Universiteit Leiden

Master Computer Science

Unlearning for sequential recommender systems

Name:	Xiang He
Student ID:	s3627136
Date:	13/11/2024
Specialisation:	Data Science
1st supervisor:	Dr. Zhaochun Ren
2nd supervisor:	Prof. dr. Suzan Verberne

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Sequential recommender systems (SRSs) predict user next preferences for items based on their chronological historical interactions. These systems are widely used on online platforms such as e-commerce, video streaming, and news websites. However, real-world platforms may require the deletion of the outdated or inaccurate items. Therefore, machine unlearning techniques are becoming increasingly attractive in SRS. Existing unlearning methods, primarily developed for computer vision and large language models, are not tailored to the chronological data used in SRSs. To address this problem, we propose Sequential-based Recommendation Unlearning with Gradient Ascent (SRUGA), a lightweight framework that achieves the unlearning effect approximately to retrain the entire model. Specifically, we apply gradient ascent on the to-be-forgotten data, rendering the model incapable of correctly predicting the forgotten items. Simultaneously, Kullback-Leibler divergence is applied to fine-tune the model on the remaining data, minimizing the negative impact of gradient ascent on overall recommendation performance. Moreover, we propose a data partitioning strategy that preserves the chronological dependencies of user interactions during unlearning, ensuring that forgotten items cannot be inferred from earlier data. Experimental results show that SRUGA achieves effective unlearning while being more stable and faster than retraining-based methods. We confirm that SRUGA offers an efficient solution to approximate unlearning in sequential recommender systems, advancing privacy protection in these models.

Contents

1 Introduction

The volume of data in the world is growing exponentially. Nowadays, the internet has become the primary source for news, entertainment, and online shopping. Recommender systems serves as the core technique for providing such information by recognizing user preferences. Given the development of deep learning techniques LeCun et al. [2015], the reliability of recommender systems has also grown rapidly in recent years. However, despite the power of deep learning techniques, concerns have been raised about their safety, particularly regarding data usage and privacy Hitaj et al. [2017].

Security concerns related to deep learning are gathering increasing attention. Carlini et al. [2021] mentioned the problem of training data memorization in 2021. They point out that the attacker could extract the training samples from a large language model (LLM) such as GPT-2. They show how personal privacy, such as name, phone number, physical address, and even the content of online conversations, could be explored through their extraction attack. As a result, some users may request the data holder to delete their information completely from the systems. Laws like the General Data Protection Regulation (GDPR)[1] in the EU, the California Consumer Privacy Act (CCPA)[2] in the US, and the Act on the Protection of Personal Information (APPI)[3] in Japan also indicate that the users have the right to require data holders to stop using their personal information, which is so-called 'the right to be forgotten.'

Deep learning models the hidden state of data features, which makes it difficult to remove user information as easily as traditional databases. In most cases, the cost of retraining the entire model for deleting a small amount of data is unacceptable. Therefore, an efficient unlearning method is needed. A typical solution is to train the model following the "Sharded, Isolated, Sliced, and Aggregated (SISA)" framework Bourtoule et al. [2021] for machine unlearning. They split complete data into smaller shards and slices, training a sub-model for each shard independently, then aggregating the results of the sub-models to produce the model output. As a result, when users request to delete their data, it only needs to retrain a single sub-model, dramatically improving the unlearning efficiency.

However, the situation is more complex in recommender systems. The data consists of users' interactions, with each interaction highly correlated with others. Thus, even when a user's sample is removed from the training data, the recommender system can still predict the user's preferences based on other user's samples. Moreover, retraining-based unlearning methods like SISA are still resource-intensive. Although some retraining-free methods exist, such as the one proposed by Mehta et al. [2022], they rely on identifying model parameters associated with the data to be forgotten by calculating the Hessian matrix. By inhibiting these parameters, the model can effectively "unlearn" specific data. However, the high computational complexity of Hessian matrix calculation limits the scalability of this method.

To address the above issue, we propose Sequential-based Recommendation Unlearning with Gradient Ascent (SRUGA) to effectively unlearn user information from SRSs without retraining

[1] https://eur-lex.europa.eu/eli/reg/2016/679/oj

[2] https://oag.ca.gov/privacy/ccpa

[3] https://www.cas.go.jp/jp/seisaku/hourei/data/APPI

the model. Inspired by recent studies on large language model unlearning Yao et al. [2024] , we apply gradient ascent to perturb the parameters corresponding to the data to be forgotten in the recommender system. In addition, we fine-tune the unlearned model on the remaining datasets to slow down the speed of the gradient ascent that destroys the model, while KL divergence is used to reduce the fluctuations during this process. Since sequential dependency is important in SRSs, we implement a specific data partition strategy and evaluation metric for our task. In addition, we expect a potential conflict between the targets of optimizing the model's recommendation performance and unlearning effects. We trying to find the balance in our research.

Therefore, our research questions are as follows

- (RQ1) The effectiveness of unlearning: Can the gradient ascent method make the model forget data, and how much does it affect the recommendation performance?
- (RQ2) Recommendation performance: Does KL divergence and fine-turning method mitigate the negative impact of gradient ascent on model performance?
- (RQ3) Efficiency: How much faster is our algorithm than retraining-based baselines?
- (RQ4) Optimization: How do we balance the impact across three loss functions, and does the model converge after unlearning?

We evaluate our unlearning method on three representative sequential recommender systems, GRU4Rec, SASRec, and BERT4Rec, with varying sparsity datasets: Amazon Beauty, Amazon Games, and Steam. We use the systems that entirely retrained on the datasets after deletion and SISA as the baselines for each experiment. The results demonstrate that our approach achieves effective unlearning while maintaining comparable recommendation performance to retraining-based methods.

- We propose an approximate unlearning algorithm that provides a faster alternative to retraining-based methods.
- We propose a data partition strategy for addressing forgotten data, which is specifically designed for the sequential recommendation task.
- There is not sufficient research on approximate unlearning for sequential recommender systems. We provide a novelty solution to this research field.

2 Background

2.1 Recommender systems

Rich [1979] first introduced the concept of recommender systems in 1979 to respond to the growing information overload. She designed a recommender system, Grundy, to recommend books to users based on asking specific questions. This method categorizes the users into different groups and assumes the users have similar preferences in the same group. The system collected user ratings through direct questioning, known as explicit recommendation. Although the method is intuitive, it is inefficient and low-dimensional. Karlgren [1990] introduced the implicit recommendation in 1990, trying to capture user preferences through their historical interactions. The concept of these earlier-stage works inspires modern recommender systems. The implicit recommendation remains the core idea for precision recommendations. At the same time, explicit information is useful for addressing the cold start problem, where new users lack historical information.

2.1.1 Traditional recommender systems

Resnick et al. [1994] proposed the GroupLens system in 1994, introducing collaborative filtering from information retrieval into the recommender systems field. They calculate the similarity between users by user-based collaborative filtering, predicting a user's ratings based on the historical ratings of similar users. In addition, they introduced one of the most widely used recommendation datasets, MovieLens, for evaluating recommendation performance. Many other recommender systems based on collaborative filtering have appeared at that stage. Sarwar et al. [2001] proposed item-based collaborative filtering, which calculates the similarity between items rather than users. The system proved more stable since the number of items changed less frequently than that of users. The content-based collaborative filtering from Van Meteren and Van Someren [2000] uses implicit information such as the user's gender and age, the item's label and brand, which can be represented as feature vectors. These features are used to calculate the similarity between users or items in a multi-dimensional space. Therefore, these feature engineering-based methods achieve better performance in complex recommendation scenarios.

However, collaborative filtering-based methods often ignore the users' preferences can change over time. Users may have purchased certain products a lot in the past but prefer other products now. Therefore, it is important to capture the long-term preferences of users. Moreover, traditional recommender systems rely on the user-item matrix. The dimensionality of the matrix can increase rapidly when new users and items are added. Thus, neural recommender systems are designed to overcome these problems.

2.1.2 Neural recommender systems

In the early stages, neural recommender systems use machine learning techniques to model non-linear relationships. Logistic regression and XGBoost Wang et al. [2016] are applied to capture user preferences, enabling the system to learn high-dimensional patterns from user interactions. Additionally, these methods treat recommendation as a classification task rather than the jigsaw puzzle of collaborative filtering. The model predicts the likelihood of a user's interest in various items and recommends the most relevant ones by ranking them. This

approach also allows the system to provide general recommendations for new users without the dimensional explosion problem in traditional methods. However, the machine learning-based method cannot model the chronological relation between users' historical interactions.

With the successful application of recurrent neural networks such as RNN Rumelhart et al. [1986] and LSTM Graves [2012] for natural language processing. Researchers have realized their potential in recommender systems. Recurrent units connect current information with previous hidden states, leading the model to capture the long-term dependency, helping the system to learn the user's dynamic behavior. Moreover, the graph network-based recommender system handles the items and users as graph nodes. A classic implementation is the GraphRec Fan et al. [2019] proposed by W Fan et. al in 2019. They use multi-graphs to model the user-user, user-item, and item-item relations. The system can fuse relational information from the above three graphs to make accurate recommendations. The choice of recommendation models depends on specific scenarios. In the following sections, we describe the division of recommendation stages and the models used in sequential recommendation tasks.

2.2 Sequential recommendations

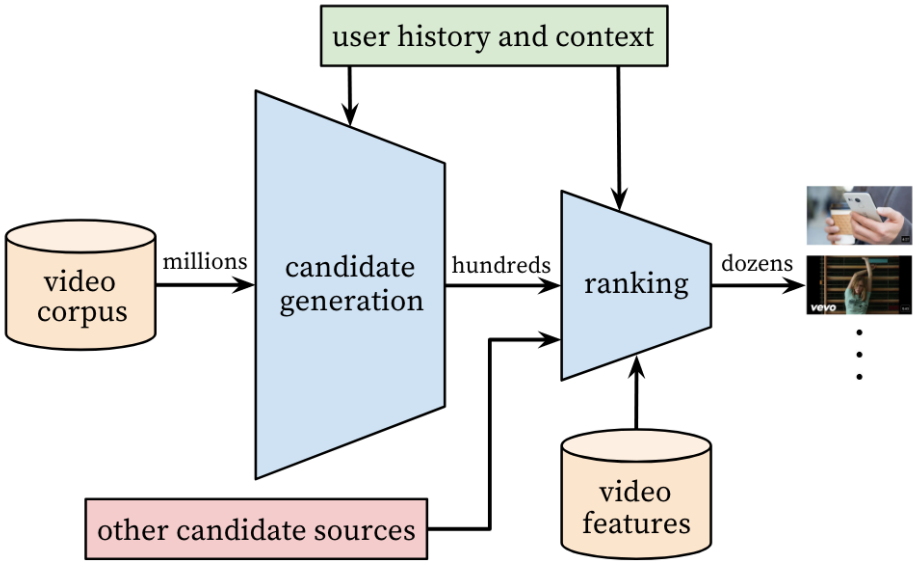


Figure 1: The recommendation pipeline of YouTube Covington et al. [2016]

In real-world applications, recommender systems are generally divided into two stages. Figure 1 demonstrates the architecture of YouTube's recommendation system in 2016. In the candidate generation stage, the system filters high-recall candidates from millions of videos. This stage captures the relations between the user's interaction history and potential videos. The second stage re-ranks these candidates using additional video features to provide personalized recommendations. We focus on the candidate generation stage in this paper.

Traditional recommender systems usually treat user history interactions as static behavior to capture their overall preferences. However, in real-world cases, user preferences are generally dynamic and change over time. The data with temporal order in recommendation systems

are typically represented as sessions or sequences, corresponding to session-based (SBRS) and sequential-based (SRS) recommender systems. These systems employ different approaches to process the temporal patterns of user interactions.

2.2.1 Session-based recommendation

Session-based approaches use precise time points to divide interactions occurring in different periods into multiple sessions, each reflecting the user's preferences at a particular moment. This method simulates real-life scenarios by capturing the user's preference phases. For example, in a news website, the user's reading activities are often concentrated within a single time period. By learning these phases, SBRSs can predict the next article in a session or recommend a basket of items for a new session. Moreover, the interactions within a session can be unordered when dealing with anonymous or co-occurrence data. For example, a user purchased a cart of items in a single shopping session, but we don't know the exact order in which those items were browsed. In such cases, the session consists of a set of co-occurring items without any sequential dependencies. The system focuses on the relevance between the items in the session, rather than the chronological order in which they interacted.

GRU4Rec A classic session-based recommendation system is GRU4Rec, proposed by Hidasi et al. [2015] in 2015. They introduce RNNs into session-based recommendations, using GRU units to learn cumulative information from previous items and predict the next item in the ordered session. To improve learning efficiency, they also designed a session-parallel mini-batch technique. Their results demonstrate 25% improvements over traditional methods like Item-KNN Schmidt et al. [2024] and BPR-MF Rendle et al. [2012] in $Recall@20$.

2.2.2 Sequential-based recommendation

When sorting the user's historical interactions by time, we get a sequence that reflects the user's preferences changing over time. By modeling the sequential dependencies within this sequence, we predict the next item that the user may like. The recommender systems designed to process such sequences are called sequential recommender systems.

Different from the SBRSs, which allow unordered interactions in sessions, SRSs organize user interactions in strict chronological order. Each user is represented by a single sequence of interactions, without considering time intervals between them. The SRSs are usually applied in the platforms that serve specific products. For example, in online education, SRSs can recommend advanced courses to a user who has completed introductory-level courses, without needing to know when these courses are taken. Also, SRSs can recommend personalized songs based on users' song records. However, modern SRSs are still challenged in the multi-domains recommendations. In the case of Amazon, if a user continues to purchase cosmetics and digital products, SRSs may incorrectly learn false dependencies between these unrelated preferences.

One of the solutions is to use attention mechanics, which allows the model to calculate the correlation between all items in a sequence. Attention models assign different weights to different items, ensuring the system focuses on the most relevant items, regardless of their positions in the sequence.

Attention mechanisms The transformer structure proposed by Vaswani et al. [2017] can be seen as the most important innovation in deep learning in recent years. The core idea behind it, the attention mechanisms, has changed the methodology of NLP research, especially in modeling long-term sequences. Unlike RNN-based methods, attention mechanisms do not focus on the sequence order. The authors propose to project the sequence into vector matrices and calculate the similarity between each item in the sequence through metrics dot product. Calculating the weighted similarity scores at each vector dimension as results. This approach allows the model to learn the items' dependencies without considering distance. Therefore, avoiding the issue of gradient vanishing that may happen when using RNN to process extremely long sequences Pascanu et al. [2013]. Additionally, matrix multiplication enables efficient parallel computation, improving the processing speed.

