Black-Box Attacks on Sequential Recommenders via Data-Free Model Extraction

Zhenrui Yue*
zhenrui.yue@tum.de
Technical University of Munich
Munich, Germany

Zhankui He*
zhh004@eng.ucsd.edu
University of California, San Diego
San Diego, USA

Huimin Zeng
huimin.zeng@tum.de
Technical University of Munich
Munich, Germany

Julian McAuley
jmcauley@eng.ucsd.edu
University of California, San Diego
San Diego, USA

ABSTRACT

We investigate whether *model extraction* can be used to 'steal' the weights of sequential recommender systems, and the potential threats posed to victims of such attacks. This type of risk has attracted attention in image and text classification, but to our knowledge not in recommender systems. We argue that *sequential* recommender systems are subject to unique vulnerabilities due to the specific autoregressive regimes used to train them. Unlike many existing recommender attackers, which assume the dataset used to train the victim model is exposed to attackers, we consider a *data-free* setting, where training data are not accessible. Under this setting, we propose an API-based model extraction method via limited-budget synthetic data generation and knowledge distillation. We investigate state-of-the-art models for sequential recommendation and show their vulnerability under model extraction and downstream attacks.

We perform attacks in two stages. *(1) Model extraction:* given different types of synthetic data and their labels retrieved from a black-box recommender, we extract the black-box model to a white-box model via distillation. *(2) Downstream attacks:* we attack the black-box model with adversarial samples generated by the white-box recommender. Experiments show the effectiveness of our data-free model extraction and downstream attacks on sequential recommenders in both *profile pollution* and *data poisoning* settings.

ACM Reference Format:
Zhenrui Yue, Zhankui He, Huimin Zeng, and Julian McAuley. 2021. Black-Box Attacks on Sequential Recommenders via Data-Free Model Extraction. In *Fifteenth ACM Conference on Recommender Systems (RecSys '21), September 27-October 1, 2021, Amsterdam, Netherlands.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3460231.3474275

*Both authors contributed equally to this research.

This work is licensed under a Creative Commons Attribution International 4.0 License.

RecSys '21, September 27-October 1, 2021, Amsterdam, Netherlands
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8458-2/21/09.
https://doi.org/10.1145/3460231.3474275

1 INTRODUCTION

Model extraction attacks [26, 40] try to make a local copy of a machine learning model given only access to a query API. Model extraction exposes issues such as sensitive training information leakage [40] and adversarial example attacks [32]. Recently, this topic has attracted attention in image classification [18, 30, 32, 46] and text classification [20, 31]. In this work, we show that model extraction attacks also pose a threat to sequential recommender systems.

Sequential models are a popular framework for personalized recommendation by capturing users' evolving interests and item-to-item transition patterns. In recent years, various neural-network-based models, such as RNN and CNN frameworks (e.g. GRU4Rec[13], Caser[38], NARM[25]) and Transformer frameworks (e.g. SASRec[17], BERT4Rec[37]) are widely used and consistently outperform non-sequential [12, 34] as well as traditional sequential models [11, 35]. However, only a few works have studied attacks on recommenders, and have certain limitations: (1) Few attack methods are tailored to sequential models. Attacks via adversarial machine learning have achieved the state-of-the-art in general recommendation settings [4, 6, 39], but experiments are conducted on matrix-factorization models and are hard to directly apply to sequential recommendation; though some *model-agnostic* attacks [2, 22] can be used in sequential settings, they heavily depend on heuristics and their effectiveness is often limited; (2) Many attack methods assume that full training data for the victim model is exposed to attackers [4, 6, 24, 39, 44]. Such data can be used to train surrogate local models by attackers. However this setting is quite restrictive (or unrealistic), especially in implicit feedback settings (e.g. clicks, views), where data would be very difficult to obtain by an attacker.

We consider a *data-free* setting, where no original training data is available to train a surrogate model. That is, we build a surrogate model without real training data but limited API queries. We first construct downstream attacks against our surrogate (white-box) sequential recommender, then transfer the attacks to the victim (black-box) recommender.

Model extraction on sequential recommenders poses several challenges: (1) no access to the orignial training dataset; (2) unlike image or text tasks, we cannot directly use surrogate datasets with semantic similarities; (3) APIs generally only provide rankings (rather

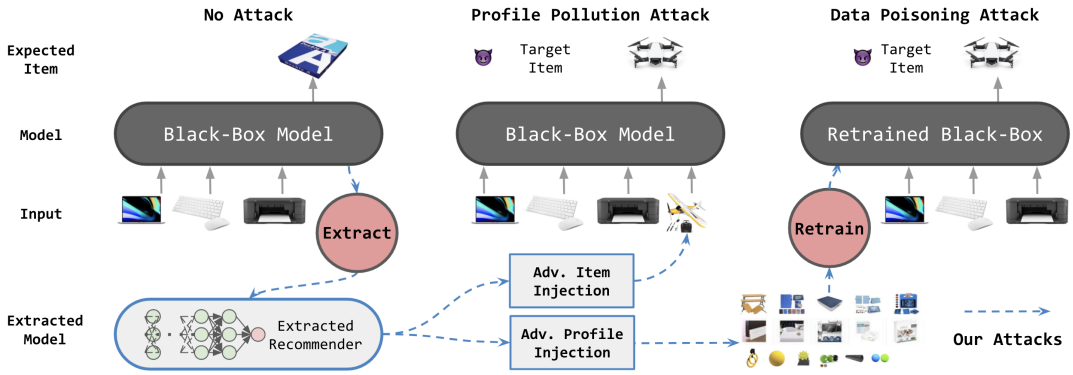


Figure 1: We illustrate two adversarial attack scenarios against sequential recommenders via model extraction.

than e.g. probabilities) and the query budget can be limited. Considering these challenges, sequential recommenders may seem relatively safe. However, noticing sequential recommenders are often trained in an autoregressive way (i.e., predicting the next event in a sequence based on previous ones), our method shows the recommender itself can be used to generate sequential data which is similar to training data from the 'real' data distribution. With this property and a sampling strategy: (1) 'fake' training data can be constructed that renders sequential recommenders vulnerable to model extraction; (2) The 'fake' data from a limited number of API queries can resemble normal user behavior, which is difficult to detect.

Downstream attacks are performed given the extracted surrogate model (see Figure 1). But attack methods tailored to sequential recommenders are scarce [44]. In this work, we propose two attack methods with adversarial example techniques to current sequential recommenders, including *profile pollution attacks* (that operate by 'appending' items to users' logs) and *data poisoning attacks* (where 'fake' users are generated to bias the retrained model). We extensively evaluate the effectiveness of our strategies in the setting that a black-box sequential model returns top-k ranked lists.

2 RELATED WORK

2.1 Model Extraction in Image and Text Tasks

Model extraction attacks are proposed in [26, 40] by 'stealing' model weights to make a local model copy [18, 20, 30–32, 46]. Prior works are often concerned with image classification. To extract the target model weights, *JBDA* [30] and *KnockoffNets* [32] assume the attackers have access to partial training data or a surrogate dataset with semantic similarities. Recently, methods have been proposed in data-free settings. *DaST* [46] adopted multi-branch Generative Adversarial Networks [7] to generate synthetic samples, which are subsequently labeled by the target model. *MAZE* [18] generated inputs that maximize the disagreement between the attacker and the target model. *MAZE* used zeroth-order gradient estimation to optimize the generator module for accurate attacks. Because of the discrete nature of the input space, the methods above cannot transfer to sequential data directly. For Natural Language Processing (NLP) systems, *THIEVES* [20] studied model extraction attacks on BERT-based APIs [5]. Even though attacks against BERT-based models exist for NLP, where authors find *random* word sequences and a surrogate dataset (e.g. WikiText-103 [28]) that can both create effective queries and retrieve labels to approximate the target model, we found that (1) in recommendation, it is hard to use surrogate datasets with semantic similarities (as is common in NLP); we also adopt *random* item sequences as a baseline but their model extraction performance is limited. Therefore we generate data following the *autoregressive* property of sequential recommenders; (2) Compared with NLP, it is harder to distill 'ranking' (instead of classification) in recommendation. We design a pair-wise ranking loss to tackle the challenge; (3) downstream attacks after model extraction are under-explored, especially in recommendation, so our work also contributes in this regard.

