Defense Against Model Extraction Attacks on Recommender Systems

Sixiao Zhang
sixiao001@e.ntu.edu.sg
Nanyang Technological University
Singapore

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

Hongxu Chen
hongxu.chen@uq.edu.au
The University of Queensland
Brisbane, Australia

Cheng Long*
c.long@ntu.edu.sg
Nanyang Technological University
Singapore

ABSTRACT

The robustness of recommender systems has become a prominent topic within the research community. Numerous adversarial attacks have been proposed, but most of them rely on extensive prior knowledge, such as all the white-box attacks or most of the black-box attacks which assume that certain external knowledge is available. Among these attacks, the model extraction attack stands out as a promising and practical method, involving training a surrogate model by repeatedly querying the target model. However, there is a significant gap in the existing literature when it comes to defending against model extraction attacks on recommender systems. In this paper, we introduce Gradient-based Ranking Optimization (GRO), which is the first defense strategy designed to counter such attacks. We formalize the defense as an optimization problem, aiming to minimize the loss of the protected target model while maximizing the loss of the attacker's surrogate model. Since top-k ranking lists are non-differentiable, we transform them into swap matrices which are instead differentiable. These swap matrices serve as input to a student model that emulates the surrogate model's behavior. By back-propagating the loss of the student model, we obtain gradients for the swap matrices. These gradients are used to compute a swap loss, which maximizes the loss of the student model. We conducted experiments on three benchmark datasets to evaluate the performance of GRO, and the results demonstrate its superior effectiveness in defending against model extraction attacks.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Data mining*;
• **Computing methodologies** → *Neural networks*.

KEYWORDS

robustness, adversarial defense, model extraction attacks, recommender systems

ACM Reference Format:
Sixiao Zhang, Hongzhi Yin, Hongxu Chen, and Cheng Long. 2024. Defense Against Model Extraction Attacks on Recommender Systems. In *WSDM '24: The 17th ACM International Conference on Web Search and Data Mining, March 04–08, 2024, Merida, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3616855.3635751

1 INTRODUCTION

Recommender systems, which provide suggestions for items that best fit the user's preference, are ubiquitous in our daily lives. They serve as important components in e-commerce [23], social platforms [5], healthcare [34], finance [25], and more. A good recommender system is vital for both users and service providers, as it significantly improves user experience by directing them to new items or products that precisely match their preferences. This, in turn, increases the number of active users and leads to higher profits for the service providers.

Service providers integrate recommender systems into their products and make them accessible to the target users or the public. However, two significant problems need to be addressed before deploying recommender systems: robustness and information leakage [6, 8, 29, 39, 40]. On the one hand, recommender systems are sensitive to noise in the training data, where even a small perturbation can lead to a significant degradation in performance [7, 32]. On the other hand, recommender systems often involve intellectual property that needs protection, or contain private information of the training data that should not be disclosed [33, 37, 38]. Therefore, it is crucial to find ways to protect recommender systems against various adversarial attacks that aim to either poison the model or extract specific information.

Most existing adversarial attack methods for recommender systems assume that the attacker has certain prior knowledge. For example, some attacks are white-box or gray-box attacks [15, 17, 31], where the attacker has access to the target recommender system or the data. Other attacks are black-box attacks but assume that the attacker has access to external knowledge, such as data from a different domain [4], item metadata [3], or some of the real users from the target data [36]. However, such assumptions are often invalid in real-world scenarios, as the required knowledge is not always

*Co-corresponding authors.

WSDM '24, March 04–08, 2024, Merida, Mexico

ACM ISBN 979-8-4007-0371-3/24/03
https://doi.org/10.1145/3616855.3635751

available. On the one hand, the attacker can hardly obtain access to the target model since service providers rarely publish their models. On the other hand, some works assume that the user-item interaction data is visible on the corresponding platforms, which can be easily collected by the attacker. In fact, most platforms nowadays do not make the user information public. Therefore, it is impractical to make such an assumption. Although the attacker can sometimes obtain a dataset by crawling the target platform greedily, it is an extremely time-consuming process, and the quality of such a dataset is not guaranteed.

To attack a target model without prior knowledge, one black-box attack method called *model extraction/stealing attack* [2, 9, 19] has been proposed. The attacker repeatedly queries the target model with synthesized data, and uses the feedback of the target model as labels to create a local dataset. This local dataset is used to train a local surrogate model to approximate the performance of the target model. By treating the surrogate model as the replacement of the target model, various tasks can be performed, including the adversarial attacks. Attacks are carried out based on the surrogate model and then transferred to the target model. This model extraction attack is powerful because it can be used to attack any machine learning models without any prior knowledge. There are only two universally effective defense methods, both with high risks. One method is to detect suspicious queries, but it can be easily bypassed by changing the query patterns. The other method is to alter the output of the model to fool the surrogate model, but this also decreases the utility of the target model.

Researchers have studied how to defend against model extraction attacks in classification tasks [12–14, 20]. One line of research is to detect out-of-distribution queries [12, 13]. However, for recommender systems, out-of-distribution queries are hard to define for a sequence of user interactions. Another line of research is to change the model's output by treating the defense as an optimization problem [14, 20]. However, in classification tasks, the predicted class probabilities are assumed to be visible to the attacker, so the defense methods focus on changing the probabilities of each class while maintaining accuracy, ensuring the probability of the true class is always the largest. Such methods cannot be directly applied to recommender systems, as recommender systems provide top-k rankings instead of class probabilities. Some works have also attempted to do model watermarking [1, 24, 30], where the surrogate model always produces similar outputs to the target model for certain queries. However, watermarking cannot prevent the model from being stolen, so it is not a primary choice in real-world scenarios.

