. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pp. 855–863, 2024. 2, 4, 6, 14

Jie Xu, Zihan Wu, Cong Wang, and Xiaohua Jia. Machine unlearning: Solutions and challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 8(3):2150–2168, 2024. 2

Jie Xu, Zihan Wu, Cong Wang, and Xiaohua Jia. LMEraser: Large model unlearning via adaptive prompt tuning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 2026–2034, 2025. 2

Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. ARCANE: An efficient architecture for exact machine unlearning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 4006–4013, 2022. 2

Liu Yang, Zhaochun Ren, Ziqi Zhao, Pengjie Ren, Zhumin Chen, Xinyi Li, Zhiming Peng, Daiting Shi, Maarten de Rijke, and Xin Xin. Curriculum approximate unlearning for session-based recommendation. *arXiv preprint arXiv:2508.15263*, 2025. 2, 13

Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, Yinghai Lu, and Yu Shi. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Proceedings of the International Conference on Machine Learning*, pp. 58484–58509, 2024. 1

Hengtong Zhang, Changxin Tian, Yaliang Li, Lu Su, Nan Yang, Wayne Xin Zhao, and Jing Gao. Data poisoning attack against recommender system using incomplete and perturbed data. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2154–2164, 2021a. 1

Minxing Zhang, Zhaochun Ren, Zihan Wang, Pengjie Ren, Zhumin Chen, Pengfei Hu, and Yang Zhang. Membership inference attacks against recommender systems. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pp. 864–879, 2021b. 1

Yuchuan Zhao, Tong Chen, Junliang Yu, Kai Zheng, Lizhen Cui, and Hongzhi Yin. Diversity-aware dual-promotion poisoning attack on sequential recommendation. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1634–1644, 2025. 1

Zhihao Zhu, Chenwang Wu, Rui Fan, Defu Lian, and Enhong Chen. Membership inference attacks against sequential recommender systems. In *Proceedings of the ACM Web Conference*, pp. 1208–1219, 2023. 1

Table 4: Summary of symbols and notations.

Symbol	Description
$\mathcal{U}$	The set of users
$\mathcal{V}$	The set of items
N	The number of users
K	The number of shards
$\mathcal{D}$	The set of training sequences
V^u	The interaction sequence of user u
v_t^u	The item that user u interacts with at time step t
$\mathcal{M}$	The sequential recommendation model
A	The training algorithm
θ_0	The parameters learned on $\mathcal{D}$
$\mathcal{D}_f$	The forget set
$V_{\setminus t}^u$	The sequence of user u after removing v_t^u
$\mathcal{D}_r$	The remaining training set after removing $\mathcal{D}_f$
θ^*	The parameters retrained from scratch on $\mathcal{D}_r$
$U(\cdot)$	The unlearning algorithm
θ_{unl}	The unlearned parameters
S_k	Shard k
M_k	The sub-model trained on S_k
$C_{\max}$	The maximum capacity on the size of a shard
$\boldsymbol{e}_u$	The pre-trained representation of user u
$\bar{\boldsymbol{e}}_u$	The unit-norm counterpart of $\boldsymbol{e}_u$
$\boldsymbol{\mu}_k$	The centroid of cluster k
$\boldsymbol{h}_k$	The hidden representation emitted by the sub-model M_k
$\boldsymbol{c}_k$	The prototype of S_k
$\hat{\boldsymbol{h}}_k$	The aligned counterpart of $\boldsymbol{h}_k$
$\hat{\boldsymbol{c}}_k$	The aligned counterpart of $\boldsymbol{c}_k$
$\boldsymbol{a}_k$	The dimension-wise agreement signal between $\hat{\boldsymbol{h}}_k$ and $\hat{\boldsymbol{c}}_k$
$\boldsymbol{g}_k$	The dimension-wise aggregation weights of the sub-model M_k
$\boldsymbol{h}_f$	The aggregated representation of a sequence
d	The hidden dimension
L	The maximum sequence length

A Implementation Details

A.1 Notation

We provide a table of the notation used in the main paper Table 4.