SASRec Self-attentive sequential recommendation Kang and McAuley [2018] is one of the most important advancements in SRSs in recent years. The author observes the advantages of the Markov Chain-based (MC) recommender systems Rendle et al. [2010] on sparse datasets, which recommender the next item based on the previous one or a few interactions. However, the status space of MC-based systems can be extremely complex when processing long sequences. On the other hand, RNN-based recommender systems Rumelhart et al. [1986] are effective at summarizing past interactions, but weak at capturing short-term preferences. Since the recommendation goal can differ with datasets' sparsity, the authors attempt to use the attention mechanism to focus on the most important interactions in a sequence. They design a unidirectional attention mechanism. The next item in the sequence is used as the label for the current sub-sequence, supervising the model to predict the next item based only on the previous interactions at each time step. However, the traditional attention mechanism does not consider the order of interactions during training, which allows the model to predict a current item using "future" interactions. This causes a causal violation, making it difficult for the model to understand the next-item recommendation task. Therefore, they use causal attention masks to invisible the future items. Besides, they introduce the feed-forward neural (FFN) Bebis and Georgiopoulos [1994] network and residual connections Szegedy et al. [2017] to enhance the model stability. Experimental results show that their model can capture important interactions in varying sparsity datasets and predict the next item that users may like.

BERT4Rec Sequential recommendation with bidirectional encoder representations from the transformer (BERT4Rec) is another impression work of SRSs. Sun et al. [2019] point out that the unidirectional model, like SASRec, did not fully utilize the advantage of the attention mechanism. Restricted by the causality of the unidirectional model, they only used interactions before the current position in a sequence to predict the next item, ignoring the remaining context. Therefore, they propose to use the bidirectional structure as BERT Devlin et al. [2018], training the sequence as a Cloze task Taylor [1953]. A special token randomly masks the interactions in a sequence. Then, the model predicts these masked items with both the left and right contexts. When applied to the next-item prediction, the mask token is added as a target at the end of the sequence. In summary, they implement a BERT-based masked language model for SRS, proving that bidirectional attention can capture historical and latest user features. Although this method does not follow the next-item recommendation causality during training, it still achieves better results than SASRec.

2.3 Machine unlearning

There has been increasing attention on machine unlearning research since the enactment of GDPR. Cao and Yang [2015] first propose the concept of machine unlearning, which aims to remove specific samples and their effects from a trained model. An intuition approach is to delete the data from the training set and retrain the entire model. However, this approach is costly and inefficient due to the complexity of modern models and the large datasets they use. Therefore, many strategies have been developed to improve unlearning efficiency, which can be broadly divided into the exact unlearning and the approximate unlearning Xu et al. [2024].

2.3.1 Exact unlearning

The strictest definition of exact unlearning is proposed by Nguyen et al. [2024]. They require the weight distribution and output distribution of the unlearned model to be identical to the model that has never seen the forgotten data. However, in this situation, only complete retraining can be considered exact unlearning. In practice, exact unlearning is commonly defined as the complete removal of the influence of the forgotten data, as a model had never seen it during training.

Therefore, exact unlearning methods are usually based on data partition and model retraining. A representative work is the Sharded, Isolated, Sliced, and Aggregated training (SISA) proposed by Bourtoule et al. [2021] in 2019. The training set is split into multiple shards, with each shard training a sub-model. The final predictions are made by aggregating the output of the sub-models. When data needs to be deleted, only the sub-model trained on the relevant shard requires retraining. A similar idea is employed by Yan et al. [2022] in ARCANE. They split the dataset by class into shards, with each shard used to train a sub-model. During training, each shard is further divided into smaller blocks, and the training states are saved at each block. When unlearning is required, only the block containing the forgotten data is deleted, and the model retraining from the saved state that corresponds to that block. Thus improving unlearning efficiency.

The unlearning effect of exact unlearning is desirable because it fully removes the forgotten data from the training set. However, these methods still require a retraining phase, which can be costly for large networks, particularly when only a small amount of data needs to be deleted.

2.3.2 Approximate unlearning

Approximate unlearning methods aim to balance unlearning performance and cost. These methods typically perturb specific parameters of the existing model to produce incorrect results for the data to be forgotten, which can enhance unlearning effectiveness during evaluation. However, some criticism Foster et al. [2024] points out that outputting incorrect results could lead to another kind of data leakage. Attackers may identify a candidate that the model is trying to avoid. Therefore, the unlearning goal should not be to lead the model to learn incorrect labels but to predict irrelevant results as it never saw. Another challenge is that model parameters are often interrelated, perturbing them can negatively impact the model's prediction ability.

Representative approximate unlearning methods include fine-tuning the model on the remaining data to reduce the impact of the to-be-forgotten data. However, this approach has limited unlearning effectiveness. Another method involves calculating the Hessian matrix Chen et al. [2021], which identifies the parameters associated with the to-be-forgotten data and inhibits them directly. This approach reduces the negative impact of unlearning irrelevant parameters, but its high computational complexity makes it hard to apply in large models.

Gradient ascent in machine unlearning Gradient descent (GD) is the most common optimizing method in machine learning. The main concept is to update the model in the opposite direction of the gradient to minimize the loss function. In contrast, gradient ascent (GA) updates model parameters to increase the loss. Therefore, applying it to the data to be forgotten can reduce the chance that unlearned models predict them, achieving machine unlearning. However, applying GA to specific data can also increase the loss of the entire model, leading to performance declines in normal data. Additionally, commonly used loss functions like cross-entropy are unbounded Halimi et al. [2023]. This means the losses from GA can increase without limit, leading to catastrophic forgetting, where the model's output becomes random.

To address this issue, existing machine unlearning methods that use gradient ascent typically adopt some strategies to maintain the model performance. One representative method is to use an additional model as a supervisor. The idea is to calculate the difference between the unlearning model and the supervisor to prevent heavy parameter changes. In the work of Yao et al. [2024], they proposed to calculate the distribution difference caused by gradient ascent (GA) using Kullback-Leibler (KL) divergence. KL divergence commonly used in regularization and knowledge distillation, measures the difference between two probability distributions. The authors use this method to approximate unlearning for a generative large language model (LLM). By combining KL divergence with label noise injection and gradient ascent, they prevent the LLM from outputting harmful data while maintaining generated performance. In summary, gradient ascent achieves effective unlearning, but often reduces model performance and requires additional techniques to mitigate this issue.

Another challenge of approximate unlearning is the evaluation of the unlearning effectiveness. The gold standard Foster et al. [2024] is to achieve the same performance as a model that has been entirely retrained. Researchers have to find a trade-off between prediction performance and unlearning effectiveness by themselves.

2.3.3 Unlearning in sequential recommendations

In sequential recommendations, applying machine unlearning involves removing one or more samples from the sequences. The goal is to ensure that the model cannot infer the forgotten samples from the remaining interactions, while maintaining continuous recommendation ability for the remaining samples in the sequence.

Exact unlearning often involves retraining the model, allowing it to relearn the sequential dependencies of the remaining sequence. Therefore, this approach is considered suitable for applying in SBRSs or SRSs. A successful implementation was demonstrated by Xin et al. [2024]. They introduce a SISA based unlearning framework into SBRS. During data partitioning, they

divided the training data into several shards based on similarity, training a sub-model for each shard. For aggregation, they weighted the sub-models' hidden states according to the similarity between the data shard and the input data. For machine unlearning, they removed additional data using collaborative, neighbor, or random strategies. These strategies prevent the forgotten sample from being inferred through related samples, enhancing unlearning effectiveness. This method outperforms traditional SISA in both recommendation accuracy and unlearning performance.

However, the situation is more complex in approximate unlearning. The algorithm needs to maintain the sequential dependencies of the remaining sequence while perturbing the model parameters. The research on approximate unlearning in sequential recommender systems is limited, and we hope this work will contribute to this field.

3 Methodology

3.1 Notations

Notation	Description
$\mathcal{U}, \mathcal{V}$	User and item set
$u \in \mathcal{U}, v \in \mathcal{V}$	User and item
S_u	Historical interaction sequence for a user S_u: $[v_1^u, v_2^u, \ldots, v_{\lvert S_u \rvert}^u]$
$n \in \mathbb{N}$	Maximum length of a sequence
$r_{fgt} \in (0, 1)$	Proportion of sequences to be forgotten
$\mathcal{D}$	The set of all sequences
$\mathcal{D}^{fgt}$	The set of to be forgotten sequences
$\mathcal{D}^{rem}$	The remaining set of $\mathcal{D}$ after remove to-be-forgotten items
y^{pos}, y^{neg}	The positive and the negative samples

Table 1: The summary of notations.

As shown in table1, $\mathcal{U}$ and $\mathcal{V}$ represent all the users and items set from dataset $\mathcal{D}$. For each user u, a n length sequence S_u represents the user's historical interacted items v in sequential order. During the unlearning, $r_{fgt}\%$ of users from $\mathcal{D}$ are selected as candidates to be forgotten. Then we extract the forgotten set $\mathcal{D}^{fgt}$ and the remaining set $\mathcal{D}^{rem}$ from $\mathcal{D}$ based on our data preparation strategy. We further describe the usage of these parameters in subsequent sections.

3.2 Dataset

We implement our algorithm on three real-world datasets commonly used for evaluating sequential recommender systems. These datasets demonstrate different sparsity, an essential factor in recommendation performance. In sparse datasets, users have limited interactions, and each item is interacted with only a few times. This often leads to cold-start problems and less diverse recommendations.

Amazon-Beauty and Games belong to Amazon-Review[4]. As the largest online shopping company in the world, it has collected many user ratings and comments for different product categories since 1996. We chose the data from beauty and video game products. Due to the large number of brands and products present in the market, these datasets are notable for their high sparsity.

Steam games datasets include users' ratings, reviews, and purchase history on the Steam platform. Game players' preferences are more similar compared to online shopping. Players buy and play the most popular games in each quarter. Therefore, recommender systems have a greater opportunity to infer the deleted items based on the preferences of similar users, which challenges the unlearning algorithm.

[4] https://cseweb.ucsd.edu/~jmcauley/datasets.html#amazon_reviews

Dataset	Users	Items	Avg. actions/users	Avg. actions/items
Amazon Beauty	52,864	55,618	6.8	6.5
Amazon Games	36,239	23,439	7.4	11.5
Steam	429,339	13,040	7.9	262.9

Table 2: Statistics of the datasets after preprocessing.

In the next-item recommendations, we only consider users' interactions with items. Following the same preprocessing of Kang and McAuley [2018], we treat user ratings and reviews of an item as an interaction. Each user and item was assigned a unique code starting with 1. Sorting a user's interactions with timestamps as a sequence, represented by $S_u : [v_1^u, v_2^u, \ldots, v_{|S_u|}^u]$. All the datasets we collected have been processed according to this method.

3.2.1 Data preparation

The data preprocessing in our study is designed to prepare the dataset for training, evaluation, and implementing machine unlearning techniques. When training a sequential recommender system, we define a maximum length of n for a sequence that can be considered for model training. This is an efficient design for matrix operations in the model, particularly for standardized embedding layers and position embedding, ensuring computational efficiency.

Some researchers, like the author of SASRec, ignore the items that exceed the maximum length in a sequence. In our work, we prefer to keep them. We use the padding and splitting strategies for different length sequences. For a short sequence, we pad it front with 0 to reach a length of n as $S_u : [0, 0, \ldots, v_1^u, v_2^u, \ldots, v_n^u]$. For a long sequence $S_u : [v_1^u, v_2^u, \ldots, v_{n+i}^u], i < n$, we split it into multiple sub-sequences $S_{u'} : [0, \ldots, v_1^{u'}, v_2^{u'}, \ldots, v_i^{u'}]$ and $S_{u''} : [v_i^{u''}, v_{i+1}^{u''}, \ldots, v_{n+i}^{u'}]$. The short sub-sequence after splitting is also padded to the length of n if it contains more than 2 items. Otherwise, we remove it. We tend to keep the most recent interactions entirely when dividing long sequences cause they represent the user's recent interests. Moreover, the user IDs are not actually used as training resources in sequential recommender systems. Therefore, each sub-sequence is equally treated as other sequences. This approach allows us to maximize the use of available data while maintaining a standard input format. In a sparse dataset like Amazon Beauty, where average user interactions are rare, assigning too large n can bring many gaps (0) into the sequence. Resulting in the model learning poor information per epoch. However, for some long sequences, a small n can lead to the model not learning the user's long-term preference. We set $n = 10$ to reach a balance. The statistics after preprocessed are shown in Table 2.

3.2.2 Data partitions

We extract the to-be-forgotten and the remaining parts from each dataset to simulate real-world unlearning scenarios. We denote the complete preprocessed dataset as $\mathcal{D}$, consisting of sequences $S_u \in \mathcal{D}$. This raw dataset $\mathcal{D}$ is divided into the forgotten set $\mathcal{D}^{fgt}$ composed of unlearning samples and the remaining set $\mathcal{D}^{rem}$ for evaluating the recommendation performance after unlearning. We aim to achieve the item-level unlearning rather than remove the entire sequence. Therefore, it is important to maintain the chronological order of sequences in both $\mathcal{D}^{fgt}$ and $\mathcal{D}^{rem}$. We design the following data partitioning strategy.

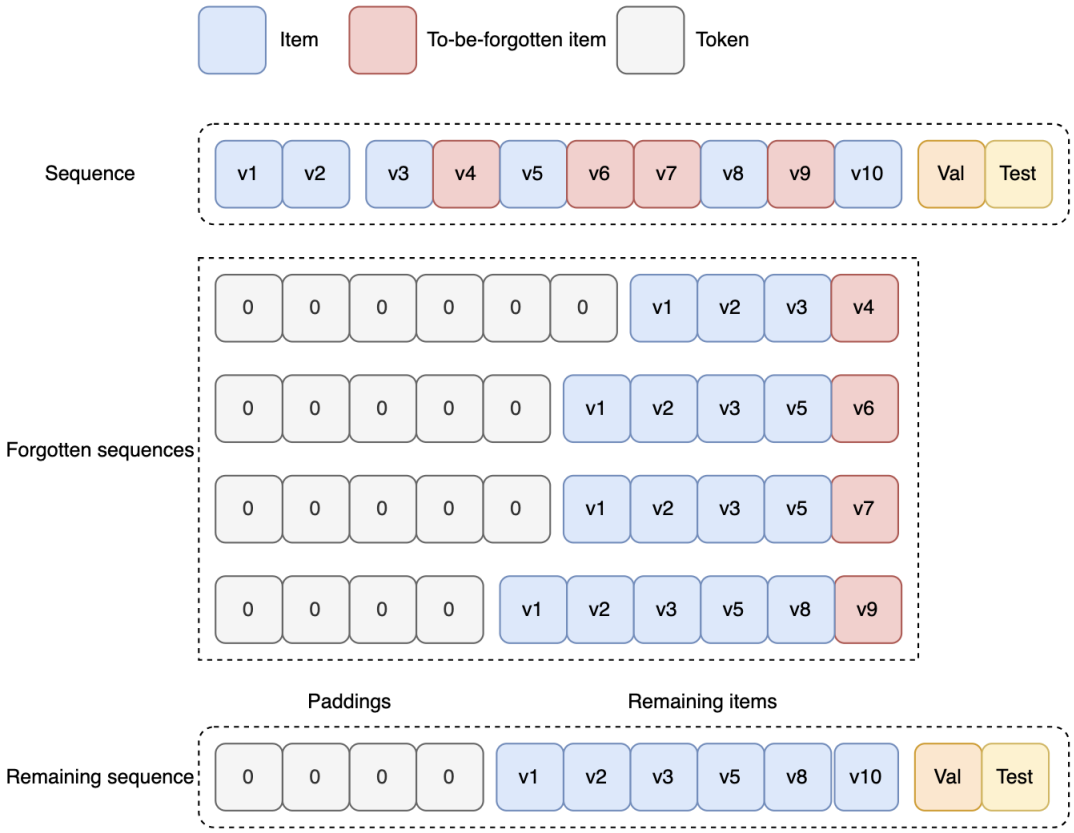


Figure 2: A case of data partition.