2.2 Attacks on Recommender Systems

Existing works [15, 42] categorize attacks on recommender systems as *profile pollution attacks* and *data poisoning attacks*, affecting a recommender system in test and training phases respectively.

Profile pollution attacks aim to pollute a target user's profile (such as their view history) to manipulate the specific user's recommendation results. For example, [41] uses a cross-site request forgery (CSRF) technique [43] to inject 'fake' user views into target user logs in real-world websites including YouTube, Amazon and Google Search. However the strategy used in [41] to decide which 'fake' item should be injected is a simple and heuristic without the knowledge of victim recommenders. In our work, given that we can extract victim recommender weights, more effective attacks can be investigated, such as *evasion attacks* in general machine learning [8, 21, 32]. Note that in our work, we assume we can append items to target user logs with injection attacks via implanted malware [23, 33], where item interactions could be added on users' behalf, therefore, we focus on injecting algorithm design and attack transferabilities (exact approaches for malware development and activity injection are cybersecurity tasks and beyond our research scope).

Data poisoning attacks (*a.k.a.* Shilling attacks [9, 22]) generate ratings from a number of fake users to poison training data. Some poisoning methods are recommender-agnostic [2, 22] (i.e., they do not consider the characteristics of the model architecture) but they heavily depend on heuristics and often limit their effectiveness.

Meanwhile, poisoning attacks have been proposed for specific recommender architectures. For example, [4, 24, 39] propose poisoning algorithms for matrix-factorization recommenders and [42] poisons co-visitation-based recommenders. Recently, as deep learning has been widely applied to recommendation, [15] investigate data poisoning in the neural collaborative filtering framework (NCF) [12]. The closest works to ours are perhaps LOKI [44], because of the *sequential* setting and [4, 6, 39] which adopt adversarial machine learning techniques to generate 'fake' user profiles for matrix-factorization models. One (fairly restrictive) limitation of LOKI is the assumption that the attacker can access complete interaction data used for training. Another limitation is that LOKI is infeasible against deep networks (e.g. RNN / transformer), due to an unaffordable (even with tricks) Hessian computation. In our work, black-box recommender weights are extracted for attacks without any real training data; we further design approximated and effective attacks to these recommenders.

3 FRAMEWORK

Our framework has two stages: *(1) Model extraction:* we generate informative synthetic data to train our white-box recommender that can rapidly close the gap between the victim recommender and ours via knowledge distillation [14]; *(2) Downstream attacks:* we propose gradient-based adversarial sample generation algorithms, which allows us to find effective adversarial sequences in the discrete item space from the *white-box* recommender and achieve successful profile pollution or data poisoning attacks against the *victim* recommender.

3.1 Setting

To focus on model extraction and attacks against black-box sequential recommender systems, we first introduce some details regarding our setting. We formalize the problem with the following settings to define the research scope:

- *Unknown Weights:* Weights or metrics of the victim recommender are not provided.
- *Data-Free:* Original training data is not available, and item statistics (e.g. popularity) are not accessible.
- *Limited API Queries:* Given some input data, the victim model API provides a ranked list of items (e.g. top 100 recommended items). To avoid large numbers of API requests, we define budgets for the total number of victim model queries. Here, we treat each input sequence as one budget unit.
- *(Partially) Known Architecture:* Although weights are confidential, model architectures are known (e.g. we know the victim recommender is a transformer-based model). We also relax this assumption to cases where the white-box recommender uses a different sequential model architecture from the victim recommender.

3.2 Threat Model

3.2.1 Black-box Victim Recommender. Formally, we first denote $\mathcal{I}$ as the discrete item space with $|\mathcal{I}|$ elements. Given a sequence of length T ordered by timestamp, i.e., $\boldsymbol{x} = [x_1, x_2, \ldots, x_T]$ where $x_i \in \mathcal{I}$, a victim sequential recommender f_b is a *black-box*, i.e., the weights are unknown. f_b should return a truncated ranked list over the next possible item in the item space $\mathcal{I}$, i.e., $\hat{\boldsymbol{I}}^k = f_b(\boldsymbol{x})$, where $\hat{\boldsymbol{I}}^k$ is the truncated ranked list for the top-k recommended items. Many platforms surface ranked lists in such a form.

3.2.2 White-box Surrogate Recommender. We construct the white-box model f_w with two components: an embedding layer $f_{w,e}$ and a sequential model $f_{w,m}$ such that $f_w(\boldsymbol{x}) = f_{w,m}(f_{w,e}(\boldsymbol{x}))$. Although the black-box model only returns a list of recommended items, the output scores $\hat{S}_w$ of white-box model f_w over the input space $\mathcal{I}$ can now be accessed, i.e., $\hat{S}_w = f_w(\boldsymbol{x})$.

3.3 Attack Goal

3.3.1 Model Extraction. As motivated previously, the first step is to extract a **white-box** model f_w from a trained, **black-box** victim model f_b. Without accessing the weights of f_b, we obtain information from the black-box model by making limited queries and saving the predicted ranked list $\hat{\boldsymbol{I}}^k$ from each query. In other words, we want to minimize the distance between the black-box f_b and white-box model f_w on query results. We are able to achieve the goal of learning a white-box model via knowledge distillation [14]. Mathematically, the model extraction process can be formulated as an optimization problem:

$$f_w^* = \arg\min_{f_w} \sum_{i=1}^{|\boldsymbol{\mathcal{X}}|} \mathcal{L}_{dis}(\hat{\boldsymbol{I}}^{k^{(i)}}, f_w(\boldsymbol{x}^{(i)})), \tag{1}$$

where $\boldsymbol{\mathcal{X}}$ represents a set of sequences and $\boldsymbol{x}^{(i)} \in \boldsymbol{\mathcal{X}}$ with unique identifier i. We define $\hat{\boldsymbol{\mathcal{I}}}^k$ as a set of top-k predicted ranked lists, where $\hat{\boldsymbol{I}}^{k^{(i)}} \in \hat{\boldsymbol{\mathcal{I}}}^k$ is the black-box model output for $\boldsymbol{x}^{(i)}$. Note that data $(\boldsymbol{\mathcal{X}}, \hat{\mathcal{I}}^k)$ are not real training data. Instead, they are generated with specific strategies, whose details will be included in Section 4. $\mathcal{L}_{dis}$ is a loss function measuring the distance between two model outputs, such as a ranking loss.