To the best of our knowledge, there are no existing defense methods against model extraction attacks on recommender systems. In this paper, we propose a defense method called Gradient-based Ranking Optimization (GRO). The basic idea of GRO is to learn a target model whose output will maximize the loss of the attacker's surrogate model. Specifically, we use a student model as a replacement of the attacker's surrogate model. The student model tries to extract the target model by training on the top-k lists generated by the target model. We calculate the gradients of the top-k lists w.r.t. the loss of the student model. We can infer how to increase the loss of the student model by altering the top-k lists according to the gradients. However, we cannot directly obtain the gradients of the top-k lists because they are discrete and non-differentiable. We instead convert the list into a swap matrix $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if the j-th item is ranked at the i-th position. The gradients of such swap matrices can be easily obtained. We use these gradients to define a new swap matrix $\mathbf{A}'$ by setting the entries with the largest positive gradients to 1. If $\mathbf{A}'$ is used as the input to the student model, the loss of the student model will be maximized. Therefore, we define a swap loss to force $\mathbf{A}$ to approximate $\mathbf{A}'$, so that the target model will learn to fool the student model. The target model trained with GRO can be deployed directly. The black-box model extraction attacks will acquire a surrogate model with much worse performance than the target model. We make our implementation of GRO available online[1]. Our contributions are summarized as follows:

- We propose GRO, which is, to our knowledge, the first defense method against model extraction attacks on recommender systems. GRO is a general framework and can be used to protect any recommender systems.
- We propose to compute the gradients of the top-k ranking lists by converting them into swap matrices. Such gradients are used to compute a swap loss which can maximize the loss of the attacker's surrogate model.
- Extensive experiments show that GRO can effectively protect the target model from model extraction attacks by reducing the performance of the attacker's model while maintaining the utility of the target model.

2 RELATED WORK

2.1 Black-box Adversarial Attacks on Recommender Systems

The robustness of machine learning models has been studied extensively by the research community [10, 21, 22, 28]. It is one of the major challenges when deploying machine learning models to real-world applications, such as recommender systems. Plenty of works have been proposed to explore the robustness of recommender systems [3, 4, 11, 17, 18, 26, 31, 36]. Many of them assume a black-box setting, where the attacker knows nothing about the target model and its training data. But most of them require external knowledge or extra operations. For example, CopyAttack [4] requires access to user data of a different domain; PoisonRec [26] needs to know the number of times that the target item is recommended (clicked) by all users; KGAttack [3] builds an item knowledge graph by item metadata; Leg-UP [18] requires access to some of the real users; PC-Attack [36] requires data from a different domain, as well as partial data from the target domain.

The above black-box adversarial attacks are impractical to be applied to real-world scenarios. They require either external knowledge [36] or retraining the target model [26], or both [3, 4, 18]. On the one hand, there is no guarantee that high-quality external knowledge is always available. On the other hand, retraining the target model is simply impractical for the attacker. Compared to these attacks, the model extraction attack is a more practical black-box attack. Zhang et al. [41] proposed Reverse Attack. They train a surrogate model to approximate the target model by training

[1] https://github.com/RinneSz/GRO-Gradient-based-Ranking-Optimization

on observed ranking lists. These ranking lists are crawled on the websites of those platforms. No user information is included. Both the training and inference use the similarity of item embeddings as the criteria. Yue et al. [35] proposed a black-box model extraction attack on recommender systems. They query the target model and use the top-k ranking lists returned by the target model to train a surrogate model. The surrogate model is forced to return similar top-k ranking lists to those by the target model. This surrogate model is used as a replacement for the target model to perform various downstream attacks.

2.2 Defenses Against Model Extraction Attacks in Other Domains

People have tried to defend against model extraction attacks in other domains, most of which are for classification tasks [12–14, 20]. However, they are not applicable to the recommender systems because their model assumptions and attack settings are different. For example, Lee et al. [14] proposed Reverse Sigmoid, an activation function used at the last layer of neural networks to replace the traditional Sigmoid. However, one of the premises of this defense is that the attacker can access the output posterior class probabilities. Orekondy et al. [20] modeled the defense problem as a bi-level optimization problem. The goal of the optimization problem is to maximize the gradient deviation between the target model and the surrogate model, to mislead the surrogate model into a different gradient direction. It also assumes that the attacker can access the class probabilities. Kariyappa et al. [13] proposed detecting suspicious queries which are out of distribution (OOD). Then, the output of those queries has minimized probabilities for the correct classes. Juuti et al. [12] proposed PRADA, another detection method to identify OOD queries. However, for recommender systems, OOD queries are more difficult to detect. On the one hand, it is hard to define OOD patterns for sequences. On the other hand, real-world user behaviors are highly irregular, so it is difficult to distinguish abnormal queries from benign queries.

3 PRELIMINARIES

3.1 Problem Definition

Model extraction attacks query the target model with their own data and then use the output of the target model to train a surrogate model. The surrogate model can have a different architecture from the target model, as both the attacker and the defender have no idea about each other's model architecture. This process can be formalized as follows:

$$\min_{\phi} \; L_{\text{surrogate}}(x, f_\theta, g_\phi) = |f_\theta(x) - g_\phi(x)| \tag{1}$$

Here, $L_{\text{surrogate}}$ represents the loss of the surrogate model, x is the query, f is the target model, θ is the parameter of f, g is the surrogate model, and ϕ is the parameter of g. The attacker's goal is to minimize the difference between the output of the target model and the output of the surrogate model.

To defend against model extraction attacks, we need to maximize the loss of the surrogate model while minimizing the loss of the target model. This can be formalized as:

$$\min_{\theta} \; L_{\text{target}}(x, y, f_\theta) - L_{\text{surrogate}}(x, f_\theta, g_\phi), \tag{2}$$

$$\text{s.t. } \phi = \operatorname*{argmin}_{\phi} L_{\text{surrogate}}(x, f_\theta, g_\phi) \tag{3}$$

Here, L_{target} is the original loss function of the target model, and y is the label. This is a bi-level optimization problem. There are two major challenges in solving this problem. First, the defender has no access to the surrogate model, so they cannot optimize $L_{\text{surrogate}}$ directly. But this can be addressed by using a local model to simulate the surrogate model. Second, $f_\theta(x)$ is a discrete ranking list. We cannot obtain its gradient. This makes it impossible to perform back-propagation and optimization. However, we can convert it into a *swap matrix* so that its gradient can be computed. We will discuss how we convert the ranking list into a swap matrix in subsection 4.4.

