

Defending Substitution-Based Profile Pollution Attacks on Sequential Recommenders

ZHENRUI YUE, University of Illinois Urbana-Champaign, USA

HUIMIN ZENG, University of Illinois Urbana-Champaign, USA

ZIYI KOU, University of Illinois Urbana-Champaign, USA

LANYU SHANG, University of Illinois Urbana-Champaign, USA

DONG WANG, University of Illinois Urbana-Champaign, USA

While sequential recommender systems achieve significant improvements on capturing user dynamics, we argue that sequential recommenders are vulnerable against substitution-based profile pollution attacks. To demonstrate our hypothesis, we propose a substitution-based adversarial attack algorithm, which modifies the input sequence by selecting certain vulnerable elements and substituting them with adversarial items. In both untargeted and targeted attack scenarios, we observe significant performance deterioration using the proposed profile pollution algorithm. Motivated by such observations, we design an efficient adversarial defense method called Dirichlet neighborhood sampling. Specifically, we sample item embeddings from a convex hull constructed by multi-hop neighbors to replace the original items in input sequences. During sampling, a Dirichlet distribution is used to approximate the probability distribution in the neighborhood such that the recommender learns to combat local perturbations. Additionally, we design an adversarial training method tailored for sequential recommender systems. In particular, we represent selected items with one-hot encodings and perform gradient ascent on the encodings to search for the worst case linear combination of item embeddings in training. As such, the embedding function learns robust item representations and the trained recommender is resistant to test-time adversarial examples. Extensive experiments show the effectiveness of both our attack and defense methods, which consistently outperform baselines by a significant margin across model architectures and datasets.

CCS Concepts: • **Information systems** → **Recommender systems**; • **Security and privacy** → **Software and application security**.

ACM Reference Format:

Zhenrui Yue, Huimin Zeng, Ziyi Kou, Lanyu Shang, and Dong Wang. 2022. Defending Substitution-Based Profile Pollution Attacks on Sequential Recommenders. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22)*, September 18–23, 2022, Seattle, WA, USA. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3523227.3546770>

1 INTRODUCTION

Sequential recommenders are a common approach for making personalized recommendations, such recommenders take user interaction history as input and generate potential items that may be of interest to the user [15, 18, 40]. Different from traditional recommenders, sequential recommendation can capture users' evolving dynamics by treating item transition patterns as temporal sequences. As such, various recent sequential recommenders (e.g., Locker[13]) consistently outperform previous state-of-the-art models.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

Nevertheless, the reliability of recommenders is known to deteriorate under adversarial perturbations [9, 36, 41]. While gradient-based methods can not be directly applied on recommenders for searching perturbations in the item space, certain heuristics can be used to perform adversarial attacks [25, 34]. Based on such methods, profile pollution¹ is adopted to perform user-specific adversarial attacks [34, 36]. To perform such attacks, malware can be used to access user profiles and interactions (e.g., cross-site request forgeries (CSRF) or backdoor attacks) [16, 26, 37].

We argue that sequential recommenders have unique vulnerabilities against substitution-based profile pollution attacks. Substitution-based adversarial attacks manipulate vulnerable items in previous or ongoing interactions to sabotage the recommendation (i.e., untargeted attacks) or manipulate the recommended items (i.e., targeted attacks) [16, 20, 26, 37]. Previous works on profile pollution study attack algorithms and have the following limitations: (1) Existing methods designed for traditional recommenders can not be applied, or are not tailored for sequential recommenders [32, 34, 43]; (2) Previous methods do not explore substitution-based attacks, instead, they focus on the injection of adversarial items [36, 43]; and (3) Adversarial defense methods are not discussed [24, 32, 34, 36, 43].

A similar sequential setting can be found in natural language processing (NLP) [5]. NLP adversarial attacks primarily study word-level substitutions, where a small percentage of input words are substituted [1, 17, 27]. To find adversarial examples in NLP, one possible method is to search among synonyms iteratively [1]. Another common approach is to select vulnerable words and replace them with adversarial ones [17, 27]. Despite similarity between NLP and sequential recommendation, it is difficult to build adversarial examples for recommenders using similar techniques. The reasons are two-fold: (1) the item space is too large to enumerate without synonym information in recommender systems; (2) existing NLP attack algorithms do not exploit gradient-based methods to maximize the attack performance.

In this paper, we design a gradient-guided algorithm for substitution-based profile pollution attacks. Our attack algorithm modifies the input sequences by selecting a limited number of vulnerable elements and replacing them with carefully chosen adversarial items. By design, the substitution-based attack can perform either substitution attacks in a sequence, or inject malicious items to user interactions (by inserting and attacking an item, as in [36]). To prevent from being identified as malicious input, we impose two constraints: (1) we restrict the maximum number of substitutions in each sequence; (2) for substitute items, we additionally require similarity to the original items. Our experiment results demonstrate a significant performance deterioration in both untargeted and targeted scenarios, suggesting that the proposed substitution-based attacks pose critical threats to sequential recommenders.