3.3.2 Downstream Attacks. We use the extracted white-box model f_w^* as the surrogate of the black-box model f_b to construct attacks. In this work, we investigate **targeted promotion attacks**, whose goal is to increase the target item exposure to users as much as possible, which is a common attack scenario [39]. Note that targeted *demotion* attacks can also be constructed with similar techniques. Formally, the objective of targeted promotion attacks is:

- *Profile Pollution Attack:* We define profile pollution attacks formally as the problem of finding the optimum injection items $\boldsymbol{z}^*$ (that should be items appended after the original sequence $\boldsymbol{x}$) that maximize the target item exposure $\boldsymbol{E}_t$, which can be characterized with common ranking measures like *Recall* or *NDCG* [17, 25, 37]:

$$\boldsymbol{z}^* = \arg\max_{\boldsymbol{z}} \boldsymbol{E}_t(f_b([\boldsymbol{x}; \boldsymbol{z}])), \tag{2}$$

  where $[\boldsymbol{x}; \boldsymbol{z}]$ refers to the concatenation of the sequence $\boldsymbol{x}$ and attacking items $\boldsymbol{z}$. Note that in profile pollution attacks setting, no retraining needed and this **user-specific profile $\boldsymbol{x}$ is assumed to be accessed and can be injected** (e.g. using malwares [23, 33]; see Section 2.2).
- *Data Poisoning Attack:* Similarly, poisoning attacks can be viewed as finding biased injection profiles $\mathcal{Z}$, such that after retraining, the recommender propagates the bias and is more likely to recommend the target. $\mathcal{Z} \cup \mathcal{X}$ refers to the injection of fake profiles $\mathcal{Z}$ into normal training data and f_b'

is the retrained recommender with recommender training loss function $\mathcal{L}_{rec}$ as:

$$\mathcal{Z}^* = \arg\max_{\mathcal{Z}} \sum_{i=1}^{|\mathcal{X}|} \boldsymbol{E}_t(f_b^{'}(\boldsymbol{x}^{(i)})) \quad \text{s.t.} \quad f_b^{'} = \arg\min_{f_b} \mathcal{L}_{rec}(f_b, \mathcal{Z} \cup \mathcal{X}). \tag{3}$$

4 METHODOLOGY

4.1 Data-Free Model Extraction

To extract a black-box recommender in a data-free setting, we complete the process in two steps: (1) *data generation*, which generates input sequences $\mathcal{X}$ and output ranking lists $\hat{I}^k$; (2) *model distillation*, using $(\mathcal{X}, \hat{I}^k)$ to minimize the difference between the black- and white-box recommender.

4.1.1 Data Generation. Considering that we don't have access to the original training data and item statistics, a trivial solution is to make use of random data and acquire the model recommendations for later stages.

- *Random:* Items are uniformly sampled from the input space to form sequences $\mathcal{X}_{rand} = \{\boldsymbol{x}_{rand}^{(i)}\}_{i=1}^{B}$ where $\boldsymbol{x}_{rand}^{(i)}$ is a generated sequence with identifier i and B is the budget size. Top-k items ($k = 100$ in our experiments) are acquired from the victim recommender in each step to form the output result set, i.e., $\hat{I}_{rand}^k = \{[f_b(\boldsymbol{x}_{rand[:1]}^{(i)}), \ldots, f_b(\boldsymbol{x}_{rand[:T_i]}^{(i)})]\}_{i=1}^{B}$, where the $[:t]$ operation truncates the first t items in the sequence (corresponding to a recommendation list after each click). T_i is the length of $\boldsymbol{x}_{rand}^{(i)}$, where T_i can be sampled from a predefined distribution or simply set as a fixed value. Following this strategy, we generate inputs and labels $(\mathcal{X}_{rand}, \hat{I}_{rand}^k)$ for model distillation.

However, random data cannot simulate real user behavior sequences $\boldsymbol{x}_{real}$ where sequential dependecies exist among different steps. To tackle this problem, we propose an autoregressive generation strategy. Inspired by autoregressive language models where generated sentences are similar to a 'real' data distribution, and the finding that sequential recommenders are often trained in an autoregressive way [13, 17, 25], we generate fake sequences autoregressively.

- *Autoregressive:* To generate one sequence $\boldsymbol{x}_{auto}^{(i)}$, a random item is sampled as the start item $\boldsymbol{x}_{auto[1]}^{(i)}$ (the $[t]$ operation selects the t-th item in sequence $\boldsymbol{x}$) and fed to a sequential recommender to get a recommendation list $f_b(\boldsymbol{x}_{auto[:1]}^{(i)})$. We repeat this step autoregressively i.e., $\boldsymbol{x}_{auto[t]}^{(i)} = \texttt{sampler}(f_b(\boldsymbol{x}_{auto[:t-1]}^{(i)}))$ to generate sequences up to the maximum length T_i. Here `sampler` is a method to sample one item from the given top-k list. In our experiment, sampling from the top-k items with monotonically decreasing probability performs similarly to uniform sampling, so we favor sampling when selecting the next item from the top-k list. Accordingly, we generate B sequences $\mathcal{X}_{auto} = \{\boldsymbol{x}_{auto}^{(i)}\}_{i=1}^{B}$ and record top-k lists, forming a dataset $(\mathcal{X}_{auto}, \hat{I}_{auto}^k)$ for model distillation.

For autoregressive sequence generation, Figure 2a visually represents the process of accumulating data by repeatedly feeding current data to the recommender and appending current sequences with the sampled items from the model output. *Autoregressive* method is beneficial as: (1) Generated data is more diverse and more representative of real user behavior. In particular, sampling instead of choosing the first recommended item will help to build more diversified and 'user-like' distillation data; (2) Since API queries are limited, *autoregressive* method generates data by resembling real data distribution can obtain higher-quality data to train a surrogate model effectively; (3) It resembles the behavior of real users so is hard to detect. Nevertheless, *autoregressive* method does not exploit output rankings and item properties like similarity due to restricted data access, which could limit the full utilization of a sequential recommender.

4.1.2 Model Distillation. We use model distillation [14] to minimize the difference between f_w and f_b by training with generated data $(\mathcal{X}, \hat{I}^k)$ (see Figure 2b). To get the most out of a single sequence during distillation, we generate sub-sequences and labels to enrich training data; for an input sequence $\boldsymbol{x} = [x_1, x_2, \ldots, x_T]$ from $\mathcal{X}$, we split it into $T-1$ entries of $\boldsymbol{x}_{[:2]}, \boldsymbol{x}_{[:3]}, \ldots, \boldsymbol{x}_{[:T]}$ following training strategies in [13, 25].

Compared to traditional model distillation [14, 20], where a model is distilled from predicted label *probabilities*, in our setting we only have top-k ranking list instead of probability distributions. So we propose a method to distill the model with a ranking loss. We can access the white-box model f_w output scores to items from the black-box top-k list $\hat{\boldsymbol{I}} = f_b(\boldsymbol{x})$, which is defined as $\hat{\boldsymbol{S}}_w^k = [f_w(\boldsymbol{x})_{[\hat{I}_{[i]}^k]}]_{i=1}^k$. For example, for $\hat{\boldsymbol{I}}^k = [25, 3, \ldots, 99]$, $\hat{\boldsymbol{S}}_w^k = [f_w(\boldsymbol{x})_{[25]}, f_w(\boldsymbol{x})_{[3]}, \ldots, f_w(\boldsymbol{x})_{[99]}]$). We also sample k negative items uniformly and retrieve their scores as $\hat{\boldsymbol{S}}_{neg}^k$. We design a pair-wise ranking loss $\mathcal{L}_{dis}$ to measure the distance between black-box and white-box outputs:

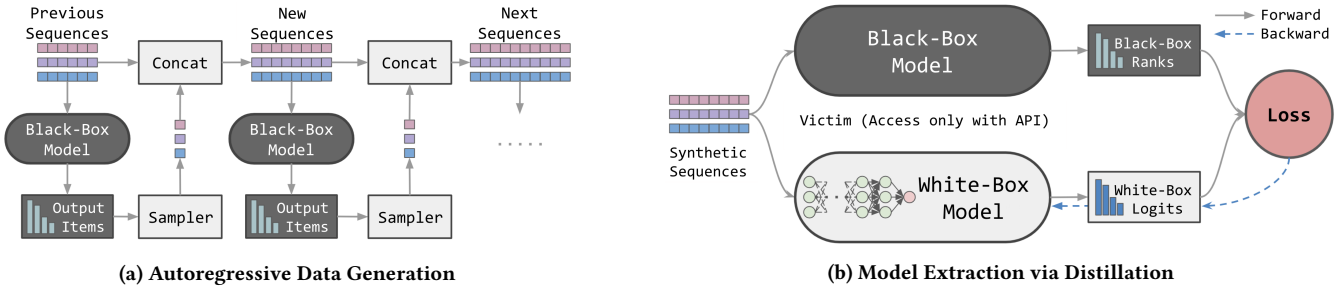
$$\mathcal{L}_{dis} = \frac{1}{k-1} \sum_{i=1}^{k-1} \max(0, \hat{S}_{w[i+1]}^k - \hat{S}_{w[i]}^k + \lambda_1) + \frac{1}{k} \sum_{i=1}^{k} \max(0, \hat{S}_{neg[i]}^k - \hat{S}_{w[i]}^k + \lambda_2). \tag{4}$$

The loss function consists of two terms. The first term emphasizes ranking by computing a marginal ranking loss between all neighboring item pairs in $\hat{\boldsymbol{S}}_w^k$, which maximizes the probability of ranking positive items in the same order as the black-box recommender system. The second term punishes negative samples when they have higher scores than the top-k items, such that the distilled model learns to 'recall' similar top-k groups for next-item recommendation. λ_1 and λ_2 are two margin values to be empirically set as hyperparameters.

4.2 Downstream Attacks

To investigate whether attacks can be transfered from the trained white-box model f_w^*[1] to the black-box model f_b, we introduce two

[1] We directly use f_w to represent the trained white-box f_w^* below.



(a) Autoregressive Data Generation

(b) Model Extraction via Distillation

Figure 2: Autoregressive sequences and the use of synthetic data for model extraction.

Algorithm 1: Adversarial Item Search for Profile Pollution

Input sequence $\boldsymbol{x}$, target t, expected length of polluted sequence T, white-box model f_w (i.e. $f_{w,e}$, $f_{w,m}$), ϵ and n;
Output polluted sequence $\boldsymbol{z}$;
initialize $\boldsymbol{z}$ with $\boldsymbol{z} = \boldsymbol{x}$;
while *length of* $\boldsymbol{z} < T$ **do**
  extend $\boldsymbol{z}$ by appending target item: $\boldsymbol{z} \leftarrow [\boldsymbol{z}; t]$;
  transform sequence into sequence embeddings: $\tilde{\boldsymbol{z}} = f_{w,e}(\boldsymbol{z})$;
  compute backward gradients using cross entropy $\nabla = \nabla_{\tilde{\boldsymbol{z}}} \mathcal{L}_{ce}(f_{w,m}(\tilde{\boldsymbol{z}}), t)$;
  compute cosine similarity scores $\boldsymbol{S}$ between $\tilde{\boldsymbol{z}} - \epsilon sign(\nabla)$ and $f_{w,e}(i)\ \forall i \in \mathcal{I}$;
  select n candidates $\boldsymbol{C}$ with highest $\boldsymbol{S}$ scores, exclude t if repeated;
  replace t in $\boldsymbol{z}$ with $c \in \boldsymbol{C}$ and keep $\boldsymbol{z}$ with highest $f_w(\boldsymbol{z})_{[t]}$;
end

Algorithm 2: Adversarial Profile Generation for Data Poisoning

Input target t, expected length T, white-box model f_w (i.e. $f_{w,e}$, $f_{w,m}$), ϵ and n;
Output adversarial profile $\boldsymbol{z}$;
initialize $\boldsymbol{z}$ with $\boldsymbol{z} = [t]$;
while *length of* $\boldsymbol{z} < T$ **do**
  sample item z_{temp} in $\mathcal{I}$ and append to $\boldsymbol{z}$: $\boldsymbol{z} = [\boldsymbol{z}; z_{temp}]$;
  transform sequence into sequence embeddings: $\tilde{\boldsymbol{z}} = f_{w,e}(\boldsymbol{z})$;
  compute backward gradients using cross entropy $\nabla = \nabla_{\tilde{\boldsymbol{z}}} \mathcal{L}_{ce}(f_{w,m}(\tilde{\boldsymbol{z}}), t)$;
  compute cosine similarity scores $\boldsymbol{S}$ between $\tilde{\boldsymbol{z}} - \epsilon sign(\nabla)$ and $f_{w,e}(i)\ \forall i \in \mathcal{I}$;
  select n candidates $\boldsymbol{C}$ with lowest $\boldsymbol{S}$ scores, exclude t if repeated;
  sample item in $\boldsymbol{C}$ and replace z_{temp} in $\boldsymbol{z}$;
  append target item $\boldsymbol{z} \leftarrow [\boldsymbol{z}; t]$;
end

model attacks against sequential recommender systems: *profile pollution* attacks and *data poisoning* attacks, see Figure 1 for illustration of the two attack scenarios.

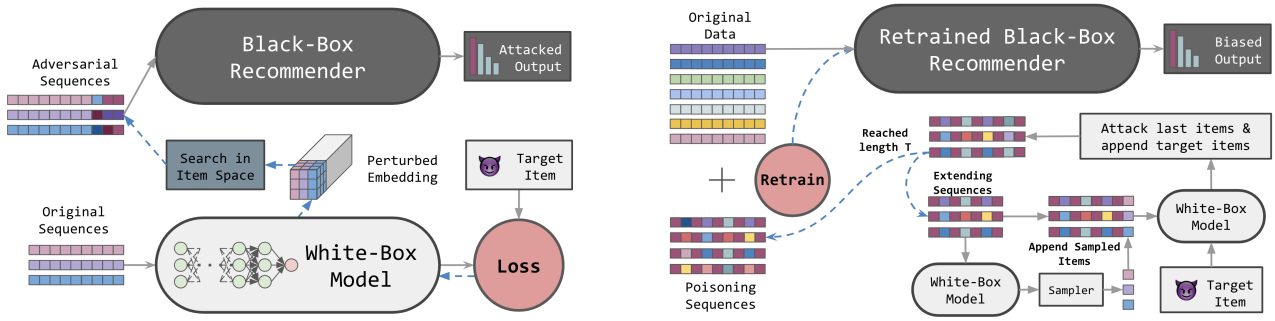
4.2.1 Profile Pollution Attack. As described in Figure 3a, we perform profile pollution attacks to promote item exposure and use Algorithm 1 to construct the manipulated sequence in the input space. Because we can access the gradients in white box models, we are able to append 'adversarial' items by extending adversarial example techniques (e.g. Targeted Fast Gradient Sign Method (T-FGSM) [8]) from continuous feature space (e.g. image pixel values) to discrete item space (e.g. item IDs), with an assumption that the optimal item is 'close' to target item in embedding space. Therefore we can achieve user-specific item promotion without the black-box recommender being retrained as below.

Step 1: Compute Gradients at the Embedding Level. Given the user history $\boldsymbol{x}$, we construct the corrupted sequence $\boldsymbol{z}$ by appending adversarial items after $\boldsymbol{x}$. We first initialize $\boldsymbol{z}$ to be the same as $\boldsymbol{x}$ and append target items to it. With $f_{w,m}$ from the previous step, we feed the embedded input $\tilde{\boldsymbol{z}}$ to the model and compute backward gradients w.r.t. the input embeddings using a cross-entropy loss, where the target item t is used as a label: $\nabla = \nabla_{\tilde{\boldsymbol{z}}} \mathcal{L}_{ce}(f_{w,m}(\tilde{\boldsymbol{z}}), t)$.