3.2 Attacker

Before introducing our defense method, we first introduce how the model extraction attack against recommender systems works. The attacker begins by generating a fake user interaction history according to certain strategy, e.g. randomly or autoregressively [35]. Then, the attacker queries the target model and obtains a top-k recommendation list. This list is ordered by preference scores, with items having high scores ranked at the top. The attacker uses these rankings to train a surrogate model that approximates the performance of the target model. The training objective of the attacker is to align the scores of the items with the ordering. It consists of two parts: first, pushing the item ranked higher to have a higher score than items ranked lower, and second, ensuring that the items in the top-k list have higher scores than items outside the top-k list. Specifically, the attacker's loss function is defined as follows:

$$L_{\text{surrogate}} = \sum_{i=1}^{k-1} \max(s_{i+1} - s_i + m_1, 0) + \sum_{i=1}^{k} \max(s_i^{neg} - s_i + m_2, 0) \tag{4}$$

Here, s_i is the surrogate model's prediction score for the item ranked at the i-th position according to the target model's top-k list, s_i^{neg} is a sampled negative item that ranks outside the top-k list, and m_1 and m_2 are two hyper-parameters indicating the margins. The function $\max(\cdot, \cdot)$ returns the maximum value between its arguments. By training the surrogate model with this loss function, it learns to produce similar top-k lists to the target model.

4 GRADIENT-BASED RANKING OPTIMIZATION

4.1 Challenges

To defend against model extraction attacks, an effective solution is to maximize the loss of the attacker's surrogate model. In a strict black-box setting, where the attacker can only use the top-k ranking lists generated by the target model to train the surrogate model, we aim to learn a target model whose generated top-k ranking lists will maximize the loss of the surrogate model. However, since the ranking lists are discrete, we cannot directly obtain the gradients

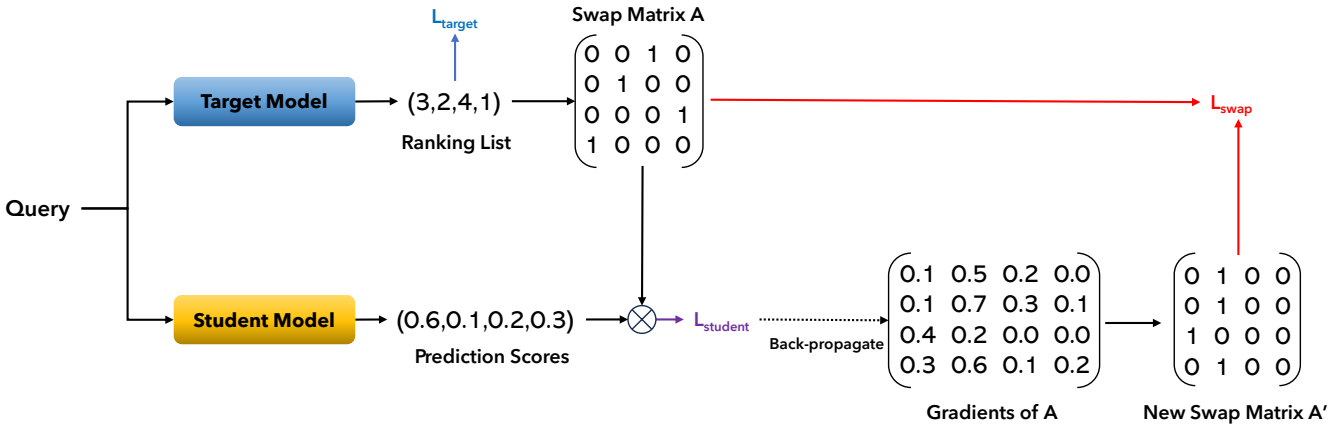


Figure 1: The workflow of GRO. The target model produces a ranking list for the input query. The ranking list is converted into a swap matrix A. Then A is multiplied with the output of the student model to calculate its loss. We back-propagate the loss and obtain the gradients of A, which are then converted into a new swap matrix A′. A swap loss is calculated using A and A′ to learn a target model that can fool the student model.

of them. We need to find an approach to calculate those gradients. This is the main challenge in designing our method.

4.2 Overview

In this section, we introduce our Gradient-based Ranking Optimization (GRO). The basic idea of GRO is to maximize the loss of the surrogate model while minimizing the loss of the target model. GRO is implemented during the training phase of the target model. By training the target model with GRO, it becomes capable of deceiving model extraction attacks while preserving good utility. An illustration of GRO workflow is depicted in Fig. 1. A student model is used to mimic the behavior of the attacker's surrogate model, whose goal is to extract the target model. In each iteration, the target model produces the ranking list of the input query, and then convert it into a swap matrix $\mathbf{A}$. The student model calculates its loss based on the swap matrix. We back-propagate the loss and obtain the gradients of the swap matrix, and convert the gradients into a new swap matrix $\mathbf{A}'$. A swap loss is calculated based on $\mathbf{A}$ and $\mathbf{A}'$. Thus, the target model can learn how to increase the loss of the student model by approximating its output to $\mathbf{A}'$. Next, we introduce each step in detail.

4.3 The Target Model and The Student Model

We first introduce the design of the target model and the student model. In model extraction attacks, it is essential for the target model to be capable of processing unseen queries (i.e., the target model should be inductive). Without loss of generality, we assume a sequential recommendation model as the target model. This is also a practical choice for real-world recommender systems, where encountering new and unseen queries is common. The target model can be any sequential model that takes a sequence of items as input and generates a top-k ranking list as the next item recommendation. The cross-entropy loss is usually used as the loss function in sequential recommendations. In our GRO, we assume that the student model has the same architecture as the target model, which is the worst case for defense where the attacker somehow figures out the target model's architecture. The input of the student model is the same sequence used to train the target model. To train the student model, we use the same loss function as Eq. 4, supervised by the top-k ranking lists produced by the target model. In this way, the student model serves as an approximation of the attacker's surrogate model. We can increase the surrogate model's loss by increasing the student model's loss.

4.4 Converting Top-k Ranking Lists to Swap Matrices

To maximize the loss of the student model, it is necessary to obtain the gradients of the top-k ranking lists. This is non-trivial since the ranking lists are non-differentiable. To solve this problem, we convert the ranking lists into the swap matrices which are instead differentiable.