Motivated by the research gap in defense methods, we additionally propose two defense methods that are tailored for sequential recommendation. Dirichlet neighborhood sampling is proposed for efficient training of robust recommenders. Specifically, we construct multi-hop neighborhoods with the embedding matrix and sample augmented item embeddings via Dirichlet distributions to perform random substitutions. Additionally, we design an adversarial training method for sequential recommenders. In particular, we represent items with one-hot encodings and update the encodings to search for the worst case augmentation. By training the recommender on such virtual input, we extend the robustness of the recommender to a larger neighborhood. Our evaluation results show that the proposed defense methods significantly reduce performance variations under profile pollution attacks by over 50% on multiple real-world datasets.

¹In the following, we use adversarial attack and profile pollution attack interchangeably.

2 RELATED WORK

2.1 Adversarial Attacks in Recommender Systems

Adversarial attacks on recommender systems can be categorized into two different classes: (1) *data poisoning* and (2) *profile pollution*. Data poisoning attacks craft fake user profiles and inject such profiles into the training data, causing biased recommendation results upon deployment [4, 6, 14, 29, 32, 42]. Unlike poisoning attacks, profile pollution attacks alter existing user profiles (i.e., user interactions) to manipulate recommended items [22, 33, 34, 36, 43]. Profile pollution can be performed via security breaches like web injection, CSRF attacks or malware [16, 26, 37, 43].

In this work, we focus on profile pollution and perform user-specific attacks to manipulate the recommendation results [36]. CSRF attacks manipulate user history on websites like YouTube under a black-box setting [33]. Malicious requests are exploited to bias web advertisements towards higher-paying advertisers [22]. With knowledge to the recommender, finding adversarial examples can be formulated as an optimization problem [34]. Web injections are used in unprotected websites to promote target items [43]. Yue et al. [36] propose to extract black-box recommenders and compute adversarial examples using the extracted model. Our proposed profile pollution algorithm differs from the previous methods in two aspects: (1) we extend the attack scope to substitutions in the full sequence; and (2) constraints are imposed on adversarial examples to enforce similarity between the clean and adversarial sequences.

2.2 Adversarial Attacks in Language

Given the setting of profile pollution by item substitution, a similar scenario can be found on various NLP classification tasks, where adversarial attacks are studied on word-level substitutions. NLP substitution attacks replace a small percentage of the input words with adversarial words to achieve the attack objective [1, 17]. NLP adversarial examples can be generated using a genetic algorithm, where synonyms are iterated and evaluated [1]. The vulnerability of input words can be evaluated via importance scores, followed by substituting the most vulnerable words to construct adversarial examples until the model is fooled [7, 17, 19, 27]. Despite similarity between language and sequential recommendation, it is difficult to transfer existing NLP methods to recommenders due to: (1) the lack of synonyms (i.e., neighbor items) in recommendation data; and (2) the inefficiency of existing NLP methods for large-scale or real-time attacks. Unlike previous methods, the proposed profile pollution algorithm constructs multi-hop neighborhoods and efficiently computes adversarial substitutes by exploiting gradient information from the recommender model.

2.3 Adversarial Defense Methods

Various defense methods are proposed to enhance the robustness of neural networks [9, 10]. Adversarial training and smoothing methods incorporate augmented examples in training to enhance robustness against perturbations [9, 38, 39]. Interval bound propagation regularizes tractable upper and lower bounds to guarantee robustness [10]. Adversarial training introduced norm-bounded local perturbations to word embeddings to learn robust features [28, 45]. Recently, Dirichlet neighborhood ensemble leverages synonym embeddings to build robust word representations [44]. For recommender systems, adversarial training introduces perturbations to improve model performance and generalization [3, 12, 35]. Multimedia recommender learns to defend perturbations on product images via adversarial training [31]. To the best of our knowledge, no previous methods attempt to enhance the robustness of recommender systems against profile pollution attacks. As such, we propose two novel defense methods tailored for sequential recommenders and show that model robustness can be significantly improved using the proposed defense methods.

3 PRELIMINARIES

3.1 Setting

3.1.1 *Data.* Our framework is based on sequential input, which takes user interaction history \mathbf{x} (sorted by timestamps) as input. \mathbf{x} represents input $[x_1, x_2, \dots, x_l]$ of length l , with each element represented in the item scope \mathcal{I} (i.e., $x_i \in \mathcal{I}$). The next item interaction $x_{l+1} \in \mathcal{I}$ after input \mathbf{x} is used as ground truth y (i.e., $y = x_{l+1}$).