Forgotten set We randomly select $r_{fgt}\%$ of users from $\mathcal{D}$ as the to-be-forgotten candidates. For each candidate, we randomly choose no more than 50% of items from their sequence to unlearn. We avoid choosing the first two items in a sequence as the unlearning items to ensure sufficient content for SRS to infer them. In the case of figure 2, we generate four forgotten sequences corresponding to the four unlearned items. Each sequence consists of a to-be-forgotten item with its previous items. The generated sequences are also padded to the length of $n = 10$. This strategy is designed for our unlearning algorithm. In sequential recommender systems, each item is predicted based on the context of the sequence. Unlearning the item at the end of the sequence reduces the chance of the forgotten item being inferred from previous information.

Remaining set The remaining items that are not removed compose the remaining sequence with padding. Despite some items being removed, the remaining ones still follow the chronological order. Thus, the sequential dependencies of the sequences are maintained.

The data partition strategy is designed for unlearning requests in real-world recommendation systems. For example, on e-commerce platforms, users may request to delete records after purchasing or browsing private products or services, such as health products, financial services, or sensitive interests. However, they typically do not wish to delete all their historical interactions. Instead, after removing certain records, users still expect the system to recommend items based on their remaining preferences. From the platform's perspective, recommending items that align with each user's interests is essential for driving sales. This is also the pri-

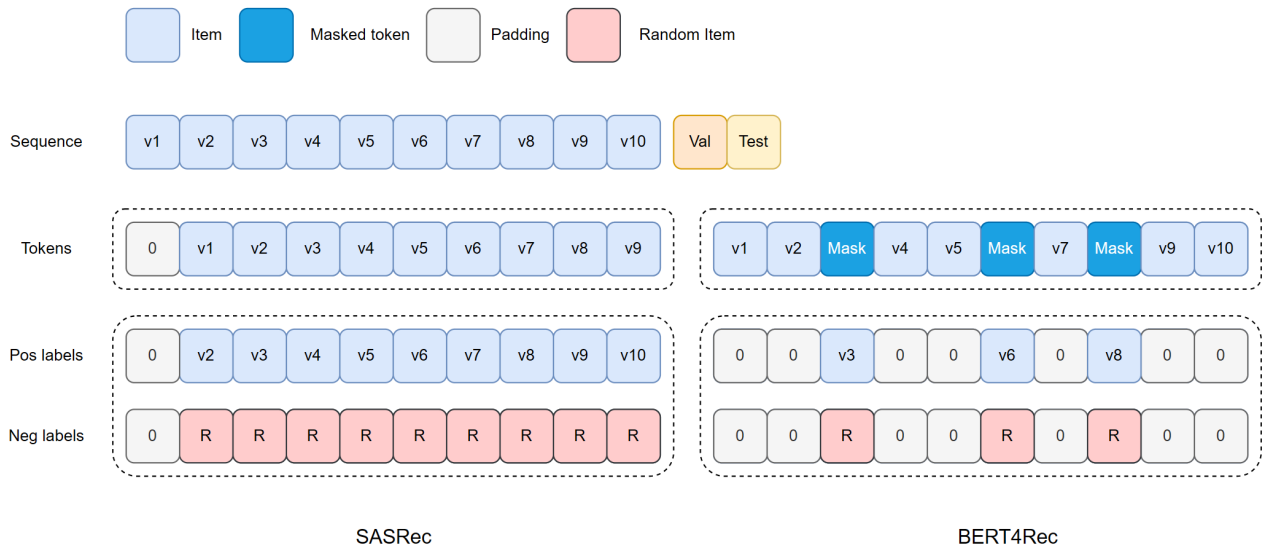


Figure 3: Data preprocessing of SASRec and BERT4Rec.

mary purpose of maintaining a recommendation system. Therefore, we focus on the item-level unlearning instead of the sequence-level in our work.

3.3 Training SRSs

We reproduce SASRec, BERT4Rec, and GRU4Rec on a uniform framework based on Jaywonchung's[5] BERT4Rec, which provides a well-defined code structure for training a sequential recommender system. We integrate the models of SASRec and GRU4Rec into this framework, modifying the data preprocessing, data loading, training, and evaluation modules. Additionally, we implement binary cross entropy (BCE) loss and negative sampling strategies for these models. We describe the training process and the modifications we made following.

3.3.1 Recommender models

SASRec The original source code of SASRec provided by Kang et al. is based on Tensorflow1.0 in Python2, which is an outdated framework and inconvenient to work with. Therefore, we use a reproduced version recommended by Kang on their GitHub[6], which is implemented in PyTorch of Python3 by Premixer[7]. We only used the model construction part of the code.

BERT4Rec The BERT4Rec is proposed by Fei Sun at el. from Alibaba. For the same reason as SASRec about the outdated framework, we use the reproduced version code from Jaywonchung's GitHub[8]. Different from the cross-entropy loss used by Fei Sun. They saw the recommendation as a multi-classification task by adding a fully connected layer to the end of the model. In our work, we use the Binary cross-entropy (BCE) loss to ensure that our unlearning algorithm can run at the same loss function, ensuring the comparability between different models in each experiment. Therefore, we remove the fully connected layer but add an

[5] https://github.com/jaywonchung/BERT4Rec-VAE-Pytorch
[6] https://github.com/kang205/SASRec
[7] https://github.com/pmixer/SASRec.pytorch
[8] https://github.com/jaywonchung/BERT4Rec-VAE-Pytorch

item embedding module for embedding the positive samples y^{pos} and negative samples y^{neg}. The embedded samples are dot products with the model's output to calculate the similarity, then calculate the loss by BCE with true labels consisting of 0 and 1 as a binary classification task.

Gru4Rec GRU4Rec, proposed by Balázs Hidasi, aims to learn the dynamic patterns of user interactions. We use the source code from Xin Xin et al. on their GitHub[9]. We implement the same sampling and training methods as SASRec and additional embedding layers for BCE loss calculation, with the same approach used in BERT4Rec. It is worth noting that GRU4Rec is designed for session-based recommendation in original work. The authors proposed a novelty sampling method to address the problem of varying lengths of user interactions in each session. However, it is not necessary for sequential recommendations that do not consider session relationships, only one interaction sequence for each user. Also, the GRU4Rec predicts the next recommended item at each time step. Therefore, using the next item in the sequence as the target of the current prediction, training it with the same sampling method as SASRec is allowed. Thus, we share the same loss function and training methods with SASRec.

3.3.2 Training

We have standardized the loss functions used for training SRSs. However, the sampling strategies differ between SASRec and BERT4Rec due to their model structure. SASRec learns the sequential dependencies by making predictions based on previous items at each timestep. As shown in figure 3 SASRec uses the next item in the sequence as the label for the current position. Here we denote the positive sequence as y^{pos} for the training sequence x, with a list of marks to indicate the positions where the item in y^{pos} is not 0. These can be seen as self-supervised labels. In sequential recommender systems, we use the next item v_{i+1} as the label for the current item v_i in a sequence. Thus, the positive samples of a training sequence $x : [0, 0, v_1, v_2, \ldots, v_8]$ should be $y^{pos} : [0, 0, v_2, v_3, \ldots, v_9]$ with label $[0, 0, 1, \ldots, 1]$. In contrast, BERT4Rec uses a cloze strategy, which randomly masks some of the items in the sequence as the label, and updating only relies on the losses of the masked positions. Thus, we construct the y^{pos} by assigning the items masked as prediction targets and padding 0 at other positions. The corresponding position is set to 1 in the true label used by BCE.

Negative sampling Generally, a user only interacts with a small portion of the items in the training items list $\mathcal{V}$. We want the model also to learn the relationship between the user and the items they haven't interacted with. Therefore, we also train the model using random negative samples. We use random negative sampling to generate negative samples. For each y^{pos}, we randomly generate a negative sequence y^{neg}, which consists of random items from the items list $\mathcal{V}$ that do not appear in y^{pos} and x. The generated items align in the same position as the true label at y^{pos}. This technique is commonly used in natural language processing (NLP) and recommender systems, which helps to balance the positive and negative samples when training a model, leading to improved recommendation performance and faster model convergence.

Then, we calculate the BCE loss with both y^{pos} and y^{neg} at each epoch, updating the model based on the Adam optimizer. Since the output of SRSs' is a ranking, we evaluate

[9] shirryliu/SRU-code: The source code for SRU. (github.com)

its performance by the $NDCG@10$ metric at every 10 epochs, and early stop when the best metric does not change in the last 50 epochs.

3.4 Unlearning functions

In this section, we describe the definitions of our unlearning algorithm SRUGA. The algorithm is divided into three loss functions based on binary cross-entropy (BCE): gradient ascent ($\mathcal{L}_{GA}$), fine-turning ($\mathcal{L}_{FT}$), and KL divergence ($\mathcal{L}_{KL}$). We define the to-be-forgotten sequence as $x^{fgt} \in \mathcal{D}^{fgt}$ and y^{fgt} as its positive samples.

3.4.1 Gradient ascent

In backward propagation, we calculate the derivatives of the losses for the model parameters to obtain the gradient of the loss function. The gradient represents the direction of the steepest ascent of the loss function in parameter space. We minimize the error between the model's predictions and the true labels to achieve convergence by updating the model parameters iterative in the opposite direction of the gradient. In contrast, gradient ascent updates the model parameters in the direction that maximizes the loss. Therefore, updating the model in the gradient ascent direction on specific samples can lead it to incorrectly predict them, achieving model unlearning.

$$\mathcal{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right) \tag{1}$$

We first demonstrate the process of BCE calculation and how we used it to implement gradient descent. As shown in equation1, BCE minimizes the difference between the model's predicted probability $\hat{y}$ and the binary labels y. Below, we illustrate how these features are processed in our work.

1. we denote $\hat{y}_i$ as the model's prediction at the i_{th} position of the sequence. Equation 2 illustrates the forward propagation process, where the sequence $x^{fgt} = [v_1, \ldots, v_{n-1}]$ is input into the model to generate the prediction of the positive samples using their embedded representations. To calculate the similarity between predictions with y^{pos}, we map the y^{pos} to the same embedding space, denoted as $M(y^{pos})$. Then we dot product the model's predictions with $M(y^{pos})$. After applying the sigmoid function, we obtain the similarity probability $\hat{y}_i$ for item v_i.

$$\hat{y}_i = f_\theta(v_{1\sim i-1}, M(v_i)) \tag{2}$$

2. The binary label y_i corresponds to the position of $\hat{y}_i$ in the sequence. $y_i = 1$ means we hope the model's prediction aligns with the positive samples as much as possible. Conversely, $y_i = 0$ when the model is trained on the negative samples to learn which sample should not be recommended. Therefore, we assign the binary label of the items in positive samples $y^{pos} = [0, \ldots, v_{i-1}, v_i]$ as $y = [0, \ldots, 1, 1]$, indicating that the model should accurately predict the target item v_i. For the same reason, the label of negative samples $y^{neg} = [0, \ldots, v_{rand}, v_{rand}]$ is assigned to $y = [0, \ldots, 0, 0]$, aiming to maximize the difference between the negative samples and the model's prediction.

Thus, we calculate the BCE loss between $\hat{y}_i$ and y_i for gradient descent. In general, gradient ascent can be implemented by directly reversing the update direction of the loss. Yuanshun Yao et al. use gradient ascent to prevent the generation of privacy-sensitive or harmful information in generative large language models (LLMs). In generation tasks, model unlearning aims to disrupt the LLM's ability to align queries with answers. Since queries are independent entities in the training set, gradient ascent can be directly applied by reversing the original loss function to unlearn the entire query-answer relationship. However, in sequential recommendation systems (SRSs), the sequence still exists in the training set after unlearning, with only specific items needing to be forgotten. Therefore, we focus on unlearning these particular items using the following steps.

$$\mathcal{L}_{BCE} = -\log(\hat{y}_i) \tag{3}$$

1. As mentioned above, we use 0 to pad the sequence to the required length, which is excluded from the BCE calculation. As shown in figure3, the forgotten sequence consists of a forgotten item with its previous items. In the case of $S^{fgt} = [0, \ldots, v_{i-2}, v_{i-1}, v_i]$, we aim to unlearn v_i from the SASRec. The input sequence is $x^{fgt} = [0, \ldots, v_{i-3}, v_{i-2}, v_{i-1}]$, then SASRec predict next item for each non-zero item. We then set $y^{fgt} = [0, \ldots, 0, v_i]$, corresponding to y^{pos}, and $y = [0, \ldots, 0, 1]$ as binary label. Then, we calculate the BCE only at the non-zero position of the sequence, which is the last position. By reversing the update direction of the BCE loss, we inhibit SASRec's ability to recommend v_i through previous continuous items while avoiding the effect of other items' recommendations. The forgotten item is placed at the end of the sequence because it contains the cumulative predictions made by the sequential recommender system based on all previous interactions. The same approach applies to BERT4Rec, where the input sequence is represented as $x^{fgt} = [0, \ldots, v_{i-2}, v_{i-1}, mask]$ with $y^{fgt} = [0, \ldots, 0, v_i]$. Thus, we simplify represent the BCE as equation3.

$$\mathcal{L}_{GA} = -\mathcal{L}_{BCE}(x^{fgt}, y^{fgt}) = \log(f_\theta(x^{fgt}, M(y^{fgt}))) \tag{4}$$

2. Additionally, we avoid using negative sampling in gradient ascent. Although reversing negative samples makes the model's output more random after unlearning, rather than deliberately avoiding correct results, which smooths the unlearning effect. However, it heavily decreases the model's recommendation performance in our experiment. Therefore, we implement gradient ascent (GA) using only positive samples. The loss function for GA is represented in equation 4.

$$\theta_{t+1} = \theta_t - \eta_{GA} \nabla_{\theta_t} \mathcal{L}_{GA} \tag{5}$$

We denote θ_t as the model's parameters that need to be unlearned at the current time t and η_{GA} as the learning rate of GA. The update procedure is represented as equation 5:

3.4.2 Kullback-Leibler Loss:

After performing gradient ascent, we perturb the model parameters to make the model incorrect in predicting specific input sequences. However, backward propagation acts on all model parameters. Gradient ascent can lead to catastrophic forgetting. Therefore, we need some functions to reduce the negative impacts of gradient ascent. A common approach is to introduce the Kullback-Leibler divergence (KL) as an additional loss function.