Assume that the dataset contains a total of m items. For a given top-k ranking list, we can construct a swap matrix $\mathbf{A}$ with dimensions $k \times m$. Suppose the top-k ranking list is $(a_1, a_2, a_3, ..., a_k)$, where a_i is the ID (starting from 1) of the item ranked at the i-th position. In the swap matrix, the entry $\mathbf{A}_{i,a_i}$ is set to 1, while all other entries are set to 0. For instance, if the item with ID=3 is ranked at the first position, then $\mathbf{A}_{1,3}$ would be 1. An example of converting the ranking list into the swap matrix is shown in Fig. 1 upper center. In such a swap matrix, each row contains only one non-zero entry, which indicates the specific item ranked at that position. By performing a matrix multiplication between the swap matrix and the item ID vector $(1, 2, 3, ..., m)^\top$, we obtain the exact top-k ranking list produced by the target model.

Meanwhile, the student model produces a 1-D vector $\mathbf{S}_{\text{student}}$ of scores for all items, ordered by their IDs. This vector has a size of $m \times 1$. By doing a matrix multiplication between $\mathbf{A}$ and $\mathbf{S}_{\text{student}}$, we obtain a new list of scores with size $k \times 1$. This new list preserves the same ordering as the top-k ranking list generated by the target model. It can then be utilized to compute the loss function of the student model using Eq. 4. We can back-propagate the loss and obtain the gradient of $\mathbf{A}$.

4.5 Gradient Computation and Optimization

By back-propagating the loss function of the student model, we can acquire the gradients of the swap matrix $\mathbf{A}$. Now we need to design an objective function based on the gradients to train the target model in order to maximize the loss of the student model.

Each entry in $\mathbf{A}$ has a corresponding gradient value. If the gradient is positive, it means that increasing the value of that entry can generally increase the loss. Instead, if the gradient is negative, it means that decreasing the value of that entry can generally increase the loss. Therefore, if an entry in $\mathbf{A}$ is 0, and its gradient is positive, we can change its value to 1 so that the loss is expected to increase. For each row, by setting the entry with the largest gradient to 1, and setting all other entries to 0, we can obtain a new swap matrix $\mathbf{A}'$. An example is shown in Fig. 1 lower right. If the output of the target model is exactly $\mathbf{A}'$, then the loss of the student model will be maximized. However, simply forcing the model to learn to output $\mathbf{A}'$ is not a good choice. On the one hand, we want to preserve the utility of the target model, but $\mathbf{A}'$ can be very different from $\mathbf{A}$, which will significantly degrade the performance of the model. On the other hand, the model's output may not converge to $\mathbf{A}'$ since $\mathbf{A}'$ can be an invalid swap matrix. Note that one item may be ranked at multiple positions simultaneously in $\mathbf{A}'$. For example, in Fig. 1, the second item has the largest gradient in the first, the second, and the fourth row, then it will be ranked at the first, the second, and the fourth position at the same time, which is impossible for a valid swap matrix where each item can only be ranked at one position. Therefore, we cannot let the model to learn the raw $\mathbf{A}'$ directly. We need to design a loss function that can achieve two goals: first, it should let the output of the target model approach $\mathbf{A}'$ as much as possible, while still being a valid swap matrix; second, the learned swap matrix should preserve the original ranking as much as possible, to avoid a severe utility degradation.

To this end, we define the swap loss as:

$$L_{\text{swap}} = \frac{1}{k}\sum_{i=0}^{k-1} \max((\mathbf{A}_i - \mathbf{A}'_i)\mathbf{S}_{\text{target}} + m_{\text{swap}}, 0) \tag{5}$$

where k is the number of items in the output ranking list (top-k), $\mathbf{A}_i$ and $\mathbf{A}'_i$ are the i-th row of the original swap matrix and the new swap matrix, $\mathbf{S}_{\text{target}}$ is the target model's predicted scores for all items ordered by the item IDs, m_{swap} is a hyper-parameter that denotes the margin, $\max(\cdot,\cdot)$ returns the maximum value. Such a swap loss pushes the item with the largest gradient to have a higher score than the item ranked at the corresponding position. When converged, this loss can ensure that both the orderings of $\mathbf{A}$ and $\mathbf{A}'$ are preserved as much as possible.

Lemma 4.1. *Suppose the top-k rankings indicated by* $\mathbf{A}$ *and* $\mathbf{A}'$ *are* $(a_1, a_2, a_3, ..., a_k)$ *and* $(a'_1, a'_2, a'_3, ..., a'_k)$ *respectively, where* a_i *and* a'_i *denote the items ranked at the i-th position. When the swap loss defined in Eq. 5 (assume* $m_{swap} = 0$*) is converged to 0, for* $\forall i \in [1, k]$*, if* $a'_i \neq a'_j$ *for* $\forall j \in [1, i)$*, we have* $a_i = a'_i$*.*

Proof. Let's consider the first two rows of $\mathbf{A}$ and $\mathbf{A}'$ for an example. Suppose the scores of item a_i and a'_i predicted by the target model are s_i and s'_i respectively. When the loss is converged to 0, for a_1 and a'_1, we have $s_1 \leq s'_1$. If $s_1 < s'_1$ is true, it means that $a_1 \neq a'_1$, because they cannot be the same item if they have different scores. In other words, a_1 and a'_1 are two different items. Therefore, a'_1 should rank higher than a_1 in $\mathbf{A}$ because a'_1 has a higher score. However, a_1 is already ranked at the first position in $\mathbf{A}$, so it is contradictory. Therefore, we can only have $s_1 = s'_1$. In this case, a_1 and a'_1 might be the same item, or might be different items with the same score. Without loss of generality, we assume that they are the same item, since they are interchangeable if they are different items. Then we consider a_2 and a'_2. If converged, we have $s_2 \leq s'_2$. If $s_2 = s'_2$ is true, then a_2 and a'_2 are the same item. If $s_2 < s'_2 < s'_1$, it means that a_2, a'_2, and a'_1 (also a_1) must be three different items. However, since a_2 is already ranked at the second place, it is impossible for two different items to have higher scores than a_2. This is contradictory and cannot be true. But things are different if $s_2 < s'_2 = s'_1$. In this case, we have $a'_2 = a'_1$, which means that the first row and the second row of $\mathbf{A}'$ are identical. One item is ranked at both the first position and the second position in $\mathbf{A}'$ (for example, Fig. 1). In this case, a_2 can be an arbitrary item as long as its score is no less than a_3. □

Table 1: Dataset Statistics

Dataset	# Users	# Items	Avg. length	Density
ML-1M	6040	3416	165.5	4.84%
ML-20M	138493	18345	144.3	0.79%
Steam	334542	13046	12.6	0.10%

According to Theorem 4.1 and its proof, the target model will rank an item at the same position as $\mathbf{A}'$ if the item first appears in $\mathbf{A}'$. For items with multiple appearances, those slots will be filled with other items that did not appear in $\mathbf{A}'$. Most of these filler items are, however, from the top-k list of $\mathbf{A}$, because they tend to have higher scores than other items. Therefore, Eq. 5 can force the model to both learn the new swap matrix $\mathbf{A}'$ and preserve the original swap matrix $\mathbf{A}$.