Step 2: Search for Adversarial Candidates. Based on $\boldsymbol{z}$ and ∇ from the previous step, we first perform T-FGSM to compute the perturbed embeddings $\tilde{\boldsymbol{z}}' = \tilde{\boldsymbol{z}} - \epsilon sign(\nabla)$, then the cosine similarity of the embedded injection item in $\tilde{\boldsymbol{z}}'$ is computed with all item embeddings across $\mathcal{I}$. We select n adversarial candidates with the highest cosine similarity. These items are tested with f_w such that the candidate leading to the highest ranking score is kept as the adversarial item. It can be repeated for multiple injection items for better attack performance, and to avoid disproportionate target items in injection, we require target items not appear continuously.

4.2.2 Data Poisoning. Data poisoning attacks operate via fake profile injection, to promote target item exposure as much as possible (after retraining with the fake and normal profiles). We propose a simple adversarial strategy (visualized in Figure 3b) to generate poisoning data with white-box model f_w. The intuition behind our poisoning data generation is that *the next item should be the target even given sequences of seemingly irrelevant items.*

In this case, we follow co-visitation approach [36, 42] and apply adversarial example techniques [8] to generate poisoning data. (1) We consider one poisoning sequence using alternating items



(a) Profile pollution attack with white-box model

(b) Data poisoning attack via adversarial co-visitation

Figure 3: Two scenarios: profile pollution attack and data poisoning attacks.

Table 1: Data Statistics

Datasets	Users	Items	Avg. len	Max. len	Sparsity
ML-1M	6,040	3,416	166	2277	95.16%
Steam	334,542	13,046	13	2045	99.90%
Beauty	40,226	54,542	9	293	99.98%

Table 2: Sequential Model Architecture

Model	Basic Block	Training Schema
NARM [25]	GRU	Autoregressive
SASRec [17]	TRM	Autoregressive
BERT4Rec [37]	TRM	Auto-encoding

pairs (e.g. $\boldsymbol{z}$ =[target, z_2, target, z_4, target, ...]); (2) We try to find *irrelevant items* to fill z_{2d} in $\boldsymbol{z}$. In detail, we use a similar approach to Algorithm 1, where the generation process first computes backward gradients similarly to the cross entropy loss and T-FGSM. However, in narrowing candidate items we choose from $\mathcal{I}$ with the lowest similarity scores to $\tilde{\boldsymbol{z}} - \epsilon sign(\nabla)$ (instead of the highest); (3) We repeat (1) and (2) to generate the poisoning data; details can be found in Algorithm 2.

Note that though the alternating pattern of co-visitation seems detectable, we can control the proportion of target items (apply noise or add 'user-like' generated data) to avoid this rigid pattern and make it less noticeable. Gradient information from the white-box model also empowers more tailored sequential-recommendation poisoning methods.

5 EXPERIMENTS

5.1 Setup

5.1.1 Dataset. We use three popular recommendation datasets (see Table 1) to evaluate our methods: Movielens-1M (ML-1M) [10], Steam [27] and Amazon Beauty [29]. We follow the preprocessing in BERT4Rec [37] to process the rating data into implicit feedback. We follow SASRec [17] and BERT4Rec [37] to hold out the last two items in each sequence for validation and testing, and the rest for training. We set black-box API returns to be top-100 recommended items.

5.1.2 Model. To evaluate the performance of our attack, we implement a model extraction attack on three representative sequential recommenders, including our PyTorch implementations of NARM [25], BERT4Rec [37] and SASRec [17], with different basic blocks and training schemata, shown in Table 2.

- *NARM* is an attentive recommender, containing an embedding layer, gated recurrent unit (GRU) [3] as a global and local encoder, an attention module to compute session features and a similarity layer, which outputs the most similar items to the session features as recommendations [25].
- *SASRec* consists of an embedding layer that includes both the item embedding and the positional embedding of an input sequence as well as a stack of one-directional transformer (TRM) layers, where each transformer layer contains a multi-head self-attention module and a position-wise feed-forward network [17].
- *BERT4Rec* has an architecture similar to *SASRec*, but using a bidirectional transformer and an auto-encoding (masked language modeling) task for training [37].

5.1.3 Implementation Details. Given a user sequence $\boldsymbol{x}$ with length T, we follow [17, 37] to use $\boldsymbol{x}_{[:T-2]}$ as training data and use the last two items for validation and testing respectively. We use hyper-parameters from grid-search and suggestions from original papers [16, 17, 25]. For reproducibility, we summarize important training configurations in Table 3. Additionally, all models are trained using Adam [19] optimizer with weight decay 0.01, learning rate 0.001, batch size 128 and 100 linear warmup steps. We follow [17, 37] to set allowed sequence lengths of ML-1M, Steam and Beauty as $\{200, 50, 50\}$ respectively, which are also applied as our generated sequence lengths. We follow [39] to use 1% of a real profile's size as a poisoning profile size. Code and data are released[2].

5.1.4 Evaluation Protocol. We follow SASRec [17] to accelerate evaluation by uniformly sampling 100 negative items for each user. Then we rank them with the positive item and report the average

[2]https://github.com/Yueeeeeeee/RecSys-Extraction-Attack

Table 3: Configurations. N:NARM, S:SASRec, B:Bert4Rec, ly:layer, h:attention head, dr:dropout rate, mp:masking probability.

Phase	Model	Config. on {ML-1M, Steam, Beauty}	Phase	Config. on {ML-1M, Steam, Beauty}
Black-box Training	N	GRU ly=1; dr={0.1, 0.2, 0.5}	**Model Extraction**	λ_1, λ_2: {(0.75, 1.5), (0.5, 1.0), (0.5, 0.5)}
	S	TRM ly=2; h=2; dr={0.1, 0.2, 0.5}	**Profile Pollution**	Append items: {10, 2, 2}, ϵ = 1.0, n = 10
	B	TRM ly=2; h=2; dr={0.1, 0.2, 0.5}; mp={0.2, 0.2, 0.6}	**Data Poisoning**	Injected users: {60, 3345, 402}, ϵ = 1.0, n = 10

performance on these 101 testing items. Our Evaluation focuses on two aspects:

- **Ranking Performance**: We to use truncated *Recall@K* that is equivalent to Hit Rate (*HR@K*) in our evaluation, and Normalized Discounted Cumulative Gain (*NDCG@K*) to measure the ranking quality following SASRec [17] and BERT4Rec [37], higher is better.
- **Agreement Measure**: We define *Agreement@K* (*Agr@K*) to evaluate the output similarity between the black-box model and our extracted white-box model:

$$Agreement@K = \frac{|B_{\text{topK}} \cap W_{\text{topK}}|}{K}, \tag{5}$$

where B_{topK} is the top-K predicted list from the black-box model and W_{topK} is from our white-box model. We report average *Agr@K* with $K = 1, 10$ to measure the output similarity.

5.2 RQ1: Can We Extract Model Weights without Real Data?

5.2.1 Standard Model Extraction. We evaluate two ways (*random* and *autoregressive* as mentioned in Section 4.1.1) of generating synthetic data with labels for model extraction. In standard model extraction, we assume the model architecture is known, so that the white-box model uses the same architecture as the black-box model (e.g. SASRec→SASRec) without real training data. We report results with a fixed budgets $B = 5000$ in Table 4.

Observations. From Table 4 we have a few observations: (1) Interestingly, without a training set, *random* and *autoregressive* can achieve similar ranking performance (N@10 and R@10) as the *black-box*. For example, compared with *black-box* NARM on ML-1M, R@10 for *random* drops 1.34% and *autoregressive* drops only 0.98%. On average, extracted recommenders' R@10 is about 94.54% of the original. Particularly, *random* is trained on random data, but labels are retrieved from *black-box* model, reflecting correct last-click relations. Last clicks help *random* rank well, but agr@K is much poorer than *autoregressive* (see Table 4).