A.2 Data Preprocessing

Following prior work (Kang & McAuley, 2018; Sun et al., 2019; Deng & Hu, 2024; Yang et al., 2025), every dataset passes a 5-core filter before any sequence is formed, retaining only users and items with at least five recorded interactions. The surviving interactions of each user are ordered by timestamp into a single sequence V^u. The dataset is randomly partitioned into training, validation and test sets at an 8:1:1 ratio across unique users prior to sequence expansion, thereby establishing a strictly disjoint setting where an individual's complete history belongs to a single subset. Following this partition, each sequence of length $|V^u|$ is expanded into $|V^u| - 1$ sequence-target pairs, where the t-th pair matches the sequence $[v_1^u, \ldots, v_t^u]$ with the target v_{t+1}^u. Sequences are left-truncated to the most recent L interactions and left-padded with a reserved index 0. Models are optimised using cross-entropy loss over the full candidate set, ignoring padding tokens. The same process is applied in both the sub-model and the aggregator training phases.

A.3 Training Settings

Sub-model and Partitioning Settings. For data partitioning (Algorithm 1), we set the default number of shards to $K = 8$, with a maximum capacity of $C_{\max} = \lfloor N/K \rfloor + 5$ and $T = 50$ iterations. Cluster centroids are initialised using the k-means++ scheme (Arthur & Vassilvitskii, 2007). We fix the maximum sequence length L at 10 for Games/Beauty and 50 for MovieLens-1M. All sub-models share a hidden dimension $d = 64$. The Transformer-based models (SASRec and BERT4Rec) stack 2 layers. SASRec uses 1 attention head with dropout 0.5 on Games/Beauty and 0.3 on MovieLens-1M. BERT4Rec uses 2 heads with dropout 0.3 and masking probability 0.2 across all datasets. All sub-models are optimised using Adam with a learning rate of 10^{-3} and a batch size of 256. Training runs for a maximum of 100 epochs, with validation performed every 5 epochs. We employ early stopping with a patience of 20 epochs based on validation NDCG@10.

Aggregation Phase Settings. The aggregation module is trained for a maximum of 10 epochs. To prevent overfitting during aggregation, the ℓ_2 regularisation strength λ of equation 9 is tuned via grid search over the candidate set $\{0, 10^{-4}, 10^{-3}\}$.

Unlearning Settings. Following SRU (Xin et al., 2024), the unlearning ratio is defined as the percentage of users issuing an unlearning request, where each request removes one interaction. The ratio is 10% by default and varies from 0 to 50% in the resilience experiment (§5.2). Affected users are sampled within each shard at this ratio, and the removed interaction is chosen uniformly at random from each affected user's training sequence. The same unlearning ratio is applied to Retrain to ensure a fair comparison.

Random Seeds Settings. The main results (Table 5 and Table 2) and the ablation studies (§5.4) use five seeds $\{1, 2, 3, 4, 5\}$. The unlearning ratio experiment uses ten seeds $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

Computing Environment. All experiments are conducted across two identical servers, each equipped with a single NVIDIA RTX A4000 GPU (16 GB GDDR6), an Intel Core i9-12900K CPU, and 128 GB DDR5 RAM, running Ubuntu 22.04 LTS with PyTorch 2.5.1 and CUDA 11.8.

B Extended Results and Analysis

B.1 Performance before unlearning

Table 5 reports the recommendation performance of all methods before any unlearning request is processed, across three datasets and three representative sequential recommendation models.

B.2 Impact of the hidden representation cache

Table 6 replicates the efficiency experiments from §5.3 with the hidden representation cache disabled. The aggregator recomputes the responses of the K frozen sub-models at every epoch. All other settings match Table 3. Consequently, the Retrain and Sub-model rows remain identical across both tables, as caching affects only the aggregator retraining.