Kullback-Leibler divergence, also known as relative entropy, was proposed by Kullback and Leibler [1951] to measure the difference between two probability distributions. In our work, we design an agent model θ_A, which is copied from the original state of the unlearned model θ. The parameters of θ_A are frozen and not be updated during unlearning. We calculate the KL divergence between the outputs of θ_A and θ on the same batch of $\mathcal{D}^{rem}$. By using KL divergence as a loss function, we aim to minimize the difference between the distributions of the unlearned and agent models. Unlike BCE, which optimizes the model's output to match the ground-truth labels, the optimization target in KL is dynamic and determined by the output of an agent during iterations. Therefore, KL acts as a constraint during the unlearning process, correcting the unlearned model when the output distribution deviates significantly due to gradient ascent (GA). We represent $\mathcal{L}_{KL}$ as shown in equation 6.

$$\mathcal{L}_{KL} = \sum_{x^{rem} \in \mathcal{D}^{rem}} P_{\theta_A}(y^{rem} \mid x^{rem}) \log \left(\frac{P_{\theta_A}(y^{rem} \mid x^{rem})}{P_{\theta}(y^{rem} \mid x^{rem})} \right) \tag{6}$$

$P_{\theta_A}(y^{rem} \mid x^{rem})$ denotes the probability distribution of the agent model's output for the input x^{rem} from the dataset $\mathcal{D}^{rem}$. Similarly, $P_{\theta}(y^{rem} \mid x^{rem})$ represents the probability distribution of the unlearned model's output for the same input, which corresponds to $Softmax(\hat{y})$ in our work. Here we implement the forward KL, which helps the unlearned model align its output distribution with the agent. Since we aim to maintain the overall performance on $\mathcal{D}^{rem}$ during unlearning, thus, $\mathcal{L}_{KL}$ is updated on the unlearned model along with $\mathcal{L}_{GA}$ in each epoch.

3.4.3 Fine-tuning

KL reduces the damage caused by GA to the model, but we still need to ensure the model convergence after unlearning. Moreover, the KL divergence is weak at the first few epochs when GA has not dramatically changed the distribution of the model's output. Additionally, the losses of GA can increase infinitely, while KL is not powerful enough to restrict these changes. Therefore, we fine-tune (FT) the unlearned model on the $\mathcal{D}^{rem}$ simultaneous with GA and KL. In our experiments, we observed that the KL loss stabilizes after multiple epochs, which means the output distribution of the unlearned model is already similar to the agent model. However, the evaluation metrics on $\mathcal{D}^{rem}$ are still low and decreasing with the GA updating. This means that the model is still unable to fit the true labels. Even the unlearned model can finally converge due to the Adam optimizer with learning rate annealing. The damages have already been made. While FT can lead this convergence faster, reaching convergence before catastrophic forgetting. $\mathcal{L}_{FT}$ share the same loss function as in the original model training, which is shown in equation 7.

$$\mathcal{L}_{FT} = \mathcal{L}_{BCE}(x^{rem}, y^{rem}) \tag{7}$$

This differs from the method Yuanshun Yao et al. used on LLMs, since they implement a random mismatching loss $\mathcal{L}_{rand}$ instead of a fine-turning unlearned model. This can be seen as a negative sampling of GA. Fine-turning the unlearned model on $\mathcal{D}^{fgt}$ by additional random labels enhanced unlearning efficiency and smoothing the data in $\mathcal{D}^{fgt}$ to prevent the overfitting caused by GA. The main reason for the differences is the unlearning scenarios and evaluation methods used by the unlearning algorithms. First, their goal is not to unlearn the data in the model's training set but to unlearn additional harmful datasets. By unlearning this dataset as incremental training, they limit the ability of the model to generate harmful responses. Second, there is a large number of parameters in LLM, and the generation relies on a large

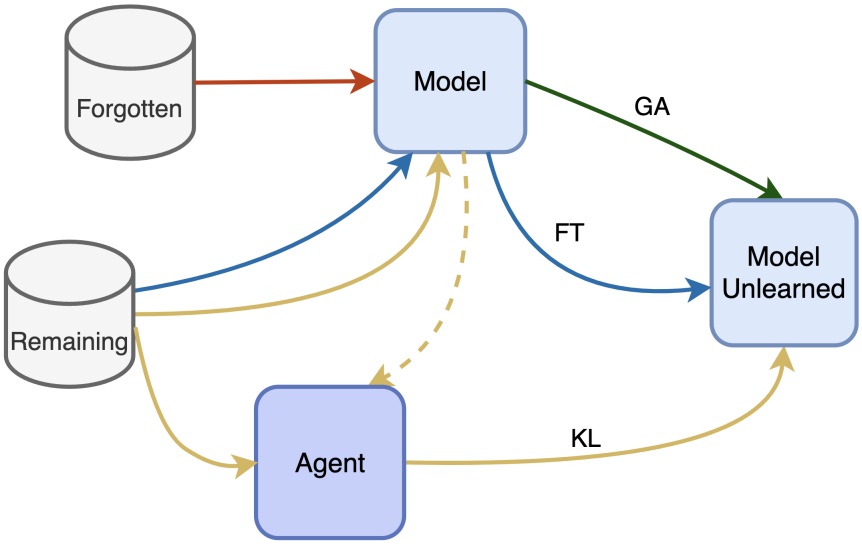


Figure 4: Unlearning pipeline

pre-trained BERT. These factors result in the negative effect of LLM caused by GA being lower than that in the sequential recommender system in our task. In addition, the generation performance is evaluated by fluency and diversity. The impact of GA on these metrics is not intuitive. However, in our experiments, sequential recommender systems are more sensitive than LLMs to parameter changes, especially when the model is trained on a small dataset like Beauty. In this scenario, a lightweight GA can cause devastating damage to the model. And introducing the random mislabeling only makes this worse. Thus, we chose FT to maintain model performance. However, there are two optimization targets in our algorithm for $\mathcal{D}^{rem}$. The model may conflict between fitting the true labels and aligning the output distributions. We search for the balance by adjusting the weights of these two loss functions in our experiments.

$$\theta_{t+1} = \theta_t - \eta_{GA}\nabla_{\theta_t}\mathcal{L}_{GA} - \eta_{KL}\nabla_{\theta_t}\mathcal{L}_{KL} - \eta_{FT}\nabla_{\theta_t}\mathcal{L}_{FT} \tag{8}$$

Finally, our unlearning algorithm can be represented as equation8, each loss function is controlled by an independent loss weight.

3.4.4 Unlearning procedure

Now, we describe the entire unlearning pipeline. As shown in figure4, we first initialize the data loaders for the to-be-forgotten set ($\mathcal{D}^{fgt}$) and the remaining set ($\mathcal{D}^{rem}$). Since $\mathcal{D}^{fgt}$ is smaller than $\mathcal{D}^{rem}$, we limit the number of iterations per epoch to the size of $\mathcal{D}^{fgt}$ to balance the sampling between the two datasets.

At each unlearning epoch, a batch from $\mathcal{D}^{fgt}$ and a batch from $\mathcal{D}^{rem}$ are fed into both the unlearned model and the agent model. The batch from $\mathcal{D}^{fgt}$ is passed only through the unlearned model to compute the gradient ascent loss ($\mathcal{L}_{GA}$) without using negative samples. Meanwhile, the batch from $\mathcal{D}^{rem}$ is processed by both the unlearned and agent models to calculate the KL divergence loss ($\mathcal{L}_{KL}$) without negative samples and the fine-tuning loss ($\mathcal{L}_{FT}$) with negative samples. The total loss is computed by combining these losses with corresponding weights. Moreover, we implement gradient clipping to prevent exploding gradients during

back-propagation. We set the unlearning batch size to 256 and the total unlearning rate to 0.0001. The specific weights of the loss functions are discussed in the following sections. We evaluate the unlearned model on $\mathcal{D}^{fgt}$ at every 10 epochs, early stop when the $HIT@10$ is not decreased in the last 30 epochs.

Annealing is used for faster model convergence during unlearning. As mentioned above, we fine-tune the unlearned model on $\mathcal{D}^{rem}$ simultaneously with applying GA. In our experiment, we observe that the model can reach convergence in final. For example, the unlearning of SASRec reaching the balance of unlearning and recommendation after convergence. However, in some cases, significant damage occurs before the model fully converges. Therefore, we introduce an additional annealing algorithm to gradually reduce the unlearning rate, leading to earlier termination of the unlearning process. It is important to note that this does not means that the FT lacks a role in helping the model converge, we prove that through an ablation study without annealing.

3.4.5 SISA

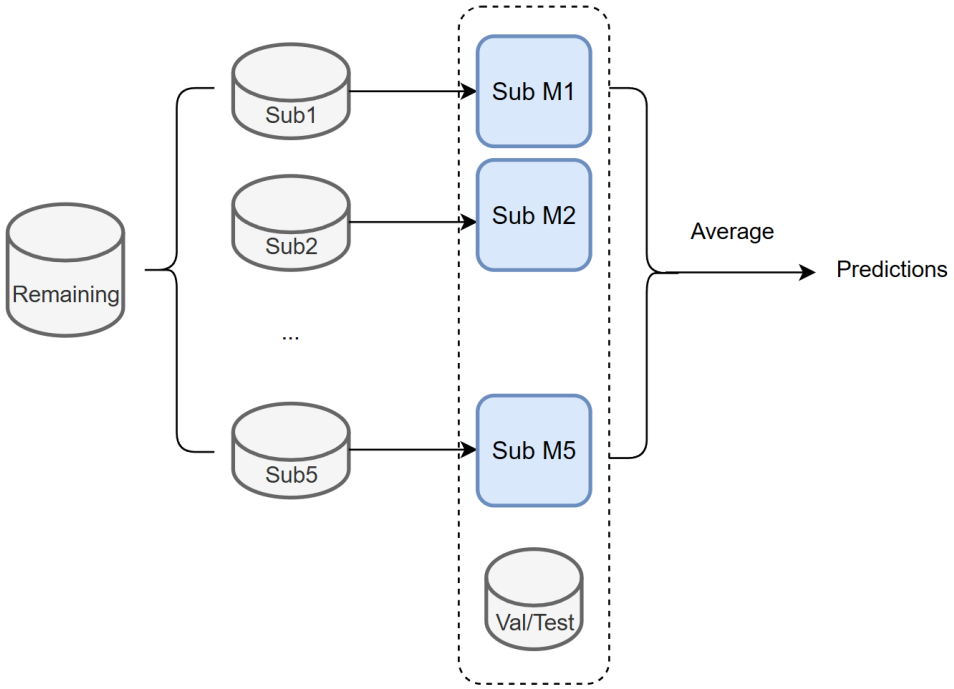


Figure 5: Training process of SISA

The training framework of sharded, isolated, sliced, and aggregated (SISA) proposed by Bourtoule et al. [2021]. As the representative method in exact unlearning, we implement it as the baseline. As shown in figure5, at the training stage, we randomly split $\mathcal{D}^{rem}$ into 5 shards and trained a sub-model on each shard. In the prediction stage, the validation and test sequences are passed through all sub-models, generating prediction scores for each. The aggregation layer is not involved in the training process but aggregates the outputs of the sub-models by averaging them, contributing to the final recommendation. Since the aggregation only requires the predictions from the sub-models, it is unnecessary to modify the structure of the model

like SASRec but use it to train directly on the data shard and save the best checkpoints as sub-models.

An alternative approach is to design a trainable aggregation model, which trains on additional datasets to assign different weights to the sub-models. However, this aggregation model would also need to be retrained when unlearning requests. Moreover, the research of Bourtoule et al. [2021] has shown that the performance improvement from this approach is limited. Thus, we implement the basic version of SISA in our work. Here we simulate the worst scenario, in which the to-be-forgotten items are distributed across all datasets. We retrain all 5 sub-models for each unlearning request.

3.5 Evaluation design

In this section, we discuss how to evaluate the effectiveness of unlearning and recommend performance. These evaluation metrics are used for all the models, datasets, and baselines in our experiments.

Recommendation performance: A common evaluation method for the next-item recommendation task is leaving one out. Different from the traditional method, where datasets are split into training, validation, and test sets in specified ratios. In this method, a part of the samples of each sequence is taken as the validation and test samples. In our task, as shown in figure 2, the last two items from a sequence are extracted. The penultimate item is used as the validation sample, and the last is used as the test sample. In the prediction phase, the model recommends the penultimate or the last item based on the user's sequence in the training set. The evaluation metrics we use are $NDCG@[10, 20]$ (Normalized Discounted Cumulative Gain) and $Recall@[10, 20]$.

$$\text{DCG@k} = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} \tag{9}$$

$$\text{IDCG@k} = \sum_{i=1}^{|REL_k|} \frac{rel_i - 1}{\log_2(i+1)} \tag{10}$$

$$\text{NDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}} \tag{11}$$

In equation9, the DCG is a method for scoring a recommendation list. The score is calculated by the relevance of the i-th item and its position in the list. The value of rel_i is 0 or 1 for relevant or irrelevant. The $\frac{1}{\log_2(i+1)}$ decreases as i increases, meaning that lower-ranked items contribute less to the total score. $|REL_k|$ is the number of relevant items in the top k of the recommendation list. Therefore, $IDCG$ in equation10 represents the ideal situation of the recommendation list, where the ranked item in the list is the same as that in real labels. $NDCG$ in equation11 is the ratio of DCG to $IDCG$, representing how the recommendation list is close to the ideal results.

$$\text{Recall@k} = \frac{|REL_k \cap Rec_k|}{|REL|} \tag{12}$$

In equation 12, $|REL_k \cap Rec_k|$ represents the number of recommended items in the top k positions that are truly relevant.$|REL|$ is the total number of the relevant items. Thus, we use $Recall$ to measure the model's ability to recommend the relevant items and $NDCG$ to measure the recommendation qualities.

Compare this to Kang and McAuley [2018] in SASRec, who only evaluate a maximum of 10000 sequences from the test set. In contrast, we use a more comprehensive evaluation, where the model is evaluated on the complete testing set. Moreover, as mentioned in the previous section, we tend to keep all sequences, even the ones longer than the maximum length, by splitting them into sub-sequences rather than truncating them. Resulting in our testing set containing more samples than originally. Thus, the model's performance with our metric is slightly lower than the original authors' report. Because ranking the correct item from a larger set is more challenging. We implement uniform training and evaluation functions for all models, ensuring that our experiments are comparable.