(2) *Autoregressive* has significant advantages in narrowing the distance between the two recommenders in all datasets, with an average Agr@10 of 0.574 against 0.452 from randomly generated data. (3) Figure 4a shows *autoregressive* resembles the true training data distribution much better than *random*, because *autoregressive* generates data following interactions with recommended items. Although sampling from the popularity distribution can also resemble the original data distribution, it breaks the assumption that we have no knowledge about the training set, and cannot capture similarities from sequential dependencies. (4) We also note that the datasets have large differences in the distillation process. For example, relatively dense datasets with many user interactions like ML-1M and Steam increase the probability of correct recommendations. Extracted recommenders based on such data distributions sustain a high level of similarity with respect to the black-box output distribution, while in sparser data, it could lead to problems like higher divergence and worse recommendation agreement, which we will examine in the next subsection. (5) Moreover, Table 4 indicates that NARM has the best overall capability of extracting black-box models, as NARM is able to recover most of the black-box models and both its similarity and recommendation metrics are among the highest. As for SASRec and BERT4Rec, both architectures show satisfactory extraction results on the ML-1M and Steam datasets, with SASRec showing slight improvements compared to BERT4Rec in most cases.

5.2.2 Cross Model Extraction. Based on the analysis of different architectures, a natural question follows: which model would perform the best on a different black-box architecture? In this setting, we adopt the same budget and conduct cross-extraction experiments to find out how a white-box recommender would differ in terms of similarity when distilling a different black-box architecture. Cross-architecture model extraction is evaluated on these three datasets.

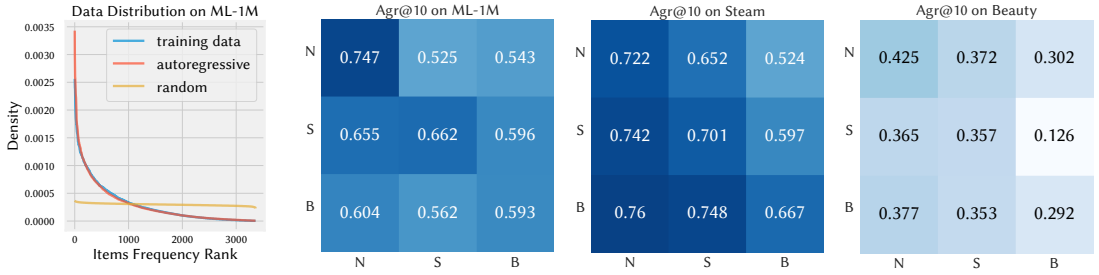
Observations. Results are visualized with heatmaps in Figure 4b, where the horizontal / vertical axes represent white-box / black-box architectures. The NARM model performs the best overall as a white-box architecture, successfully reproducing most target recommender systems with an average of 0.597 agreement in the top-10 recommendations, compared to 0.548 of SASRec and 0.471 of BERT4Rec.

5.3 RQ2: How do Dataset Sparsity and Budget Influence Model Extraction?

Data Sparsity. In the previous experiments, we notice that the sparsity of the original dataset on which the black-box is trained might influence the quality of our distilled model, suggested by the results on the Beauty dataset from Table 4. For the sparsity problem, we base a series of experiments on the most sparse dataset (Beauty); all three models are used to study whether model extraction performance deterioration is related to the dataset. We choose slightly different preprocessing techniques to build a k-core Beauty dataset (i.e. densify interactions until item / user frequencies are both $\geq k$). The processed k-core datasets are denser with increasing k. Our item candidate size (100 negatives) in evaluation does not change. Black-box models are trained on such processed data followed by autoregressive extraction experiments with a 5000 sequence budget in Table 5. As sparsity drops, compared to the 5-core Beauty data, both black-box and extracted models perform better, where the increasing Agr@10 indicates that the extracted models become more 'similar' to the black-box model. Our results indicate that

Table 4: Extraction performance under identical model architecture and 5k budget, with Black-box original performance.

		Black-Box		White-Box-Random				White-Box-Autoregressive			
Dataset	Option	N@10	R@10	N@10	R@10	Agr@1	Agr@10	N@10	R@10	Agr@1	Agr@10
ML-1M	NARM	**0.625**	**0.820**	0.598	0.809	0.389	0.605	0.615	0.812	**0.571**	**0.747**
	SASRec	**0.625**	**0.817**	0.578	0.796	0.270	0.516	0.602	0.802	**0.454**	**0.662**
	BERT4Rec	**0.602**	**0.806**	0.565	0.794	0.241	0.488	0.571	0.791	**0.339**	**0.593**
Steam	NARM	**0.628**	0.848	0.625	**0.849**	0.679	0.642	0.601	0.806	**0.743**	**0.722**
	SASRec	**0.627**	**0.850**	0.579	0.802	0.434	0.556	0.593	0.805	**0.668**	**0.702**
	BERT4Rec	**0.622**	**0.846**	0.609	0.838	0.199	0.490	0.585	0.793	**0.708**	**0.667**
Beauty	NARM	**0.356**	**0.518**	0.319	0.477	**0.356**	**0.511**	0.272	0.380	0.344	0.425
	SASRec	0.344	0.494	0.304	0.459	0.251	0.213	**0.347**	**0.505**	**0.343**	**0.357**
	BERT4Rec	**0.349**	**0.515**	0.200	0.352	0.026	0.043	0.300	0.454	**0.178**	**0.291**



(a) Data Distribution for original and generated data.

(b) Cross model extraction results. Horizontal / vertical axes represent white-box / black-box model architectures. Darker colors represents larger values of Agr@10.

Figure 4: Data distributions and cross model extraction results.

Table 5: Influence of data sparsity. Model extraction on k-core Beauty

		Black-Box		Ours		
Model	k-core	N@10	R@10	N@10	R@10	Agr@10
NARM	5	0.360	0.536	0.331	0.488	0.559
	6	0.372	0.562	0.352	0.539	0.614
	7	0.386	0.630	0.369	0.597	**0.726**
	8	0.347	0.597	0.362	0.643	0.690
SASRec	5	0.351	0.514	0.332	0.479	0.651
	6	0.380	0.558	0.373	0.547	0.744
	7	0.424	0.640	0.427	0.648	0.782
	8	0.415	0.675	0.410	0.672	**0.791**
BERT4Rec	5	0.346	0.509	0.351	0.520	0.561
	6	0.366	0.547	0.374	0.555	0.652
	7	0.402	0.643	0.399	0.650	0.682
	8	0.403	0.694	0.383	0.659	**0.717**

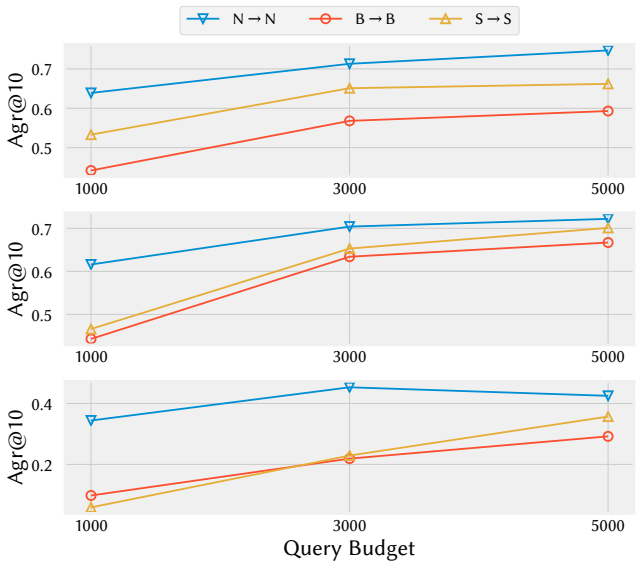


Figure 5: Influence of query budgets on ML-1M (top), Steam (middle) and Beauty (bottom).

data sparisty is an important factor for model extraction performance, where training on denser datasets usually leads to stronger extracted models.