At last, we jointly train the target model and the student model with the following loss function:

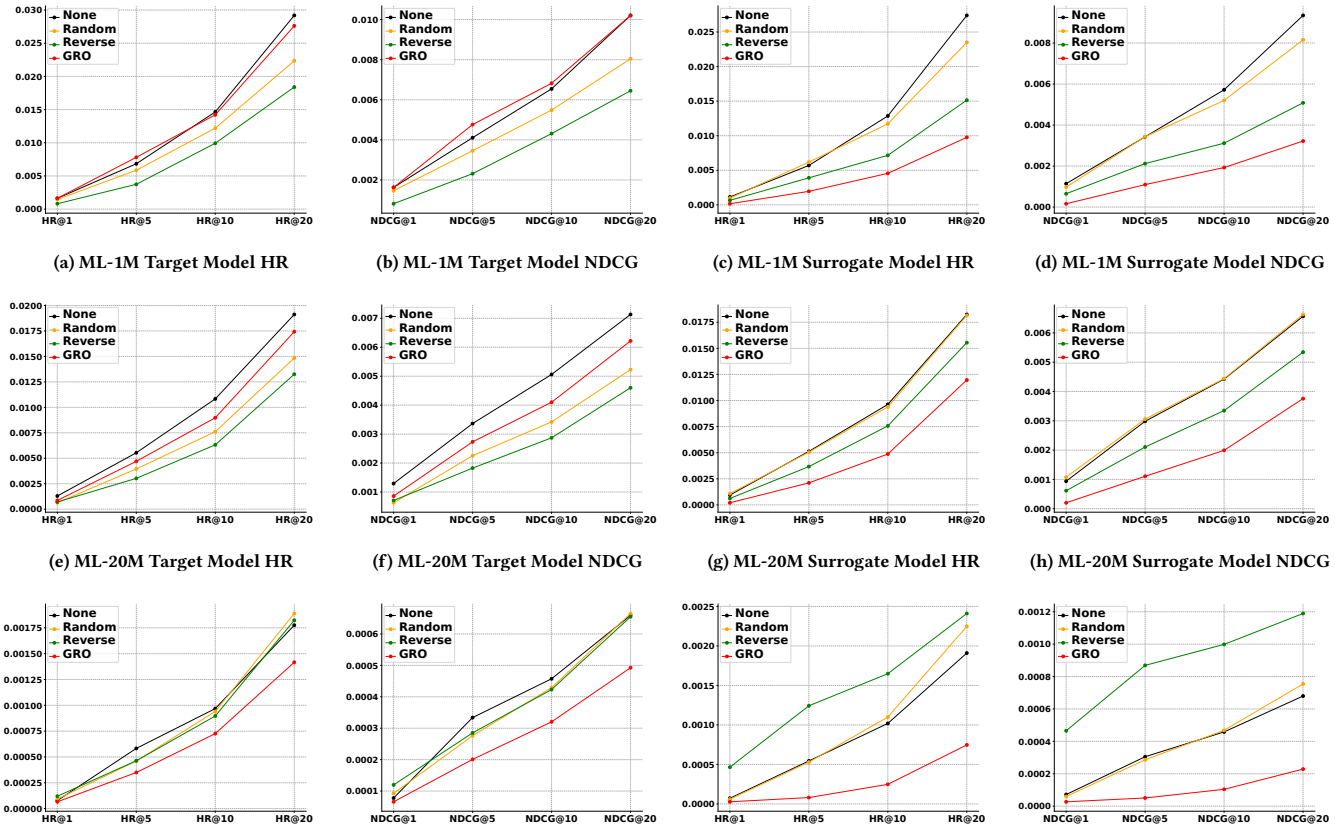
$$L = L_{\text{target}} + L_{\text{student}} + \lambda L_{\text{swap}} \tag{6}$$

where L_{target} is the loss function of the target model, such as the cross-entropy loss; L_{student} is the loss function of the student model (Eq. 4); L_{swap} is the aforementioned swap loss in Eq. 5, and λ is a hyper-parameter controlling the magnitude.

After training, the student model is discarded, and the target model can be deployed directly. The model extraction attacks aiming at stealing the target model can fail to acquire a satisfying performance, as we will show in the experiment section.

5 EXPERIMENTS

We apply GRO to the state-of-the-art sequential recommendation model Bert4Rec [27], a transformer-based model, on three benchmark datasets, then use the state-of-the-art model extraction attack developed by Yue et al. [35] to extract the target model. We compare our GRO with three baselines and show how the target model and the surrogate model perform under each defense. First, we show the result when the surrogate model has the same architecture (Bert4Rec) as the target model. Second, we change the surrogate model into NARM [16], a GRU-based sequential recommendation model, to show the performance of GRO when the target model and the surrogate model have different architectures. At last, we show



(a) ML-1M Target Model HR (b) ML-1M Target Model NDCG (c) ML-1M Surrogate Model HR (d) ML-1M Surrogate Model NDCG

(e) ML-20M Target Model HR (f) ML-20M Target Model NDCG (g) ML-20M Surrogate Model HR (h) ML-20M Surrogate Model NDCG

(i) Steam Target Model HR (j) Steam Target Model NDCG (k) Steam Surrogate Model HR (l) Steam Surrogate Model NDCG

Figure 2: Recommendation performance of the target model and the surrogate model. Both models are Bert4Rec.

how two important hyper-parameters influence the performance, including the number of queries and the λ in Eq. 6.