Unlearning efficiency: Evaluating the unlearning efficiency is a challenge in the approximate unlearning field. Foster J et al. propose the golden standard of unlearning is to achieve the same performance as the entire retraining model Foster et al. [2024]. They point out that over-unlearning can lead to information leakage. The attackers can extract information that the model trying to avoid. Yao et al. [2024] define the decrease in fluency and increased diversity of the model's output for harmful prompts as representing effective unlearning in the LLM. The membership inference attack (MIA) evaluates the effectiveness of unlearning by training a neural network on the outputs of an unlearned model using data that the model has seen and not seen. Unlearning is seen as effective if the neural network cannot distinguish the unlearned data from a part of the unlearned model's training set based on its outputs. This is probably the best method for evaluating the unlearning task. However, our $\mathcal{D}^{fgt}$ consists of the sub-sequences extracted from the sequences of to-be-forgotten items, which is challenging in our task.

$$\text{HIT@k} = \frac{1}{|\mathcal{D}^{fgt}|} \sum_{n=1}^{|\mathcal{D}^{fgt}|} \mathbb{I}(Rec_i \leq k) \tag{13}$$

Therefore, we implement the $HIT@k$ metric. As equation13, $|\mathcal{D}^{fgt}|$ is the number of users in $\mathcal{D}^{fgt}$. Thus, we calculated the rate of the recommended items ranked in the top k position. The lower the metric, the lower the probability of these forgotten items being recommended to users, representing better unlearning efficiency. However, there are no standards in the evaluation of approximate unlearning. Researchers must balance the model performance and the unlearning degree by themselves. In the item-level unlearning in this project, we think $HIT@k$ is a reasonable method to evaluate unlearning.

4 Experimental setup and results

In the previous chapters, we introduced our data preprocessing strategy and the concept of our unlearning algorithm. Here, we discuss the details of the implementation of our experiments.

4.1 Experiment settings

Here, we introduce the setting of our experiments. We apply our unlearning algorithm to GRU4Rec, SASRec, and BERT4Rec on the Amazon-beauty, games, and Steam datasets. The dataset is divided into the raw set ($\mathcal{D}^{raw}$), the to-be-forgotten set ($\mathcal{D}^{fgt}$), and the remaining set ($\mathcal{D}^{rem}$).

Parameter Category	Hyperparameter	Value/Description
Data preprocessing		
	Maximum sequence length (n)	10
	Unlearning rate	10%
	SISA Shard Number	5
Model common parameters		
	Optimizer	Adam
	Loss Function	Binary cross-entropy
	Learning Rate	0.003
	Hidden units	64
Model-Specific Parameters		
SASRec	Attention blocks	2
	Attention heads	1
	Dropout	0.5
BERT4Rec	Attention blocks	2
	Attention heads	4
	Dropout	0.2
	Mask probability	0.3
Loss weights		
	η_{GA}	$0.1 \sim 1$
	η_{FT}	$0.1 \sim 1$
	η_{KL}	$0.1 \sim 1$
	Unlearning Rate	0.0001

Table 3: Hyper-parameters settings

Table3 shows the parameter settings of our experiment. The length of sequences is limited to a maximum of 10 items. The unlearning rate is decided to be 10% of users in $\mathcal{D}^{raw}$. Adam optimizer updated all the models with BCE loss, with a learning rate of 0.003.

Attention blocks represent the number of stacks of the transformer layers, which consist of a multi-head attention layer and a forward layer. The parameters are passed in order of

the attention blocks, the later blocks capture the further self-attention from the output of the previous block. Since the maximum sequence length is set to 10, there is no need for deep networks to calculate the self-attentions between the items. Thus, we allocate 2 attention blocks for both SASRec and BERT4Rec.

Attention heads are the number of sub-spaces in the attention block. Each attention head calculates self-attention independently, and the results of the heads are concatenated. This designation allows the model to capture the items' relationships in different aspects. In SASRec, we use single-head attention with 64 dimensional, following the authors' conclusion in their experiments. In BERT4Rec, we choose to use the two-head attention with 32 dimensional. This design ensures the hidden units are 64 for both SASRec and BERT4Rec.

Dropout is implemented after each layer, which randomly drops some parameters during the training process. This design is for training different neurons at each epoch, helps to improve the robustness of the model, and alleviates the over-fitting. The dropout strategy is used after each embedding, attention, and feed-forward layer in SASRec with a 0.5 dropout rate, 0.2 for BERT4Rec.

The Mask probability of BERT4Rec is set to 0.3, in which 30% of items are masked in a sequence as the prediction targets in the training process. We did not use dropout GRU4Rec to prevent under-fitting. We save the checkpoints every 10 epochs, and the model is considered converged when the $NDCG@10$ metric on the validation set does not improve within 50 epochs. The optimal checkpoint is selected as the final model. We train the SISA with 5 shards on $\mathcal{D}^{rem}$ for each model as the baseline, representing an exact unlearning method based on retraining. In SRUGA, the strength of each loss function is controlled by η_{GA}, η_{KL}, and η_{FT}.

4.2 Baseline results

Figure 6 illustrates the $NDCG@10$ of each model on $\mathcal{D}^{raw}$ of three datasets. SASRec performed best on all datasets, followed by BERT4Rec and GRU4Rec. In general, the performance of BERT4Rec outperformed SASRec. However, the BCE loss and negative sampling strategy in our framework may lead to performance degradation. For GRU4Rec, the RNN architecture is weaker than the transformer-based architecture models for capturing relationships in long or sparse sequences, leading to the worst performance. On the other hand, the sparsity of the datasets also impacts the unlearning efficiency. As shown in table 2, all models performed best on the dense dataset Steam, achieving similar and high recommendation performance. In contrast, in the sparse dataset Beauty, each item only interacted 6.478 times on average, resulting in the worst performance.

4.3 Unlearning effectiveness

In this section, we evaluate the effectiveness of SRUGA in unlearning. Figure7 presents the $Hit@10$ performance of three models on the Beauty $\mathcal{D}^{fgt}$ dataset after unlearning random interactions from 10% of users. The loss weights of SRUGA are set to $\mathcal{L}_{GA} = 0.5$, $\mathcal{L}_{FT} = 0.5$, and $\mathcal{L}_{KL} = 1$, with other settings aligned as the descriptions in table3.

Table 4: Recommendation performance and unlearning effectiveness comparison after unlearning 10% of data in each dataset. N is the abridge of $NDCG$, R is the abridge of $Recall$, and H is the abridge of Hit.

Beauty		$\mathcal{D}^{raw}$				$\mathcal{D}^{rem}$				$\mathcal{D}^{fgt}$			
		N@10	N@20	R@10	R@20	N@10	N@20	R@10	R@20	H@1	H@5	H@10	H@20
GRU4Rec	**Original**	0.2558	0.2878	0.3998	0.5265	0.2559	0.2879	0.399	0.5259	0.6245	0.9486	0.9822	0.996
	Retrain	0.2391	0.2711	0.3867	0.5139	0.2524	0.284	0.3922	0.5175	0.249	0.6304	0.7016	0.7767
	SISA	**0.2361**	**0.2699**	**0.4076**	**0.5417**	**0.2497**	**0.2821**	**0.4187**	**0.547**	**0.074**	0.2227	0.3282	0.4608
	SRUGA	0.2019	0.2337	0.3285	0.4548	0.2013	0.2332	0.3274	0.4542	0.0802	**0.1862**	**0.2411**	**0.315**
SASRec	**Original**	0.3209	0.3505	0.474	0.5916	0.3203	0.3494	0.4746	0.5898	0.7352	0.9822	0.9941	0.998
	Retrain	0.299	0.3284	0.4564	0.5732	0.3089	0.3382	0.461	0.5772	0.4328	0.7688	0.8142	0.8419
	SISA	0.2151	0.2403	0.3386	0.4391	0.2146	0.2402	0.3366	0.4383	**0.0796**	**0.2299**	0.3299	0.4445
	SRUGA	**0.2436**	**0.2713**	**0.3705**	**0.4805**	**0.243**	**0.271**	**0.368**	**0.479**	0.117	0.232	**0.2749**	**0.3177**
BERT4Rec	**Original**	0.2932	0.3237	0.4612	0.5823	0.3007	0.3306	0.4685	0.587	0.3123	0.7727	0.8636	0.9348
	Retrain	0.0364	0.054	0.0769	0.1475	0.3049	0.333	0.4594	0.5707	0.2016	0.6818	0.7549	0.8063
	SISA	0.2447	0.2769	0.4054	0.533	0.2391	0.2696	0.4085	0.5296	**0.0618**	**0.1788**	**0.2754**	**0.4007**
	SRUGA	**0.2613**	**0.291**	**0.421**	**0.5386**	**0.2753**	**0.3049**	**0.4322**	**0.5497**	0.0782	0.271	0.4184	0.6029

Steam		$\mathcal{D}^{raw}$				$\mathcal{D}^{rem}$				$\mathcal{D}^{fgt}$			
		N@10	N@20	R@10	R@20	N@10	N@20	R@10	R@20	H@1	H@5	H@10	H@20
GRU4Rec	**Original**	0.5573	0.5819	0.8168	0.913	0.5565	0.5811	0.8162	0.9126	0.3317	0.6658	0.8467	0.9347
	Retrain	0.5606	0.5848	0.8187	0.9135	0.559	0.5832	0.8177	0.9122	0.3216	0.6884	0.8241	0.9196
	SISA	**0.5218**	**0.549**	**0.7818**	**0.8887**	**0.5205**	**0.5479**	**0.7807**	**0.8882**	0.2494	0.5881	0.7428	0.8631
	SRUGA	0.4946	0.5239	0.7544	0.8697	0.4905	0.5202	0.7499	0.8667	**0.1359**	**0.4076**	**0.5598**	**0.6957**
SASRec	**Original**	0.5591	0.5839	0.8197	0.9168	0.5588	0.5835	0.8192	0.916	0.3241	0.6809	0.8166	0.9196
	Retrain	0.556	0.5807	0.8169	0.9139	0.5565	0.5814	0.8164	0.914	0.299	0.6884	0.8291	0.9221
	SISA	**0.5348**	**0.5592**	**0.7935**	**0.8891**	**0.5345**	**0.5591**	**0.7921**	**0.8887**	0.2534	0.5979	0.751	0.8723
	SRUGA	0.4751	0.5082	0.732	0.8623	0.4712	0.5047	0.7282	0.8597	**0.1522**	**0.4402**	**0.6359**	**0.7554**
BERT4Rec	**Original**	0.5571	0.5816	0.821	0.917	0.5557	0.5803	0.8201	0.9167	0.2764	0.6256	0.7714	0.8995
	Retrain	0.5756	0.5981	0.8329	0.921	0.5746	0.5971	0.8322	0.9205	0.2663	0.6432	0.7889	0.902
	SISA	0.028	0.0346	0.0516	0.0782	**0.5274**	**0.5541**	**0.7911**	**0.8961**	0.2341	0.5712	0.7311	0.8617
	SRUGA	**0.5338**	**0.5608**	**0.7953**	**0.9011**	0.4982	0.528	0.7623	0.8792	**0.0978**	**0.3152**	**0.4891**	**0.6848**

Games		$\mathcal{D}^{raw}$				$\mathcal{D}^{rem}$				$\mathcal{D}^{fgt}$			
		N@10	N@20	R@10	R@20	N@10	N@20	R@10	R@20	H@1	H@5	H@10	H@20
GRU4Rec	**Original**	0.3931	0.4222	0.5917	0.7072	0.3916	0.4211	0.5907	0.7073	0.5608	0.9365	0.9683	0.9868
	Retrain	0.3845	0.4142	0.5897	0.7071	0.3971	0.4264	0.5952	0.7111	0.2487	0.5529	0.6772	0.8042
	SISA	**0.2765**	**0.3135**	**0.4747**	**0.6213**	**0.2795**	**0.3171**	**0.4757**	**0.624**	**0.0732**	**0.2264**	0.334	0.4708
	SRUGA	0.2754	0.3102	0.4503	0.5879	0.2744	0.3087	0.4494	0.5854	0.1085	0.2374	**0.3182**	**0.4161**
SASRec	**Original**	0.5065	0.5296	0.7145	0.8055	0.5066	0.5296	0.7149	0.8055	0.5608	0.9127	0.9709	0.9974
	Retrain	0.4889	0.5133	0.6998	0.796	0.498	0.5222	0.7031	0.7985	0.3122	0.6296	0.7646	0.873
	SISA	0.2632	0.2905	0.4279	0.5362	0.2633	0.2906	0.4284	0.5369	**0.0721**	**0.2247**	**0.3396**	0.4706
	SRUGA	**0.3713**	**0.3972**	**0.5539**	**0.6565**	**0.3705**	**0.3967**	**0.5529**	**0.6562**	0.1332	0.2875	0.3674	**0.448**
BERT4Rec	**Original**	0.4933	0.5161	0.6987	0.7891	0.4785	0.5029	0.6844	0.781	0.4524	0.8677	0.9603	0.9868
	Retrain	0.1657	0.202	0.3075	0.4522	0.4738	0.4984	0.6822	0.7791	0.1984	0.6058	0.7169	0.8413
	SISA	**0.2744**	**0.3118**	**0.4667**	**0.6143**	0.2738	0.3108	0.4695	0.6157	0.0718	0.2143	0.3272	0.4698
	SRUGA	0.254	0.2914	0.4385	0.5867	**0.2829**	**0.3196**	**0.4759**	**0.6212**	**0.0427**	**0.1621**	**0.2726**	**0.4461**

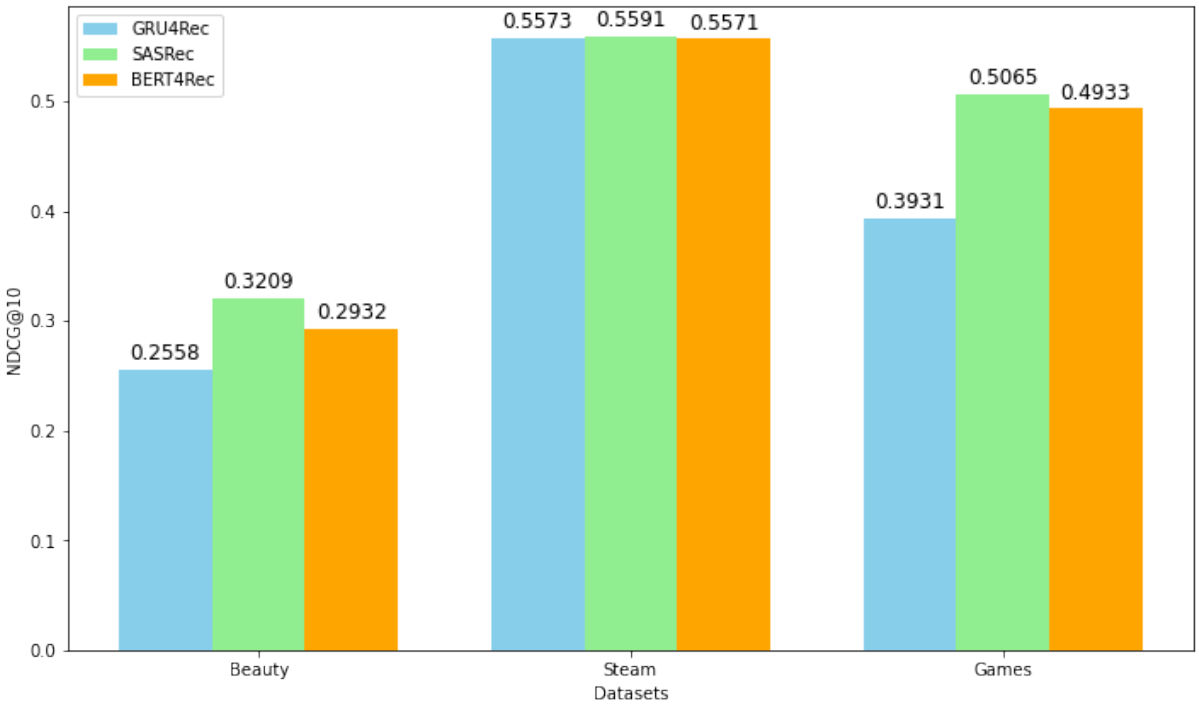


Figure 6: The results of $NDCG@10$ of the GRU4Rec, SASRec and BERT4Rec trained on Beauty dataset.