Budget. We also want to find out how important the query budget is for distillation performance. In our framework we assume that the attacker can only query the black-box model a limited number of times (corresponding to a limited budget). Intuitively, a larger budget would induce a larger generated dataset for distillation and lead to a better distilled model with higher similarity and stronger

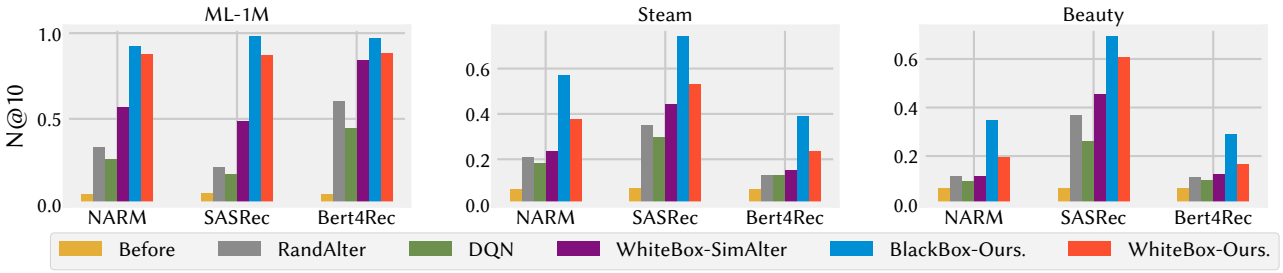


Figure 6: Profile pollution attacks performance comparisons with different methods.

Table 6: Profile pollution attacks to different sequential models for items with different popularity, reported in N@10.

		ML-1M			Steam			Beauty		
Popularity	Attack	NARM	SASRec	BERT4Rec	NARM	SASRec	BERT4Rec	NARM	SASRec	BERT4Rec
head	before	0.202	0.217	0.201	0.313	0.327	0.311	0.261	0.246	0.260
	ours	**0.987**	**0.981**	**0.968**	**0.850**	**0.745**	**0.714**	**0.650**	**0.825**	**0.521**
middle	before	0.037	0.034	0.036	0.012	0.012	0.009	0.023	0.030	0.027
	ours	**0.902**	**0.876**	**0.901**	**0.341**	**0.585**	**0.152**	**0.106**	**0.556**	**0.105**
tail	before	0.005	0.008	0.009	0.000	0.000	**0.000**	**0.002**	0.009	0.002
	ours	**0.701**	**0.760**	**0.760**	**0.017**	**0.160**	**0.000**	0.001	**0.557**	**0.010**

recommendation scores. However, it is preferable to find an optimal budget such that the white-box system is 'close enough' to generate adversarial examples and form threats. Our experiments in Figure 5 suggest that an increasing budget would lead to rapid initial improvements, resulting in a highly similar white-box model. Beyond a certain point, the gains are marginal.

5.4 RQ3: Can We Perform Profile Pollution Attacks using the Extracted Model?

Setup. In profile pollution, we inject adversarial items after original user history and test the corrupted data on the black-box recommender. The average lengths of ML-1M, Steam and Beauty are 166, 13 and 9 respectively; based on this we generate 10 adversarial items for ML-1M and 2 items for Steam and Beauty. We perform profile pollution attacks on all users and present the average metrics in Figure 6. We avoid repeating targets in injection items to rule out trivial solutions like sequence of solely target items. Based on this setting, we introduce the following baseline methods: (1) *RandAlter*: alternating interactions with random items and target item(s) [36]; (2) *Deep Q Learning (DQN)*: naive Q learning model with RNN-based architecture, the rank and number of target item(s) in top-k recommendations are used as training rewards [44, 45]; (3) *WhiteBox SimAlter*: alternating interactions with similar items and target item(s), where similar items could be computed based on similarity of the white-box item embeddings. (4) In addition, we experiment with the black-box recommender as a surrogate model and perform our attacks (*BlackBox-Ours.*).

General Attack Performance Comparisons. In Figure 6, we present the profile pollution performance compared with baselines on three different datasets. (1) Comparing our results with *BlackBox-Ours.* black-box models show the vulnerabilities to pollution attacks from extracted white-box model generally. We notice that the attacking performance of distilled models is comparable in ML-1M, but leads to worsening metrics as datasets become sparser and recommenders become harder to extract, for eaxmple, the average metrics of NARM on ML-1M reach 94.8% attack performance of the black-box model, compared to 66.2% on Steam and 55.6% on Beauty. (2) On all datasets, our method achieves the best targeted item promotion results. For example, on Steam, N@10 scores of the targeted items are significantly improved from 0.070 to 0.381. This shows the benefit of exploiting the surrogate model from model extraction, and our attacking method is designed as an adversarial example method [8], which surpasses heuristic methods that lack knowledge of the victim models. (3) Empricially, we find that the robustness varies for different victim recommenders. For example, results on Steam and Beauty datasets show SASRec is the most vulnerable model under attacks in our experiments. For instance, N@10 of SASRec increases from 0.071 to 0.571 on average on Steam and Beauty datasets, but N@10 of NARM increases from 0.0680 to 0.286.

Items w/ Different Popularities. Table 6 shows our profile pollution attacks to items with different popularities. We group target items using the following rules [1]: *head* denotes the top 20%, *tail* is the bottom 20% and *middle* is the rest according to the item appearance frequency (popularity). From Table 6, our attack method is effective for items with different popularities. But in all scenarios, ranking results after attacking decrease as the popularity of the target item declines; popular items are generally more vulnerable under targeted attacks and could easily be manipulated for gaining attention. Unpopular items, however, are often harder to attack.

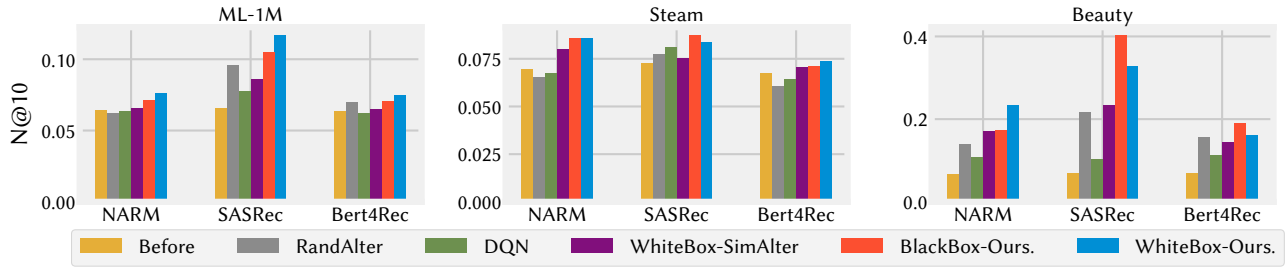


Figure 7: Data poisoning attacks performance comparisons with different methods

Table 7: Data poisoning attacks to different sequential models for items with different popularity, reported as N@10.