5.1 Datasets & Evaluation Metrics

We use three benchmark datasets, namely MovieLens-1M, MovieLens-20M[2], and Steam[3]. The statistics of the datasets are summarized in Table 1. Each user has one sequence of interaction history. We follow previous works [27, 35] to use leave-one-out evaluation. The last two items in each sequence are used as the validation set and the test set respectively. We use the Hit Ratio@k (HR@k) and Normalized Discounted Cumulative Gain@k (NDCG@k) as the evaluation metrics to evaluate the utility of the recommender systems.

5.2 Baselines

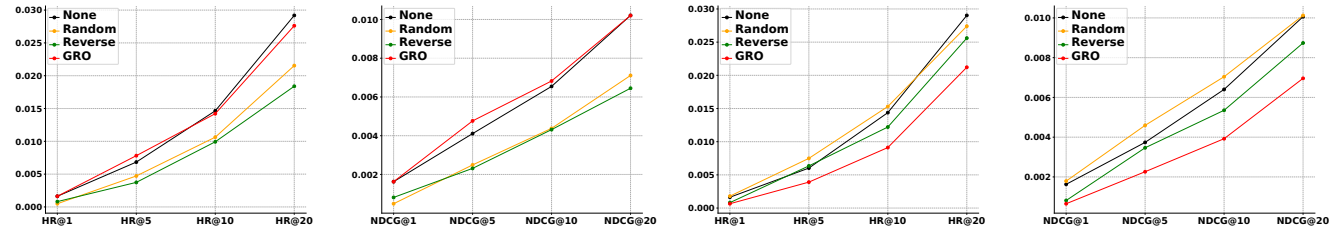
We compare GRO with three baselines, namely None, Random and Reverse. **None**: we do not apply any defense method to the target model. **Random**: we randomly shuffle the top-k recommendation list returned by the target model. The shuffled list is the final output of the recommender system and is what the attacker observes. **Reverse**: we reverse the ordering of the top-k recommendation list. For example, the k-th item becomes the 1st, and the (k-1)-th item becomes the 2nd. This is a strong baseline to defend against model extraction attacks while preserving the utility of the target model. It can be regarded as one of the worst cases of Random.

5.3 Settings

We follow the model-specific hyper-parameter settings for each dataset as suggested by the original Bert4Rec implementation [27]. For the model extraction attack, we also follow the original implementation and hyper-parameter setting [35]. We tune the batch size among {16, 32, 64, 128}, and tune the λ in Eq. 6 among {0.001, 0.01, 0.1, 1.0}. We assume that the target model can return 100 ranked items for each query sequence, based on which we train the surrogate model. The number of sequences is 3000, which means that the attacker generates 3000 sequences to query the target model. These sequences are generated autoregressively, as suggested in [35], where the next item is selected randomly according to the recommendation list returned by the target model.

Previous works for sequential recommendation sample 100 negative items for each test item, and rank the 100 negative items together with the test item to compute the metrics. This may bring sampling bias to the result. We instead rank all the items in the dataset to compute the metrics, which can lead to a more accurate and comprehensive comparison between different methods.

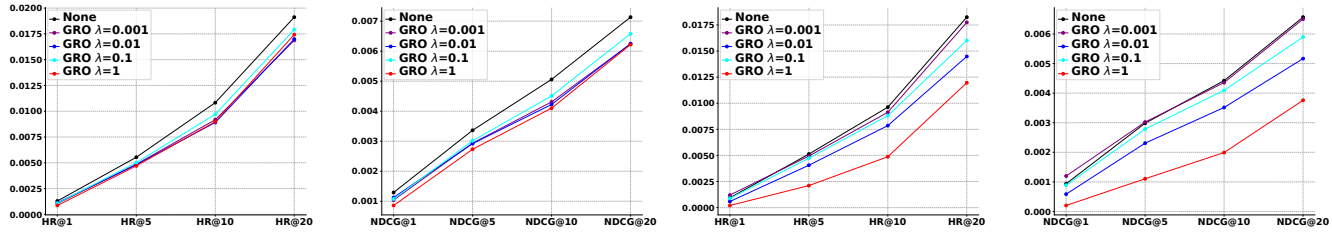
[2]https://grouplens.org/datasets/movielens/
[3]https://cseweb.ucsd.edu/~jmcauley/datasets.html#steam_data



(a) ML-1M Target Model HR (b) ML-1M Target Model NDCG (c) ML-1M Surrogate Model HR (d) ML-1M Surrogate Model NDCG

(e) Steam Target Model HR (f) Steam Target Model NDCG (g) Steam Surrogate Model HR (h) Steam Surrogate Model NDCG

Figure 3: Recommendation performance of the target model and the surrogate model. The target model is Bert4Rec, while the surrogate model is NARM.



(a) ML-20M Target Model HR (b) ML-20M Target Model NDCG (c) ML-20M Surrogate Model HR (d) ML-20M Surrogate Model NDCG

(e) Steam Target Model HR (f) Steam Target Model NDCG (g) Steam Surrogate Model HR (h) Steam Surrogate Model NDCG

Figure 4: Recommendation performance of the target model and the surrogate model with different λ values.

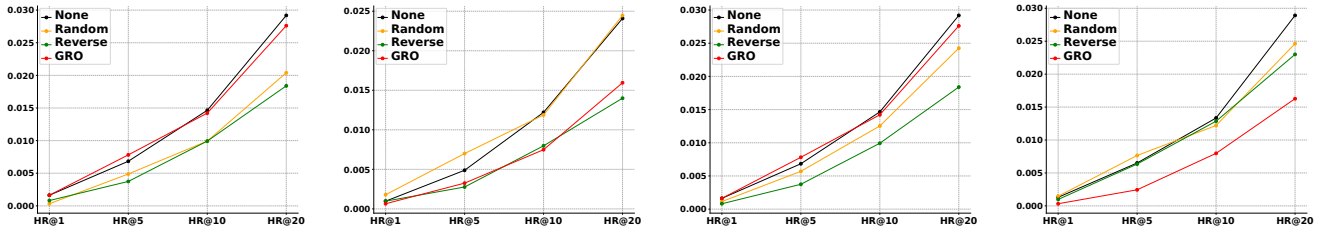
Training a model with GRO is more time-consuming than without it due to the gradient computation. Therefore, we first train the target model until convergence without GRO, then we apply GRO to the converged model and continue to train it for more epochs until convergence. This can significantly speed up the training process.