Impact on Aggregator Retraining. The cache reduces the computational cost of aggregator retraining. On MovieLens-1M using BERT4Rec, the aggregator retraining time of SREraser decreases from 981.7 s to 113.6 s. Consequently, the total unlearning process requires 755.5 s instead of 1623.6 s. Without caching, recomputing K sub-model responses per epoch binds aggregator retraining time to the backbone's forward-pass cost. This dependence creates up to a 7.2× variance across backbones on the same dataset (136.7 s for GRU4Rec versus 981.7 s for BERT4Rec on MovieLens-1M). With the cache enabled, the aggregator retraining time depends primarily on the dataset size and remains stable across shard numbers. On Games using SASRec, the aggregator retraining time rises from 18.06 s ($K = 4$) to 19.08 s ($K = 20$). This stability preserves the $1/K$ sub-model efficiency gain (which drops from 121.50 s to 18.27 s), preventing aggregator overhead from negating the overall speedup. Table 9 reports both components across the whole range of K.

Table 5: Performance comparison between our SREraser and other efficient unlearning methods (SISA, SRU) prior to any unlearning requests. All values are reported as the mean over five seeds, scaled by 10^{-3}. The best results are bolded.

		SASRec				BERT4Rec				GRU4Rec			
		Retrain	SISA	SRU	SREraser	Retrain	SISA	SRU	SREraser	Retrain	SISA	SRU	SREraser
Games	NDCG@5↑	61.4	36.5	36.8	**41.8**	46.9	35.1	38.6	**41.6**	52.2	25.5	30.9	**33.3**
	NDCG@10↑	75.0	43.6	45.7	**52.4**	61.3	44.9	50.3	**53.8**	64.2	30.3	38.4	**41.8**
	NDCG@20↑	89.8	51.1	55.9	**64.0**	76.6	55.1	63.1	**66.8**	75.8	35.4	46.6	**50.6**
	Recall@5↑	90.1	52.0	54.9	**62.7**	73.2	53.9	59.3	**63.9**	76.4	36.0	45.7	**49.5**
	Recall@10↑	132.4	74.0	83.0	**95.7**	117.8	84.4	95.6	**101.7**	113.6	51.0	69.1	**76.1**
	Recall@20↑	191.5	103.5	123.4	**141.9**	178.8	125.0	146.6	**153.5**	159.9	71.4	101.6	**111.0**
Beauty	NDCG@5↑	70.6	47.6	48.0	**51.5**	55.2	43.1	44.8	**45.4**	65.0	43.7	42.8	**46.9**
	NDCG@10↑	82.0	55.1	55.9	**60.4**	67.0	51.4	53.7	**54.9**	74.9	49.3	50.3	**54.4**
	NDCG@20↑	93.0	62.5	64.0	**69.6**	78.5	59.9	63.1	**65.4**	84.5	55.1	58.2	**62.7**
	Recall@5↑	97.7	64.5	65.6	**70.9**	80.5	62.3	64.0	**65.4**	88.1	58.5	59.0	**64.2**
	Recall@10↑	133.0	88.0	90.0	**98.5**	117.0	88.1	91.6	**94.7**	118.8	76.1	82.5	**87.4**
	Recall@20↑	176.8	117.5	122.3	**135.2**	162.7	121.8	129.1	**136.3**	157.1	98.9	113.6	**120.8**
MovieLens-1M	NDCG@5↑	120.9	107.0	106.0	**108.1**	93.8	90.4	**90.7**	89.8	101.8	95.9	94.2	**96.4**
	NDCG@10↑	148.6	131.6	132.9	**135.6**	121.6	116.5	**118.0**	117.1	127.1	118.8	119.6	**122.2**
	NDCG@20↑	176.8	157.3	161.2	**164.2**	151.8	145.5	**148.2**	147.6	153.9	143.9	147.0	**150.1**
	Recall@5↑	179.6	158.5	159.8	**162.9**	146.0	139.8	**140.5**	139.4	152.8	142.3	142.7	**146.1**
	Recall@10↑	265.7	235.1	243.4	**248.7**	232.7	221.3	**225.4**	224.4	231.5	213.8	221.9	**226.2**
	Recall@20↑	377.8	337.5	355.7	**362.1**	352.8	336.6	**345.6**	**345.6**	337.6	313.5	330.6	**337.2**

Table 6: Unlearning time for a single unlearning request across Retrain, SREraser, and SRU when the aggregator is trained *without* the hidden representation cache.