All the original models recommend the forgotten items well, with $Hit@10$ close to 1. After retraining, the $Hit@10$ for GRU4Rec, SASRec, and BERT4Rec dropped to 71%, 81%, and 84% of their original, respectively. We observe that even though these forgotten items have been removed from the training set, the models still have a big chance to recommend them. This is because, in sequential recommender systems, users who interact with the same items often share similar interests. The models can make accurate recommendations based on information from similar users.

SISA, the exact unlearning algorithm, outperforms the entire retraining methods on all models. Since there is not direct information exchange between sub-models in SISA. The system cannot completely capture the dependencies across different users' interests. Each sub-model makes predictions based only on its shard of data. Some sub-models trained on interactions that are similar to the forgotten sequences may provide higher recommendation scores. However, the final score is reduced by averaging the generalized predictions from other sub-models, thereby lowering the chance of forgotten items being recommended. This phenomenon is also reflected in its overall recommendation performance, which we discuss in the following sections.

SRUGA, the approximate unlearning method we designed, achieved the best results in GRU4Rec and SASRec. The $Hit@10$ decrease to 24%, 27%, and 41% of original models respectively. Thus, we prove that using the GA to actively inhibit the model's parameters on specific items can achieve effective unlearning. Besides, we observed that the decline of $Hit@10$ in the unlearning process of BERT4Rec is slow. This could result in the structure of our sampling strategy, in which only the final item in the forgotten sequence is masked as the unlearned target. BERT4Rec cannot use the contextual information on both sides to predict

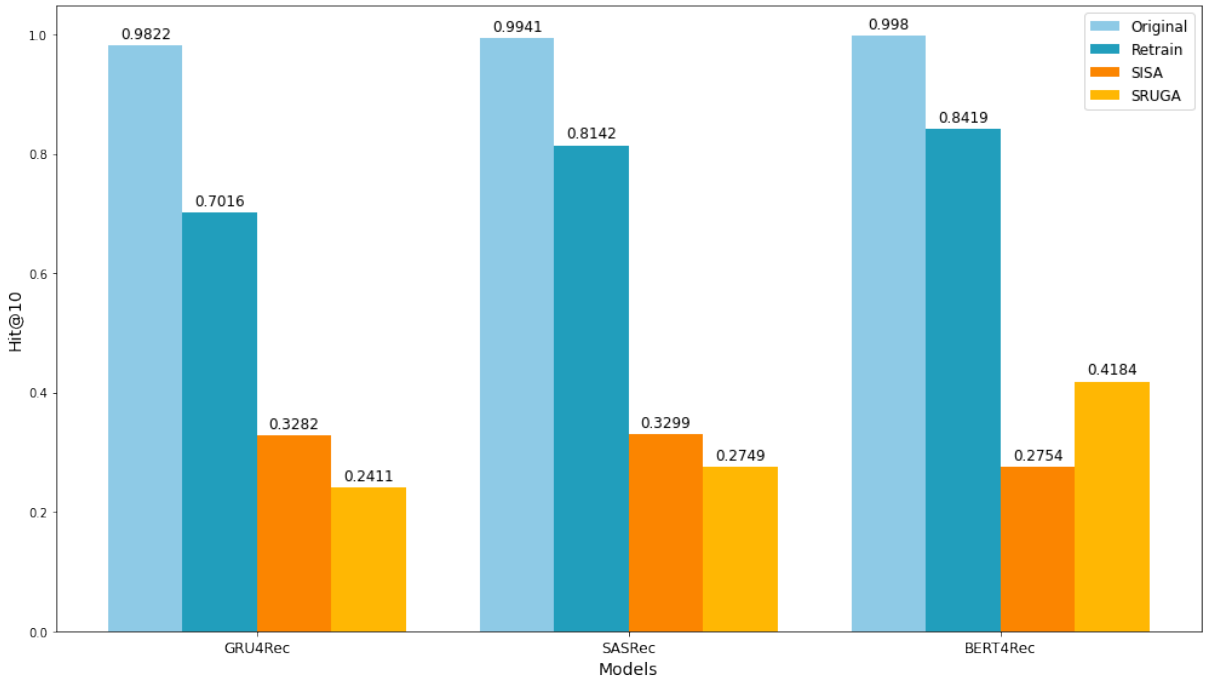


Figure 7: The results of $Hit@10$ for GRU4Rec, SASRec and BERT4Rec on Beauty dataset after applying different unlearning algorithms.

the item, resulting in lower loss when applying gradient ascent. In addition, we also tried to mask all the to be forgotten items in a sequence, unlearning multiple items at once. The results show a heavy impact on the recommendation performance of BERT4Rec, with catastrophic forgetting after a small number of epochs.

4.4 Recommendation performance

In this section, we first analyze the impact of retraining, SISA, and SRUGA on recommendation performance. Then we discussed the pros and cons of each method based on the results of all experiments. The hyper-parameters we used here are the same as the precious section.

Figure8 shows the $Recall@10$ of three models on $\mathcal{D}^{rem}$ of Beauty after unlearning random interactions from 10% of users. The retraining models maintain the recommendation performance, achieving the close results of the original. The performance of SRUGA aligns with our expectations. The negative impact of GA cannot be fully eliminated, resulting in the $Recall@10$ decrease to 82%, 77%, and 92% before unlearning. However, these results still outperform SISA in SASRec and BERT4Rec. Table 4 shows all the experimental results. Undoubtedly, the entire retaining method, which can use the complete information from the $\mathcal{D}^{rem}$, achieves the best recommendation performance. However, its unlearning effectiveness is weaker than that of SISA and SRUGA.

We observe that SRUGA is more stable across different datasets. In the sparse dataset Beauty, SRUGA reduces Hit@10 by 30% compared to the original models while maintaining an average Recall@10 of 85% on $\mathcal{D}^{rem}$. In contrast, SISA drops the $Recall@10$ to 0.3366

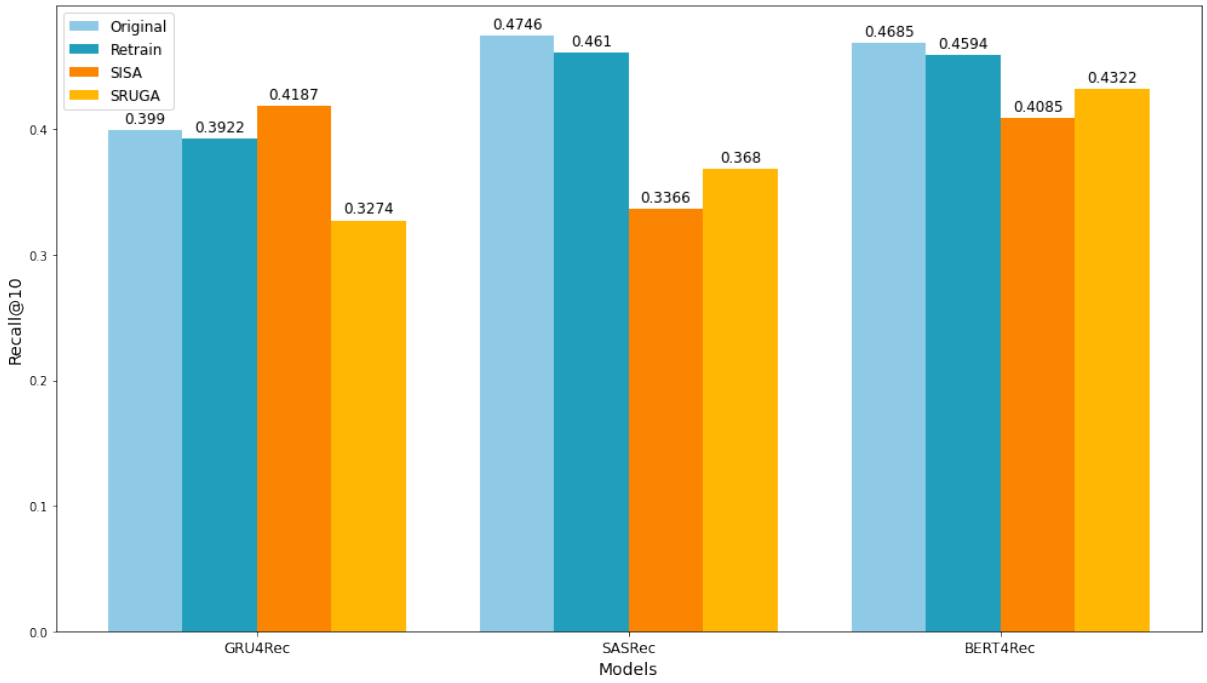


Figure 8: The results of $Recall@10$ for GRU4Rec, SASRec and BERT4Rec on Beauty dataset after applying different unlearning algorithms.

because each sub-model trained under SISA learns insufficient information after the dataset is divided into multiple shards. The same results are also shown in Games, another dense dataset. The $Recall@10$ of SASRec decreased to 0.4284 from 0.667 after SISA, only 64% of the original model. At the same time, SRUGA reached 0.5529, maintaining 82% of the recommended performance. In the dense dataset Steam, which has 334,730 sequences but only 13,047 items. Each item is interacted with 282.5 times on average, meaning users share similar preferences. In this scenario, the retraining-based methods almost lost their unlearning ability. In SASRec, the $HIT@10$ of Original, Retrain, and SISA are 0.7738, 0.7623, and 0.751, respectively. Meanwhile, SRUGA drops it to 0.6359, although the recommended performance decreases, which is still maintained at 90% on average. Overall, SRUGA balances unlearning and recommendation performance, maintaining robustness across various models and datasets. In contrast, the performance of SISA depends on the dataset and model architecture, showing a serious recommendation decline in sparse datasets and insufficient unlearning on dense datasets.

In summary, SRUGA achieves comparable results to SISA. It is also worth noting that the weight of losses $\mathcal{L}_{GA}$, $\mathcal{L}_{KL}$, and $\mathcal{L}_{FT}$ are optimized on SASRec only, and we follow the same settings in all experiments. Although it performs well in most cases, it still needs to be improved. For example, in Games datasets, the recommendation performance of GRU4Rec and BERT4Rec are seriously declining after applying SRUGA.

4.5 Unlearning efficiency

Table5 presents the running time of the three unlearning methods. All the experiments are tested on the Nvidia RTX 3080-10G. SRUGA demonstrated the best performance in most

Table 5: The comparison of unlearning efficiency after unlearning 10% of data in each dataset, measured in minutes.

Minutes	Beauty			Games			Steam		
	GRU4Rec	SASRec	BERT4Rec	GRU4Rec	SASRec	BERT4Rec	GRU4Rec	SASRec	BERT4Rec
Retrain	160.7	420.03	111.81	101.31	196.65	175.88	202.97	676.04	701.64
SISA	30.22	43.93	35.17	**11.44**	**15.76**	22.50	124.40	358.91	309.04
SRUGA	**24.84**	**5.01**	**12.19**	15.5	56.71	**7.63**	**7.4**	**10.59**	**44.35**

scenarios. SISA divides the dataset into multiple shards, allowing each sub-model to converge more rapidly. Therefore, it outperforms the entire retraining approach as an exact-unlearning algorithm. SRUGA, as the approximate unlearning algorithm, takes less time to retrain the model. For example, on the Beauty dataset, retraining SASRec takes 420.03 minutes, SISA requires 43.93 minutes, while SRUGA takes 5.01 minutes only, representing 80 times faster over retraining and 8 times over SISA. More advantages are shown on larger and denser datasets, where SRUGA outperforms the SISA and Retrain in all the results. However, there are also some disadvantages shown on GRU4Rec, in which SRUGA spends more time than SISA unlearning in Games. This is related to our early stop strategy. In some experiments, SRUGA quickly reduced the $Hit@10$ metrics, while the evaluation metrics on the $\mathcal{D}^{rem}$ were still decreasing at a very slow rate. Indicating that the model did not achieve convergence, which take extra time for this process. A useful approach is to actively reduce the weights of GA, which stops the unlearning after some epochs. Then, push the model to the convergence by applying FT only. However, this method needs additional parameters turning for each model. Unfortunately, we did not have enough time for that since our experiments covered three models across three datasets. Therefore, in this paper, we aim to verify the effectiveness of SRUGA, i.e., whether it can achieve unlearning in different environments and perform as well as SISA.

4.6 Case study

Now, we illustrate the unlearning effect using a case study of SASRec on the Beauty dataset. As shown in table 6, for user 293, the interactions are represented by a raw sequence of 10 items, with items 12536 and 55453 as the validation and test samples. The forgotten items in this case are 8742, 54912, and 55044, represented by three forgotten sequences. We compare SASRec's top 10 recommendations before and after applying SRUGA. If the forgotten items appear in the recommendation list, it means the model can still recommend them. Otherwise, the unlearning is successful.

Before applying SRUGA, SASRec accurately predicted the forgotten items at the highest ranking. Item 8742 and item 54912 are recommended at the top of the list. Representing that the model has well learned the sequential relationships during training. After applying SRUGA, none of the forgotten items are prioritized in the top 10 of SASRec's recommendations. The unlearning algorithm effectively prevents the model from recommending them.