5.4 Results of Identical Model Architectures

Fig. 2 shows the experiment results when both the target model and the surrogate model are Bert4Rec under different defense methods.

For the first two datasets, ML-1M and ML-20M, GRO significantly outperforms the baselines by being able to both preserve the utility of the target model, and decrease the utility of the surrogate model. According to Fig. 2a, Fig. 2b, Fig. 2e, Fig. 2f, the performance of the target model under GRO is consistently better than Random and Reverse. Then we can see from Fig. 2c, Fig. 2d, Fig. 2g, Fig. 2h, the attacker's surrogate model performs the worst against the target model that is protected by GRO.

For Steam, the observation is a little bit different, but still demonstrates the superiority of GRO. In Fig. 2i and Fig. 2j, GRO seems



(a) Target Model HR, 1000 seqs (b) Surrogate Model HR, 1000 seqs (c) Target Model HR, 5000 seqs (d) Surrogate Model HR, 5000 seqs

Figure 5: Recommendation performance of the target model and the surrogate model on ML-1M with different number of sequences.

slightly worse than Random and Reverse in preserving the utility of the target model. However, in Fig. 2k and Fig. 2l, Random and Reverse fail to defend against the model extraction attack. The surrogate model is comparable or even outperforms the target model under Random and Reverse. Instead, GRO can significantly reduce the utility of the surrogate model. Therefore, for Steam, GRO is still successful in defending against the model extraction attack with acceptable sacrifice of the target model's utility.

5.5 Results of Different Model Architectures

Fig. 3 shows the results when the target model is Bert4Rec and the surrogate model is NARM. To be clear, the student model is also Bert4Rec, which is assumed to be consistent with the target model. We show the results on ML-1M and Steam, while ML-20M has similar results. We can observe that GRO is still the best defense method in this case. For ML-1M, GRO not only preserves the utility of the target model, but also significantly reduces the utility of the surrogate model. For Steam, GRO is comparable or slightly worse in preserving the utility of the target model, but it can effectively fool the surrogate model.

5.6 Analysis of the Choice of λ

We further show how different choices of λ influence the performance of GRO. Fig. 4 shows the experiment results for ML-20M and Steam under different λ values, where both the target model and the surrogate model are Bert4Rec. For ML-20M, the best λ is 1.0. While for Steam, the best λ is 0.001. Besides, the best λ for ML-1M is 0.1. We can see that the performance of GRO has no obvious correlation with λ. Different datasets require different λ values in order for GRO to reach the best performance. However, we observe that the best λ seems to be correlated with the magnitude of the swap loss. For example, after convergence, the swap loss of ML-20M is close to 1, while the best λ is 1; the swap loss of ML-1M is close to 0.1, while the best λ is 0.1; the swap loss of Steam is close to 0.001, while the best λ is 0.001. However, this needs further investigations and justifications. We leave this for future work.

5.7 Results of Different Numbers of Sequences

We show how the number of sequences influence the performance of GRO in Fig. 5. We present the result of HR on ML-1M. The NDCG presents a similar trend. Other datasets also perform similarly. Since experiments in Fig. 2 are conducted with 3000 sequences, we here test the performance of 1000 sequences and 5000 sequences. Fig. 5a and Fig. 5b show the HRs of the target model and the surrogate model with 1000 sequences, while Fig. 5c and Fig. 5d show those with 5000 sequences. For both experiments, GRO significantly outperforms Random and Reverse in preserving the utility of the target model. As for the surrogate model, GRO is consistently better than Random in fooling the surrogate model with both 1000 sequences and 5000 sequences. As for Reverse, according to Fig. 5b, with 1000 sequences, Reverse is comparable with GRO in fooling the surrogate model. However, in Fig. 5d, with 5000 sequences, Reverse performs much worse than before, failing to protect the target model. Note that although Reverse works well in Fig. 5b, the corresponding target model's utility is much worse in Fig. 5a. Together with the experiments of 3000 sequences in Fig. 2, we can draw a conclusion that GRO is still the best defense method even if the number of sequences varies.

6 CONCLUSION

We propose Gradient-based Ranking Optimization (GRO), the first defense method against model extraction attacks on recommender systems. We formalize the defense problem as an optimization problem, and convert the non-differentiable top-k rankings into differentiable swap matrices. We use a student model to learn to extract the target model. We calculate the gradients of the swap matrix with respect to the student model's loss, then convert them into new swap matrices which can maximize the student model's loss. We use a swap loss to force the target model to learn to produce similar rankings as the new swap matrices. Extensive experiments show the superior performance of GRO in decreasing the surrogate model's utility while preserving the target model's utility.

ACKNOWLEDGMENT

This research is supported by the Ministry of Education, Singapore, under its Academic Research Fund (Tier 2 Award MOE-T2EP20221-0013 and Tier 2 Award MOE-T2EP20220-0011). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore. This work is partially supported by the Australian Research Council under the streams of Future Fellowship (No. FT210100624) and Discovery Project (No. DP190101985).

REFERENCES

[1] Franziska Boenisch. 2020. A survey on model watermarking neural networks. *arXiv preprint arXiv:2009.12153* (2020).

[2] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. 2018. Adversarial attacks and defences: A survey. *arXiv preprint arXiv:1810.00069* (2018).

[3] Jingfan Chen, Wenqi Fan, Guanghui Zhu, Xiangyu Zhao, Chunfeng Yuan, Qing Li, and Yihua Huang. 2022. Knowledge-enhanced Black-box Attacks for Recommendations. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 108–117.

[4] Wenqi Fan, Tyler Derr, Xiangyu Zhao, Yao Ma, Hui Liu, Jianping Wang, Jiliang Tang, and Qing Li. 2021. Attacking black-box recommendations via copying cross-domain user profiles. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1583–1594.

[5] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The world wide web conference*. 417–426.