Time (s)↓		**Games**			**Beauty**			**MovieLens-1M**		
		SASRec	BERT4Rec	GRU4Rec	SASRec	BERT4Rec	GRU4Rec	SASRec	BERT4Rec	GRU4Rec
Retrain		474.64	385.44	123.23	444.90	406.82	115.01	4655.33	6237.41	1536.64
SRU	Sub-model	60.55	54.22	23.29	48.55	36.19	22.75	569.31	671.69	139.38
	Aggregation	59.58	54.13	25.3	50.66	46.05	21.31	592.47	983.69	138.7
	Total	120.13	108.35	48.59	99.21	82.24	44.06	1161.78	1655.38	278.08
SREraser	Sub-model	60.55	53.37	22.96	46.27	38.26	23.00	570.95	641.91	141.18
	Aggregation	58.7	54.5	19.9	50.8	46.6	17.3	596	981.7	136.7
	Total	119.25	107.87	42.86	97.07	84.86	40.30	1166.95	1623.61	277.88

Cache Refresh Mechanism. The cache stores one representation per training sequence from each sub-model. Since our data processing strategy (Appendix A.2) generates $|V^u| - 1$ sequences for a user with $|V^u|$ interactions, each sub-model encodes exactly $N_{\text{seq}} = \sum_u (|V^u| - 1)$ sequences, yielding a total cache size of KN_{seq} representations.

Processing an unlearning request invalidates two distinct sets of cache entries. First, the retrained sub-model M_k generates updated representations for every sequence, rendering all N_{seq} entries generated by this sub-model stale. Second, deleted interactions modify the input sequences of the affected users. Consequently, the representations of these sequences under the remaining $K - 1$ sub-models also become stale. Affected users are drawn uniformly at random, so with ρ denoting the unlearning ratio they account for ρN_{seq} sequences in expectation, and a refresh therefore has to recompute

$$\big(1 + (K-1)\rho\big)\, N_{\text{seq}} \tag{10}$$

of the KN_{seq} cached representations, assuming all unlearning requests target a single shard. A single request represents the lower bound of this formulation, necessitating an update to $1/K$ of the cache (i.e., 12.5% when $K = 8$). The aggregator retraining times reported in Table 3 account for this cache refresh overhead. In practice, we quantify this computational cost by first measuring the baseline time required for a single sub-model to encode all N_{seq} sequences. We then determine the final cache refresh time for each unlearning request by multiplying this single-pass baseline by the theoretical scaling factor $1 + (K-1)\rho$.

Table 7: Partitioning ablation on Games, aggregation fixed to ADA: mean ± standard deviation over five seeds, scaled by 10^{-3}. The best results are bolded.