		ML-1M			Steam			Beauty		
Popularity	Attack	NARM	SASRec	BERT4Rec	NARM	SASRec	BERT4Rec	NARM	SASRec	BERT4Rec
head	before	0.202	0.217	0.201	0.313	0.327	0.311	0.261	0.246	0.260
	ours	**0.205**	**0.351**	**0.213**	**0.347**	**0.352**	**0.324**	**0.425**	**0.477**	**0.364**
medium	before	0.037	0.034	0.036	0.012	0.012	0.009	0.023	0.030	0.027
	ours	**0.053**	**0.067**	**0.048**	**0.026**	**0.021**	**0.014**	**0.217**	**0.309**	**0.135**
tail	before	0.005	0.008	0.009	0.000	0.000	0.000	0.002	0.009	0.002
	ours	**0.016**	**0.032**	**0.017**	**0.004**	**0.004**	**0.004**	**0.086**	**0.235**	**0.036**

5.5 RQ4: Can We Perform Data Poisoning Attacks using the Extracted Model?

Setup. Different from profile pollution, we do not select a single item as a target and use target groups instead as the attack objective to avoid a large number of similar injection profiles and accelerate retraining. Target selection is identical to profile pollution and we randomly select an attack target from the 25 items in each step during the generation of adversarial profiles. Then, the black-box recommender is retrained once and tested on each of the target items, we present the average results in Figure 7. We follow [39] to generate profiles equivalent to 1% of the number of users from the original dataset and adopt the same baseline methods in profile pollution.

General Attack Performance Comparisons. (1) Our methods surpass other baselines but overall promotion is not as effective as profile pollution. It is because we adopt multiple targets for simultaneous poisoning, and profile pollution attacks specific user with profiles information, where attacking examples can be stronger. (2) Compared to *RandAlter*, the proposed adversarial profile generation further enhances this advantage by connecting the unlikely popular items with targets and magnifying the bias for more exposure of our target items. For example in Beauty, the average N@10 is 0.066 before attack against 0.240 by our method across three models. Moreover, we notice that *DQN* performs worse than in profile pollution and could occasionally lead to performance deterioration. Potential reasons for the performance drop could be: Less frequent appearance of target items against the co-visitation approach; Updated model parameters are independent from the generated fake profiles, as in [39]. (3) Comparable results between *BlackBox-Ours.* and *WhiteBox-Ours.* suggest the bottleneck for data poisoning can be generation algorithm instead of white-box similarity.

Items w/ Different Popularities. Table 7 reveals the poisoning effectiveness as a function of target popularity. In contrast to the numbers in Table 6, the relative improvements are more significant for *middle* and *tail* items. For example, the relative improvement for *head* is 30.8%, compared to over 300% for *middle* items and over 1000% for *tail* items in N@10. The results suggest that data poisoning is more helpful in elevating exposure for less popular items, while the promotion attacks of popular items via profile injection can be harder in this case.

6 CONCLUSION AND FUTURE WORK

In this work, we systematically explore the feasibility and efficacy of stealing and attacking sequential recommender systems with different state-of-the-art architectures. First, our experimental results show that black-box models can be threatened by model extraction attacks. That is, we are able to learn a white-box model which behaves similarly (for instance, with 0.747 top-10 agreement on the ML-1M dataset) to the black-box model without access to training data. This suggests that attacks on the white-box could be transferred to the black-box model. To verify this intuition, we conduct further experiments to study the vulnerability of black-box models using profile pollution and data poisoning attacks. Our experiments show that the performance of a well-trained black-box model can be drastically biased and corrupted under both attacks.

For future work, first, we can extend our framework to more universal settings. In particular, how can we perform model extraction attacks for matrix factorization models or graph-based

models? Second, it would be interesting to develop defense algorithms or other novel robust training pipelines, so that sequential recommender systems could be more robust against adversarial and data poisoning attacks. Third, active learning can be applied to find more effective sampling strategies for model extraction with fewer queries.

REFERENCES

[1] Chris Anderson. 2006. *The long tail: Why the future of business is selling less of more.* Hachette Books.

[2] Robin Burke, Bamshad Mobasher, and Runa Bhaumik. 2005. Limited knowledge shilling attacks in collaborative filtering systems. In *Proceedings of 3rd international workshop on intelligent techniques for web personalization (ITWP 2005), 19th international joint conference on artificial intelligence (IJCAI 2005).* 17–24.

[3] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *EMNLP*.

[4] Konstantina Christakopoulou and Arindam Banerjee. 2019. Adversarial attacks on an oblivious recommender. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 322–330.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://arxiv.org/abs/1810.04805

[6] Minghong Fang, Neil Zhenqiang Gong, and Jia Liu. 2020. Influence function based data poisoning attacks to top-n recommender systems. In *Proceedings of The Web Conference 2020*. 3019–3025.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

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

[9] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. 2014. Shilling attacks against recommender systems: a comprehensive survey. *Artificial Intelligence Review* 42, 4 (2014), 767–799.

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

[11] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 191–200.

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

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

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[15] 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).

[16] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is BERT Really Robust. *A Strong Baseline for Natural Language Attack on Text Classification and Entailment* (2019).

[17] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[18] Sanjay Kariyappa, Atul Prakash, and Moinuddin Qureshi. 2020. MAZE: Data-Free Model Stealing Attack Using Zeroth-Order Gradient Estimation. *arXiv preprint arXiv:2005.03161* (2020).

[19] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Kalpesh Krishna, Gaurav Singh Tomar, Ankur P. Parikh, Nicolas Papernot, and Mohit Iyyer. 2020. Thieves on Sesame Street! Model Extraction of BERT-based APIs. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Byl5NREFDr

[21] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533* (2016).

[22] Shyong K Lam and John Riedl. 2004. Shilling recommender systems for fun and profit. In *Proceedings of the 13th international conference on World Wide Web*. 393–402.

[23] Sungho Lee, Sungjae Hwang, and Sukyoung Ryu. 2017. All about activity injection: Threats, semantics, and detection. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 252–262.

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

[25] 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.

[26] Daniel Lowd and Christopher Meek. 2005. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. 641–647.

[27] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 43–52.

[28] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843* (2016).

[29] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 188–197. https://doi.org/10.18653/v1/D19-1018

[30] Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. 2019. Knockoff nets: Stealing functionality of black-box models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4954–4963.

[31] Soham Pal, Yash Gupta, Aditya Shukla, Aditya Kanade, Shirish Shevade, and Vinod Ganapathy. 2019. A framework for the extraction of deep neural networks by leveraging public data. *arXiv preprint arXiv:1905.09165* (2019).

[32] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. 2017. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 506–519.

[33] Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. 2015. Towards discovering and understanding task hijacking in android. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 945–959.

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

[35] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. 811–820.

[36] 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.

[37] 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.

[38] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 565–573.

[39] Jiaxi Tang, Hongyi Wen, and Ke Wang. 2020. Revisiting Adversarially Learned Injection Attacks Against Recommender Systems. In *Fourteenth ACM Conference on Recommender Systems*. 318–327.

[40] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 601–618.

[41] Xingyu Xing, Wei Meng, Dan Doozan, Alex C Snoeren, Nick Feamster, and Wenke Lee. 2013. Take this personally: Pollution attacks on personalized services. In *22nd {USENIX} Security Symposium ({USENIX} Security 13)*. 671–686.

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

[43] William Zeller and Edward W Felten. 2008. Cross-site request forgeries: Exploitation and prevention. *Bericht, Princeton University* (2008).

[44] Hengtong Zhang, Yaliang Li, Bolin Ding, and Jing Gao. 2020. Practical Data Poisoning Attack against Next-Item Recommendation. In *Proceedings of The Web Conference 2020*. 2458–2464.

[45] Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2017. Deep reinforcement learning for list-wise recommendations. *arXiv preprint arXiv:1801.00209* (2017).

[46] Mingyi Zhou, Jing Wu, Yipeng Liu, Shuaicheng Liu, and Ce Zhu. 2020. DaST: Data-free Substitute Training for Adversarial Attacks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 234–243.