[6] Wenqi Fan, Xiangyu Zhao, Xiao Chen, Jingran Su, Jingtong Gao, Lin Wang, Qidong Liu, Yiqi Wang, Han Xu, Lei Chen, et al. 2022. A Comprehensive Survey on Trustworthy Recommender Systems. *arXiv preprint arXiv:2209.10117* (2022).

[7] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning attacks to graph-based recommender systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 381–392.

[8] Yingqiang Ge, Shuchang Liu, Zuohui Fu, Juntao Tan, Zelong Li, Shuyuan Xu, Yunqi Li, Yikun Xian, and Yongfeng Zhang. 2022. A survey on trustworthy recommender systems. *arXiv preprint arXiv:2207.12515* (2022).

[9] Xueluan Gong, Qian Wang, Yanjiao Chen, Wang Yang, and Xinchang Jiang. 2020. Model extraction attacks and defenses on cloud-based machine learning models. *IEEE Communications Magazine* 58, 12 (2020), 83–89.

[10] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[11] Hai Huang, Jiaming Mu, Neil Zhenqiang Gong, Qi Li, Bin Liu, and Mingwei Xu. 2021. Data Poisoning Attacks to Deep Learning Based Recommender Systems. *arXiv preprint arXiv:2101.02644* (2021).

[12] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. 2019. PRADA: protecting against DNN model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 512–527.

[13] Sanjay Kariyappa and Moinuddin K Qureshi. 2020. Defending against model stealing attacks with adaptive misinformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 770–778.

[14] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. 2019. Defending against neural network model stealing attacks using deceptive perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 43–49.

[15] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. *arXiv preprint arXiv:1608.08182* (2016).

[16] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1419–1428.

[17] Chen Lin, Si Chen, Hui Li, Yanghua Xiao, Lianyun Li, and Qian Yang. 2020. Attacking Recommender Systems with Augmented User Profiles. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 855–864.

[18] Chen Lin, Si Chen, Meifang Zeng, Sheng Zhang, Min Gao, and Hui Li. 2022. Shilling Black-Box Recommender Systems by Learning to Generate Fake User Profiles. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[19] Daryna Oliynyk, Rudolf Mayer, and Andreas Rauber. 2022. I Know What You Trained Last Summer: A Survey on Stealing Machine Learning Models and Defences. *arXiv preprint arXiv:2206.08451* (2022).

[20] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2020. Prediction poisoning: Utility-constrained defenses against model stealing attacks. In *International Conference on Representation Learning (ICLR) 2020*.

[21] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE, 372–387.

[22] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. 2018. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605* (2018).

[23] J Ben Schafer, Joseph A Konstan, and John Riedl. 2001. E-commerce recommendation applications. *Data mining and knowledge discovery* 5 (2001), 115–153.

[24] Masoumeh Shafieinejad, Nils Lukas, Jiaqi Wang, Xinda Li, and Florian Kerschbaum. 2021. On the robustness of backdoor-based watermarking in deep neural networks. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security*. 177–188.

[25] Marwa Sharaf, Ezz El-Din Hemdan, Ayman El-Sayed, and Nirmeen A El-Bahnasawy. 2022. A survey on recommendation systems for financial services. *Multimedia Tools and Applications* 81, 12 (2022), 16761–16781.

[26] Junshuai Song, Zhao Li, Zehong Hu, Yucheng Wu, Zhenpeng Li, Jian Li, and Jun Gao. 2020. Poisonrec: an adaptive data poisoning framework for attacking black-box recommender systems. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 157–168.

[27] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[28] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[29] Qinyong Wang, Hongzhi Yin, Tong Chen, Junliang Yu, Alexander Zhou, and Xiangliang Zhang. 2021. Fast-adapting and privacy-preserving federated recommender system. *The VLDB Journal* (2021), 1–20.

[30] Jing Xu, Stefanos Koffas, Oguzhan Ersoy, and Stjepan Picek. 2021. Watermarking graph neural networks based on backdoor attacks. *arXiv preprint arXiv:2110.11024* (2021).

[31] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. 2017. Fake Co-visitation Injection Attacks to Recommender Systems.. In *NDSS*.

[32] Wei Yuan, Quoc Viet Hung Nguyen, Tieke He, Liang Chen, and Hongzhi Yin. 2023. Manipulating Federated Recommender Systems: Poisoning with Synthetic Users and Its Countermeasures. *arXiv preprint arXiv:2304.03054* (2023).

[33] Wei Yuan, Chaoqun Yang, Quoc Viet Hung Nguyen, Lizhen Cui, Tieke He, and Hongzhi Yin. 2023. Interaction-level membership inference attack against federated recommender systems. *arXiv preprint arXiv:2301.10964* (2023).

[34] Wenbin Yue, Zidong Wang, Jieyu Zhang, and Xiaohui Liu. 2021. An overview of recommendation techniques and their applications in healthcare. *IEEE/CAA Journal of Automatica Sinica* 8, 4 (2021), 701–717.

[35] Zhenrui Yue, Zhankui He, Huimin Zeng, and Julian McAuley. 2021. Black-box attacks on sequential recommenders via data-free model extraction. In *Proceedings of the 15th ACM Conference on Recommender Systems*. 44–54.

[36] Meifang Zeng, Ke Li, Bingchuan Jiang, Liujuan Cao, and Hui Li. 2023. Practical Cross-system Shilling Attacks with Limited Access to Data. *arXiv preprint arXiv:2302.07145* (2023).

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

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

[39] Shijie Zhang, Hongzhi Yin, Tong Chen, Zi Huang, Quoc Viet Hung Nguyen, and Lizhen Cui. 2022. Pipattack: Poisoning federated recommender systems for manipulating item promotion. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*. 1415–1423.

[40] Shijie Zhang, Wei Yuan, and Hongzhi Yin. 2023. Comprehensive privacy analysis on federated recommender system against attribute inference attacks. *IEEE Transactions on Knowledge and Data Engineering* (2023).

[41] Yihe Zhang, Xu Yuan, Jin Li, Jiadong Lou, Li Chen, and Nian-Feng Tzeng. 2021. Reverse Attack: Black-box Attacks on Collaborative Recommendation. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 51–68.