Games		Random	KMeans	SKMeans	UCSP
SASRec	NDCG@5↑	41.1 ±1.1	41.4 ±0.8	41.4 ±1.1	**41.8 ±0.9**
	NDCG@10↑	51.2 ±1.0	51.9 ±0.8	52.0 ±0.9	**52.4 ±0.6**
	NDCG@20↑	62.0 ±1.0	63.5 ±0.6	63.8 ±0.8	**64.0 ±1.0**
	Recall@5↑	61.1 ±1.7	62.2 ±1.1	61.8 ±1.3	**62.7 ±1.5**
	Recall@10↑	92.5 ±1.9	94.8 ±1.9	94.8 ±0.9	**95.7 ±0.9**
	Recall@20↑	135.7 ±2.3	141.2 ±1.3	141.7 ±2.1	**141.9 ±3.0**
BERT4Rec	NDCG@5↑	40.2 ±0.6	40.8 ±1.2	40.6 ±0.7	**41.6 ±0.4**
	NDCG@10↑	52.0 ±0.7	52.8 ±1.3	52.9 ±0.8	**53.8 ±0.9**
	NDCG@20↑	65.1 ±0.7	66.5 ±1.7	66.5 ±0.9	**66.8 ±0.6**
	Recall@5↑	61.7 ±1.3	62.8 ±1.6	62.6 ±1.1	**63.9 ±0.4**
	Recall@10↑	98.3 ±1.8	100.4 ±2.2	100.8 ±2.0	**101.7 ±2.4**
	Recall@20↑	150.6 ±1.9	154.8 ±4.0	**155.0 ±3.4**	153.5 ±1.4
GRU4Rec	NDCG@5↑	**33.6 ±0.7**	32.9 ±0.3	33.4 ±1.0	33.3 ±0.8
	NDCG@10↑	41.4 ±0.7	41.1 ±0.4	**41.9 ±0.7**	41.8 ±0.8
	NDCG@20↑	49.5 ±0.8	50.3 ±0.3	**50.9 ±0.7**	50.6 ±0.7
	Recall@5↑	49.3 ±1.1	48.6 ±0.6	**49.5 ±1.2**	**49.5 ±1.1**
	Recall@10↑	73.4 ±1.4	74.0 ±0.8	76.0 ±0.9	**76.1 ±1.6**
	Recall@20↑	105.7 ±1.6	110.9 ±1.2	**111.6 ±0.8**	111.0 ±1.9

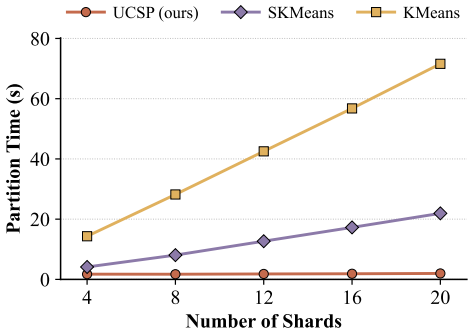

Figure 7: Partitioning time on Games as the shard number K varies, mean over five seeds.

B.3 Partitioning ablation

Table 7 gives the full six-metric view of Figure 2 on Games, averaged over five seeds.

Figure 7 shows how partitioning cost scales with the shard number K on Games. Between $K = 4$ and $K = 20$, KMeans rises from 14.33 s to 71.58 s and SKMeans from 4.12 s to 21.93 s, whereas UCSP rises only from 1.73 s to 1.97 s. UCSP therefore achieves the lowest partitioning overhead of the three similarity-based clustering strategies at every shard number tested.

B.4 Aggregation ablation

Table 8 gives the full six-metric view of Figure 4 on Games, averaged over five seeds.

B.5 Shard number ablation

Table 9 decomposes the unlearning time plotted in Figure 6 into the sub-model retrain and the aggregator retrain.

Table 8: Aggregation ablation on Games, partitioning fixed to UCSP: mean ± standard deviation over five seeds, scaled by 10^{-3}. The best results are bolded. MeanAgg-L and MeanAgg-H apply uniform weights, fusing at the logit level and aligned hidden representation level, respectively.