Then, we tested the recommendation performance on the remaining sequence. SASRec ranks the test sample at $4th$ of the list, while SRUGA ranks it at $2nd$. This demonstrates that SASRec's performance declines when part of the contextual information in the sequence

User id	Raw sequence	Val & Test
293	18426, 29351, **8742**, 54914, **54912**, 32829, **55044**, 55045, 55040, 55350	12536, 55453
Models	**Forgotten sequences**	**Forgotten item**
	0, 0, 0, 0, 0, 0, 0, 0, 18426, 29351	8742
	0, 0, 0, 0, 0, 0, 0, 18426, 29351, 54914	54912
	0, 0, 0, 0, 0, 0, 18426, 29351, 54914, 32829	55044
	Top 10 predictions	**Ranking**
SASRec	**8742**, 38572, 9938, 18819, 32361, 3246, 38035, 1637, 10651, 5330	1
	54912, 52219, 37311, 29458, 54789, 8601, 10347, 26846, 7293, 25521	1
	52081, **55044**, 49835, 55758, 17314, 42063, 48543, 24144, 48539, 40041	2
SRUGA	4755, 17003, 11402, 49837, 44009, 49449, 2433, 10716, 15565, 30040	> 10
	55980, 45230, 36603, 13879, 25790, 3924, 6134, 32862, 44151, 55142	> 10
	56005, 49833, 55184, 44378, 28961, 36483, 12165, 53370, 11989, 53889	> 10
Models	**Remain sequences**	**Test**
	0, 0, 18426, 29351, 54914, 32829, 55045, 55040, 55350, 12536	55453
	Top 10 predictions	**Ranking**
SASRec	32338, 10824, 16076, **55453**, 10484, 2014, 22484, 46899, 37253, 27831	4
SRUGA	48671, **55453**, 53884, 44149, 89, 36214, 6053, 4924, 22880, 22406	2

Table 6: A case study on the unlearning process of SASRec using the Beauty dataset.

is missing. In contrast, SRUGA benefited from fine-tuning on $\mathcal{D}^{rem}$, learning new information from the modified sequences.

4.7 Ablation studies

In this section, we discuss the effect of the components of SRUGA. First, we did an ablation study to compare the performance of GA, KL, and FT in different combinations. Next, we analyze the impact of varying the weights of η_{GA}. Finally, we illustrate how KL and FT reduce the negative effect of GA and lead the model to convergence. It is important to note that in this section, we do not use the annealing algorithm to accelerate the termination of unlearning.

Figure9 shows the ablation results of SASRec trained on Beauty. We compare the $Recall@10$ on $\mathcal{D}^{rem}$ and $Hit@10$ on $\mathcal{D}^{fgt}$ with specific SRUGA components. Obviously, applying GA without any restrictions can destroy SASRec's recommendation abilities. The $Hit@10$ drops to the lowest, reaching 0.0318, while $Recall@10$ decreases to 0.0904, leading to catastrophic forgetting. Furthermore, the performance of FT and FT+KL is almost identical to the original SASRec, which means the unlearning has not happened. Although the $Hit@10$ metric also decreases when applying KL alone, the decline in $Recall@10$ is more serious. Thus, this decline results from the model's under-fitting rather than unlearning behavior. Using KL alone can

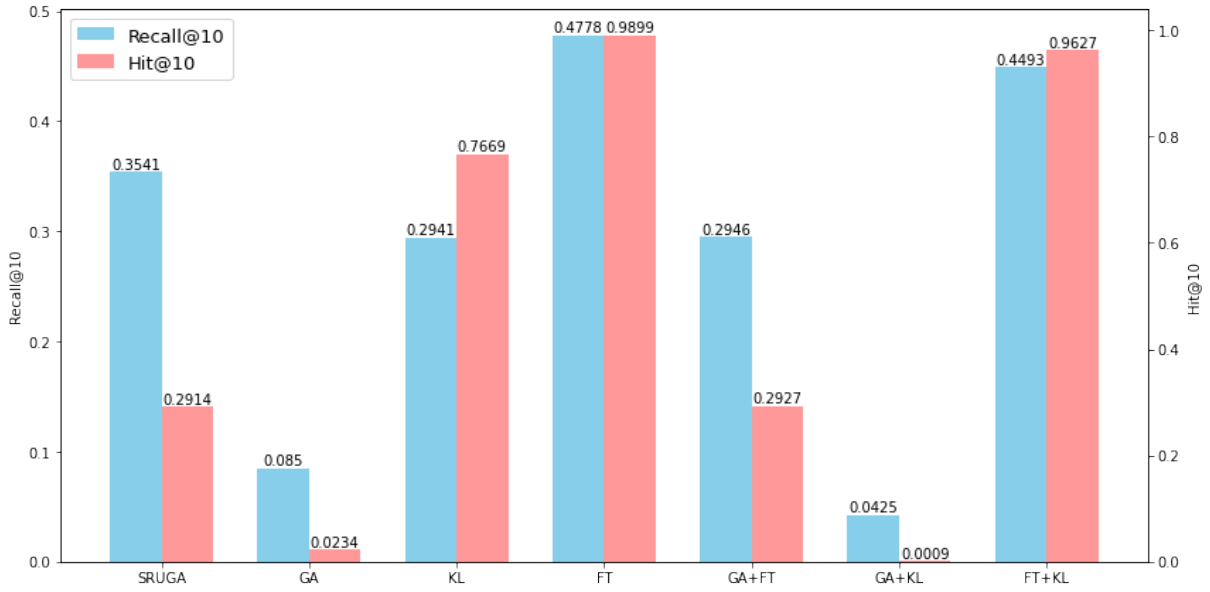


Figure 9: Ablation Study: Comparison of $Recall@10$ and $Hit@10$ for different combination of the components from SRUGA.

force the unlearned model to fit its output distribution to the agent model, which is dynamic. This unstable supervised learning damages the model's overall performance. Therefore, as we mentioned in previous sections, $\mathcal{L}_{KL}$ is designed to prevent heavy distribution changes of the unlearned model from $\mathcal{L}_{GA}$ rather than being the ground truth labels. For the same reasons, GA+KI shows similar results of using GA alone. Moreover, FT plays an important role in maintaining the recommendation performance, with the results of GA+FT only behind the complete SRUGA.

Figure10 shows how the $Recall@10$ and $Hit@10$ metrics changes with the increasing η_{GA}, η_{KL}, and η_{FT}. The experiment is implemented on a SASRec that trains on the Beauty dataset, with the η_{FT} and η_{KL} fixed to 0.5 and 1, respectively. It is clear that both $Recall@10$ and $Hit@10$ decrease with the η_{GA} increasing. When the η_{GA} is lower than 0.5, the model maintains enough recommendation ability, with higher $Recall@10$ than that of SISA. After that, it shows heavy degradation. Besides, SASRec outperforms the baseline when $\eta_{GA} = 0.4$, achieving lower $Hit@10$ while higher $Recall@10$. As shown in figure11a, $Recall@10$ and $Hit@10$ increase with the η_{FT} increasing. However, the impact of η_{KL} is not straightforward, as described in figure11b. The metrics show a trend of first dropping and then increasing when η_{KL} is lower than 0.6. After that, despite there are slight fluctuations shown on $Recall@10$, the overall trend stabilizes. This means the difference between the output distribution of the unlearned and agent models has been limited to a small range.

Figure12 shows the loss increase on $\mathcal{D}^{rem}$ during the unlearning process. The unlearning epochs are set to 1000 without early stopping. We compare the curve of the combination of GA with other components. It is clear that both KL and FT reduced the rate of loss increase. Moreover, the curve is smoother when applying KL, meaning that the oscillations caused by GA are reduced. By comparing the GA+KL with GA+KL+FT, FT prevents the model from the negative impact of GA and leads the model closer to convergence. It is worth noting that

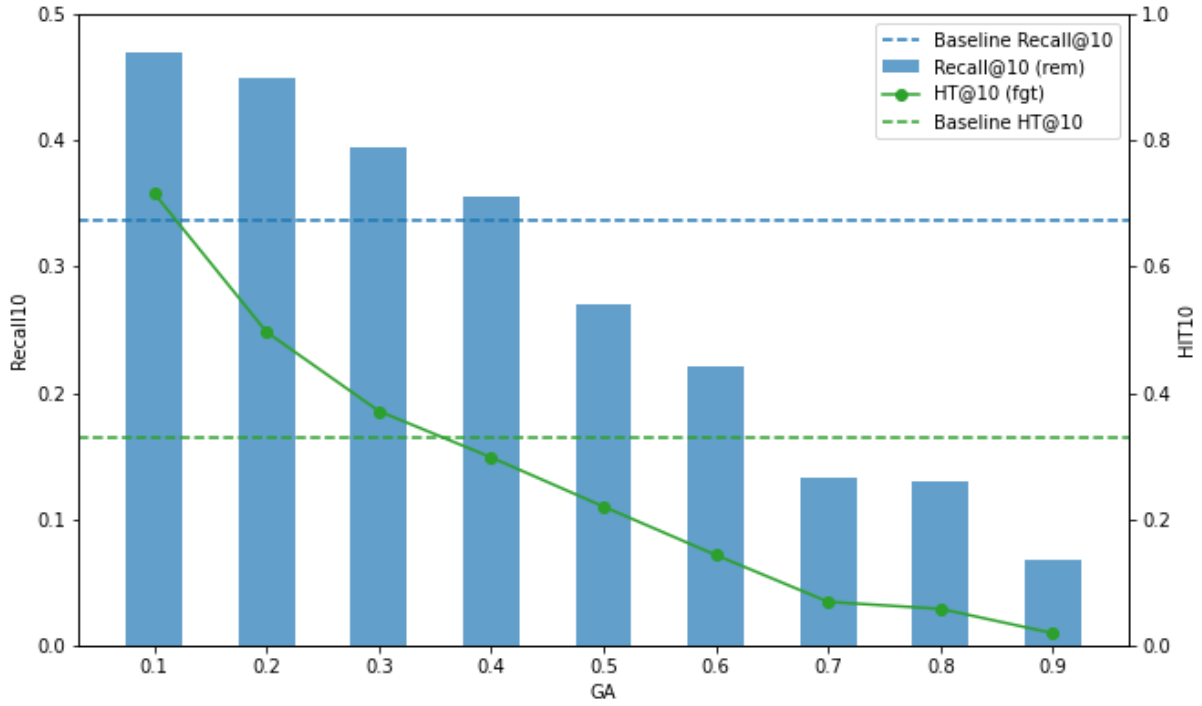


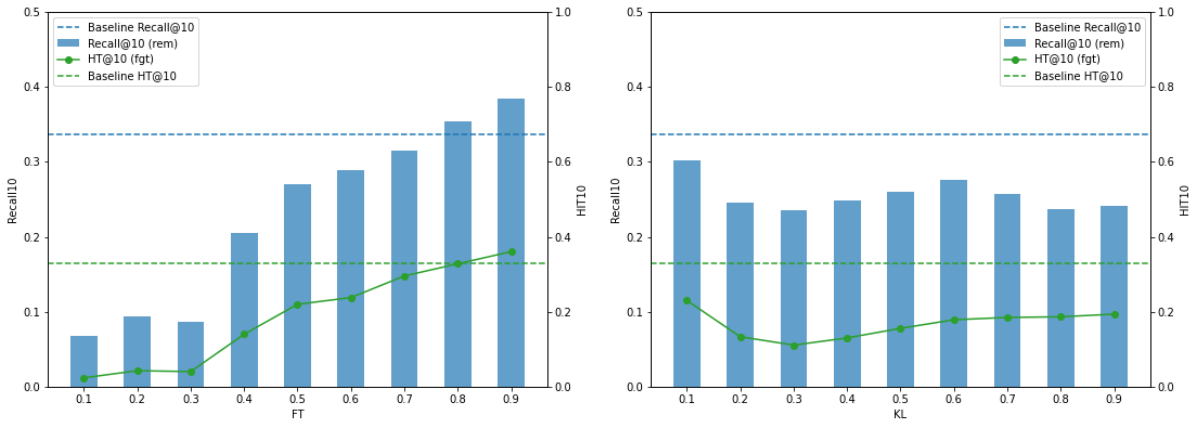
Figure 10: The results of recommendation performance and unlearning effectiveness under different η_{GA}. The η_{FT} and η_{KL} fixed to 0.5 and 1. The dashed line represents the results of SISA on SASRec trained on Beauty.

the loss on the $\mathcal{D}^{rem}$ does not directly indicate the model's recommendation performance in our unlearning algorithm. We observe that although GA+KL achieves the second-lowest loss, it does perform the worst.

We further explore the $Recall@10$ and $Hit@10$ metrics changing during the unlearning by figure13. We find that by updating the model with GA alone, the model does reach convergence in the final, but its no longer able for recommendations. When FT is joined, the metrics stabilize at earlier epochs but show a slightly increasing trend in future updates. To compare with the figure13a, GA shows stronger unlearning ability at the 200 to 400 epochs in this experiment, which also decreases the metrics in GA+FT at the same period. After that, GA tends to be stable, and FT performs higher loss during this stage. The optimization object of the loss functions here tends to fit the $\mathcal{D}^{rem}$. In figure13c, GA+KL shows a smooth and continuous decline, the model reaching a lower $Hit@10$ than that of GA. However, it seems to weaken the ability of FT to help the model converge. As shown in figure13d, we observe that the model's metrics have not completely stabilized at the end of the unlearning, representing a slight downward trend. Therefore, the biggest challenge in our research is how to trade off the model's optimized objective across three loss functions.

5 Discussion

We implement an approximate unlearning algorithm for the sequential recommender systems. We first analyze the overall performance of SRUGA based on the experiments above. Then,



(a) Results of FT.

(b) Results of KL.

Figure 11: The results of recommendation performance and unlearning effectiveness under different η_{FT}, η_{KL}. The η_{GA}, η_{FT}, and η_{KL} are default set to 0.5, 0.5, and 1, respectively. The dashed line represents the results of SISA on SASRec trained on Beauty.

we describe the advantages of our algorithm by comparing it with the exact unlearning method SISA. Moreover, we discuss the challenges we faced and the limitations of our algorithm.

Reviewing the results of our experiments, SRUGA achieves comparable performance to the exact-unlearning method SISA. Regarding the unlearning effects, SRUGA has the advantage of stabilized performance, dramatically decreasing the model's ability to predict the forgotten samples and performing similar results in all experiments. In contrast, the performance of SISA depends on the models' structures and the density of datasets. Moreover, SRUGA maintains advantages in unlearning efficiency, especially when the unlearning request happens on a large dataset, which is ten times faster than SISA. This achievement aligns with our exceptions, where the approximate unlearning method saves time from retraining the model.

5.1 Challenges

The biggest challenge in our work is how to trade off the unlearning effectiveness with recommendation performance. As mentioned previously, we manage the model's performance by combining three loss functions. However, the optimization objectives of these loss functions are different. In our experiments, we manually set the weights of these losses to reach a 'balance point', which is not the Pareto optimization in multi-objective problems. We can increase the weight of GA to achieve better unlearning effectiveness or reduce it to maintain more recommendation ability. This uncertainty confuses us when evaluating the SRUGA, as we can not define the unlearning quality through longitudinal comparisons. Therefore, we have to use the performance of SISA as the benchmark when turning the hyper-parameters.