Games		MeanAgg-L	NearAgg	MeanAgg-H	SRUAgg	ADA
SASRec	NDCG@5↑	14.5 ±0.5	20.5 ±0.7	36.8 ±1.0	37.4 ±0.5	**41.8 ±0.9**
	NDCG@10↑	18.1 ±0.6	26.6 ±0.6	46.1 ±0.8	46.5 ±0.6	**52.4 ±0.6**
	NDCG@20↑	22.4 ±0.7	33.8 ±0.7	55.9 ±1.0	56.5 ±0.7	**64.0 ±1.0**
	Recall@5↑	21.1 ±0.8	31.2 ±1.0	54.3 ±1.6	55.3 ±1.1	**62.7 ±1.5**
	Recall@10↑	32.2 ±1.2	50.2 ±1.3	83.0 ±1.4	83.6 ±1.7	**95.7 ±0.9**
	Recall@20↑	49.6 ±1.6	78.6 ±2.1	122.1 ±3.2	123.7 ±1.7	**141.9 ±3.0**
BERT4Rec	NDCG@5↑	16.6 ±0.4	18.6 ±1.1	37.2 ±0.9	38.4 ±1.0	**41.6 ±0.4**
	NDCG@10↑	21.8 ±0.4	24.3 ±1.5	48.4 ±0.8	49.8 ±0.8	**53.8 ±0.9**
	NDCG@20↑	27.7 ±0.7	31.2 ±1.8	60.8 ±0.7	62.3 ±1.4	**66.8 ±0.6**
	Recall@5↑	25.9 ±0.8	29.1 ±1.2	57.3 ±0.8	59.1 ±1.7	**63.9 ±0.4**
	Recall@10↑	42.3 ±0.6	46.8 ±2.7	92.0 ±0.7	94.6 ±1.9	**101.7 ±2.4**
	Recall@20↑	65.7 ±2.1	74.2 ±4.1	141.5 ±0.4	144.3 ±3.4	**153.5 ±1.4**
GRU4Rec	NDCG@5↑	16.5 ±0.8	23.3 ±0.8	31.0 ±1.0	30.9 ±0.1	**33.3 ±0.8**
	NDCG@10↑	20.3 ±0.9	28.6 ±0.9	38.0 ±0.8	38.2 ±0.4	**41.8 ±0.8**
	NDCG@20↑	24.5 ±0.9	34.2 ±1.1	46.0 ±0.8	46.5 ±0.8	**50.6 ±0.7**
	Recall@5↑	23.8 ±1.2	33.8 ±1.1	45.2 ±1.7	45.3 ±0.3	**49.5 ±1.1**
	Recall@10↑	35.4 ±1.4	50.2 ±1.5	67.1 ±1.5	68.2 ±1.3	**76.1 ±1.6**
	Recall@20↑	52.3 ±1.6	72.6 ±2.6	98.9 ±1.8	101.3 ±2.8	**111.0 ±1.9**

Table 9: Unlearning time for a single unlearning request on Games as the shard number K varies, matching Figure 6, which plots the Total unlearning time.

Time (s)↓		Number of shards K				
		4	8	12	16	20
SASRec	Sub-model	121.50	60.55	35.10	23.00	18.27
	Aggregation	18.06	18.53	18.26	18.66	19.08
	Total	139.56	79.08	53.36	41.66	37.35
BERT4Rec	Sub-model	115.36	53.37	34.48	26.65	15.35
	Aggregation	18.80	18.96	18.92	19.02	19.06
	Total	134.16	72.33	53.40	45.67	34.41
GRU4Rec	Sub-model	38.00	22.96	17.40	14.50	12.37
	Aggregation	18.09	17.80	18.19	17.99	18.55
	Total	56.09	40.76	35.59	32.49	30.92

C Future Work

Future work of SREraser will focus on extending its efficiency mechanisms to continuous unlearning scenarios and parameter-efficient architectures. Systems typically receive unlearning requests as a continuous stream, whereas the current formulation processes each request in isolation. Batching and scheduling strategies provide a mechanism to share the sub-model and aggregator retraining cost across multiple requests, further reducing the per-request unlearning overhead. Furthermore, because ADA consumes only the final hidden representation produced by each frozen sub-model, the architecture inherently supports treating sub-models as adapters over a shared pre-trained recommender. Transitioning sub-models to parameter-efficient experts restricts rebuild costs to single-expert adaptation, though the efficacy of SREraser on shared backbones requires formal empirical validation.