A better solution is to use gradient surgery or uncertainty weighting from multi-task learning (MTL), which automatically adjusts the weights of losses during unlearning. However, the tasks of MTL are often relevant, while the optimization targets are independent. For example, using two loss functions on a model to achieve text classification and sentiment analysis. These two tasks can share the semantic information extracted from the lower layers of the model, but there

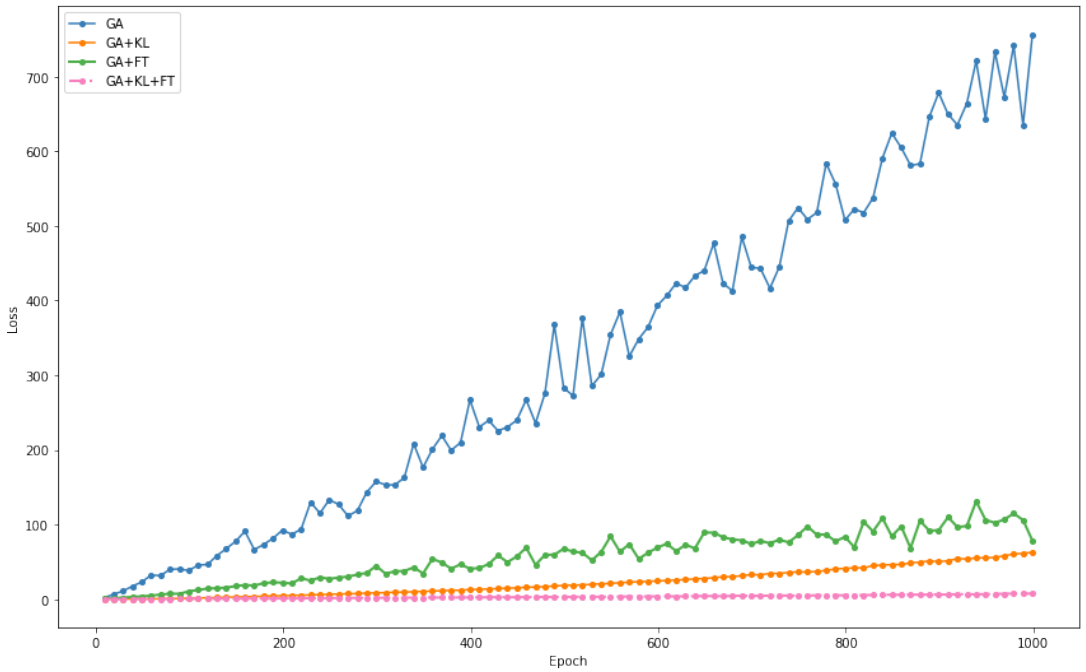
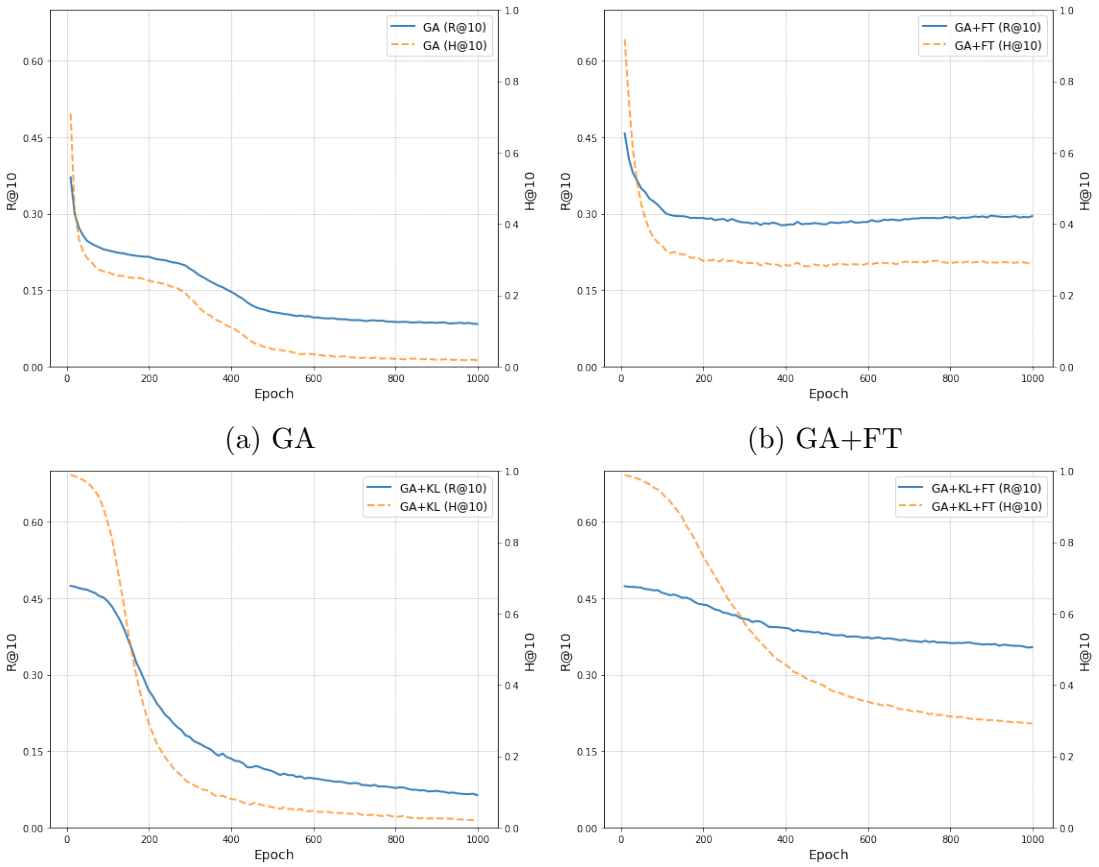


Figure 12: The loss over epochs on $\mathcal{D}^{rem}$ during unlearning SASRec on Beauty dataset.

is no direct conflict between their objectives. In our project, the GA and FT gather information from different datasets. The objective of FT is to enhance recommendation performance on the remaining set, while GA aims to reduce recommendation performance on the forgotten set. There is a conflict between these two objectives in some aspects. Therefore, finding the optimal solution in our unlearning algorithm could be more challenging.

Another challenge is how to measure unlearning effectiveness. When evaluating the sequential recommender system, we follow the leaving one out strategy to extract the last two items from the sequence as the validation and testing sample. Therefore, the model is trained and evaluated in different datasets. However, we did not assign additional validation and test data for the forgotten sequences during the unlearning stage, since they are directly extracted from the training sequence. This resulted in the forgotten sequences being used both as the training set of GA and the testing set for unlearning evaluation. This could lead to a potential advantage for SRUGA when evaluating the effectiveness of unlearning.

An alternative approach is indirectly evaluating the forgotten items using the membership inference attack (MIA), as mentioned in the evaluation sections. The method is often used to test if the target model contains information leakage. In our case, we can split the dataset into the "seen" and "unseen" parts, generating a group of forgotten sequences for each. We collect the output distribution of the target model for these forgotten sequences. Then, we can train an MIA to distinguish whether these distributions are inferred by the "seen" training set of the target model. If we did the same operation for the target model after unlearning and observed that the confidence of MIA is decreased. That means the forgotten data is more difficult to be inferred, indirectly representing the effectiveness of the unlearning.



(a) GA (b) GA+FT

(c) GA+KL (d) GA+KL+FT

Figure 13: The curve of $Recall@10$ and $Hit@10$ over epochs during unlearning SASRec on Beauty dataset.

However, we find that this method does not completely match our task during the research. The output of the SRS for the forgotten item is based on the context information of the entire forgotten sequence. Thus, the MIA trained on these outputs actually evaluates the sequence-level unlearning rather than the item-level in our work. Therefore, we chose a more intuitive $Hit@10$ metric in the experiments.

5.2 limitations

We construct a framework for implementing SRUGA into different sequential recommender systems. However, we have to modify some loss functions and training strategies that the model's original authors use. These changes led to the deterioration of some models, like BERT4Rec, which performs worse than it should. Moreover, SRUGA requires the remaining datasets for applying fine-turning and calculating the kullback-Leibler divergence, which could be a challenge in the dataset from real-world scale. These facts limit the generalization of SRUGA, which can only apply to the models trained in the same framework currently.

In our experiments, we claim the unlearned models are convergent to the stable state by measuring the changing of metrics and loss on the remaining dataset. However, we do not analyze the changes in the model parameters. Specifically, we cannot determine whether ir-

reversible adjustments have happened during the unlearning process. It's uncertain whether the model is still able to learn new information or relearn the forgotten samples. In real-world applications, this is necessary. The unlearning algorithm should execute alternately with normal fine-tuning to achieve dynamic updating of the recommender system.

6 Conclusion

In this thesis, we propose an approximate unlearning framework SRUGA for sequential-based recommender systems. We first design a data partition strategy to maintain the chronological order of the sequential data, making our approach more applicable to the item-level unlearning request. Then, we implement the unlearning algorithm by combining the loss functions of gradient ascent, kullback-Leibler divergence, and fine-turning. By modifying the loss function and sampling strategy, we design a unified training framework that allows our unlearning algorithm to be applied to three different models with the same process. Moreover, we implement an evaluation metric specified for evaluating the unlearning effectiveness. We apply the SRUGA into three representative models across three datasets. The experiment results show our algorithm can achieve similar results as the retraining-based method SISA with lower execution time.

Now, we answer the research questions.

- (RQ1) Gradient ascent dramatically reduces the probability that the unlearned model recommends unlearned samples. However, it leads to a negative impact on the models' overall recommendation abilities. The effect grows stronger when increasing the weights of GA, leading to catastrophic forgetting in the end.
- (RQ2) Kullback-Leibler divergence and fine-turning prevent the model from the negative effect of GA in different aspects. Fine-tuning reduces the performance decrease during the unlearning process and helps the unlearned model achieve earlier convergence. Kullback-Leibler divergence reduces the fluctuations caused by inconsistent objective functions during the unlearning. Also, it makes the output distribution of the unlearned model closer to its original state on the remaining data.
- (RQ3) The running time of SRUGA is an order of magnitude lower than the complete retraining method and outperforms the SISA in most experiments. This advantage grows as the scale of the dataset increases.
- (RQ4) We balance the optimization objective of three loss functions by assigning the weights manually. In the ablation study, we demonstrate that the unlearning model can finally converge on the remaining dataset. In other experiments, we use an annealing algorithm to gradually reduce the weight of gradient ascent, ensuring the unlearning can stabilize stops at the point we expected.

In future work, we can design an algorithm that adjusts the loss weights automatically by referring to the methods used in multi-task learning or multi-objective optimization. In addition, we should evaluate the unlearning effectiveness in various aspects. For example, we can analyze the model's output distribution after unlearned and observe whether the inference attack can infer them. Finally, we plan to analyze the hidden states of sequential recommender

systems to observe the potential influence of the SRUGA. Like its effect on the user who has similar interactions to the forgotten samples and whether the model can relearn this unlearned information through retraining.

References

G. Bebis and M. Georgiopoulos. Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31, 1994. doi: 10.1109/45.329294.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159, 2021. doi: 10.1109/SP40001.2021.00019.

Yinzhi Cao and Junfeng Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480, 2015. doi: 10.1109/SP.2015.35.

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting.

Kongyang Chen, Yiwen Wang, and Yao Huang. Lightweight machine unlearning in neural network, 2021. URL https://arxiv.org/abs/2111.05528.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959190. URL https://doi.org/10.1145/2959100.2959190.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, WWW '19, page 417–426, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313488. URL https://doi.org/10.1145/3308558.3313488.

Jack Foster, Stefan Schoepf, and Alexandra Brintrup. Fast machine unlearning without retraining through selective synaptic dampening. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(11):12043–12051, Mar. 2024. doi: 10.1609/aaai.v38i11.29092. URL https://ojs.aaai.org/index.php/AAAI/article/view/29092.

Alex Graves. *Long Short-Term Memory*, pages 37–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-24797-2. doi: 10.1007/978-3-642-24797-2_4. URL https://doi.org/10.1007/978-3-642-24797-2_4.

Anisa Halimi, Swanand Kadhe, Ambrish Rawat, and Nathalie Baracaldo. Federated unlearning: How to efficiently erase a client in fl?, 2023. URL https://arxiv.org/abs/2207.05521.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 603–618, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349468. doi: 10.1145/3133956.3134012. URL https://doi.org/10.1145/3133956.3134012.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, 2018. doi: 10.1109/ICDM.2018.00035.

Jussi Karlgren. An algebra for recommendations: using reader data as a basis for measuring document proximity. 1990. URL https://api.semanticscholar.org/CorpusID:61709224.

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. ISSN 00034851. URL http://www.jstor.org/stable/2236703.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Ronak Mehta, Sourav Pal, Vikas Singh, and Sathya N. Ravi. Deep unlearning via randomized conditionally independent hessians. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10422–10431, June 2022.

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, 2024. URL https://arxiv.org/abs/2209.02299.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318. Pmlr, 2013.

Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, page 811–820, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605587998. doi: 10.1145/1772690.1772773. URL https://doi.org/10.1145/1772690.1772773.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback, 2012. URL https://arxiv.org/abs/1205.2618.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, page 175–186, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916891. doi: 10.1145/192844.192905. URL https://doi.org/10.1145/192844.192905.

Elaine Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329–354, 1979. ISSN 0364-0213. doi: https://doi.org/10.1016/S0364-0213(79)80012-9. URL https://www.sciencedirect.com/science/article/pii/S0364021379800129.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133480. doi: 10.1145/371920.372071. URL https://doi.org/10.1145/371920.372071.

Michael Schmidt, Jannik Nitschke, and Tim Prinz. Evaluating the performance-deviation of itemknn in recbole and lenskit, 2024. URL https://arxiv.org/abs/2407.13531.

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 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, page 1441–1450, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369763. doi: 10.1145/3357384.3357895. URL https://doi.org/10.1145/3357384.3357895.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), Feb. 2017. doi: 10.1609/aaai.v31i1.11231. URL https://ojs.aaai.org/index.php/AAAI/article/view/11231.

Wilson L. Taylor. "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433, 1953. doi: 10.1177/107769905303000401. URL https://doi.org/10.1177/107769905303000401.

Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, volume 30, pages 47–56. Barcelona, 2000.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Yaozheng Wang, Dawei Feng, Dongsheng Li, Xinyuan Chen, Yunxiang Zhao, and Xin Niu. A mobile recommendation system based on logistic regression and gradient boosting decision trees. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1896–1902, 2016. doi: 10.1109/IJCNN.2016.7727431.

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 17th ACM International Conference on Web Search and Data Mining*, WSDM '24, page 855–863, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703713. doi: 10.1145/3616855.3635823. URL https://doi.org/10.1145/3616855.3635823.

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. doi: 10.1109/TETCI.2024.3379240.

Haonan Yan, Xiaoguang Li, Ziyao Guo, Hui Li, Fenghua Li, and Xiaodong Lin. Arcane: An efficient architecture for exact machine unlearning. In *IJCAI*, volume 6, page 19, 2022.

Yuanshun Yao, Xiaojun Xu, and Yang Liu. Large language model unlearning, 2024. URL `https://arxiv.org/abs/2310.10683`.