3.1.2 *Model.* We denote the sequential recommender with function f . Given input sequence \mathbf{x} , f predicts a probability distribution over the item scope \mathcal{I} . f comprises of an embedding function f_e and a sequential model f_m , with $f(\mathbf{x}) = f_m(f_e(\mathbf{x}))$. For data pair (\mathbf{x}, y) , ideally, f predicts y with the highest probability (i.e., $y = \arg \max f(\mathbf{x})$).

3.1.3 *Optimization.* The learning of a sequential recommender f is to maximize the probability of output item y upon input \mathbf{x} . In other words, we minimize the expectation of loss \mathcal{L} w.r.t. f over data distribution \mathcal{X} :

$$\min_f \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{X}} \mathcal{L}(f(\mathbf{x}), y), \quad (1)$$

where \mathcal{L} represents the training loss function (i.e., ranking loss or cross entropy loss).

3.2 Adversarial Attacks and Defense

3.2.1 *Assumptions.* We first introduce the assumptions of our framework in the attack scenario, we formalize the problem setting with the following assumptions to define our research scope:

- *Data Access:* Evaluation data \mathcal{X} and item scope \mathcal{I} are accessible to the attacker to perform the attacks.
- *White-box Attacks:* We assume weights of f can be accessed by the attacker to compute adversarial examples.
- *Limited Substitutions:* In each input sequence, we impose a maximum number of substitutions z to enforce the similarity between the original and adversarial examples on the sequence level.
- *Constraints on Item Similarity:* To avoid being identified as outlier, we impose a similarity constraint with a minimum cosine similarity τ between the original items and the adversarial items in the embedding space.

The above assumptions are made for a direct and undisturbed evaluation of the proposed profile pollution attack algorithm (i.e., without considering external influences like model extraction). The constructed adversarial attacks are also more challenging and thus, evaluate the effectiveness of the proposed defense methods under harsh conditions. For the defense propose, we assume full access to the data and recommender models.

3.2.2 *Threat Model.* As described, we adopt profile pollution attacks to construct adversarial examples. We investigate untargeted attacks (i.e., demotion) and targeted attacks (i.e., promotion). We formulate both attacks below.

- *Untargeted Attack:* Untargeted adversarial examples are constructed to minimize the probability w.r.t. ground truth y . From the assumptions we require: (1) The number of substitutions is constrained by Hamming distance (i.e., at most z substitutions); (2) Minimum cosine similarity τ is required between original and adversarial items. We denote the adversarial example with \mathbf{x}' and formulate the attack as an optimization problem w.r.t. \mathbf{x} :

$$\mathbf{x}' = \arg \max_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}), y) \quad s.t. \quad \text{Hamming}(\mathbf{x}, \mathbf{x}') \leq z, \quad \text{CosSim}(x_i, x'_i) \geq \tau, \quad i = 1, 2, \dots \quad (2)$$

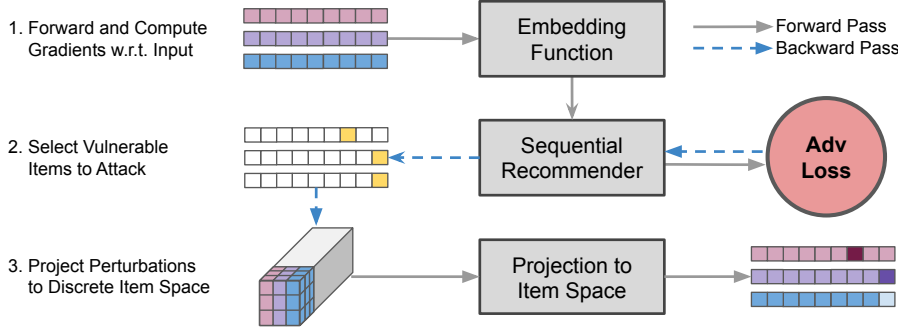


Fig. 1. The proposed profile pollution attack. We first forward clean input sequences and compute gradients w.r.t. the input with the adversarial loss. Then, we compute importance scores to select vulnerable items in input sequences. In the last step, we project the continuous perturbations of the selected items to the discrete item space based on cosine similarity.

- **Targeted Attack:** Targeted adversarial examples aim to maximize the probability w.r.t. target item t . Specifically, we minimize the loss w.r.t. input \mathbf{x} . Similar constraints are imposed for targeted attacks:

$$\mathbf{x}' = \arg \min_{\mathbf{x}} \mathcal{L}(f(\mathbf{x}), t) \quad s.t. \quad \text{Hamming}(\mathbf{x}, \mathbf{x}') \leq z, \quad \text{CosSim}(x_i, x'_i) \geq \tau, \quad i = 1, 2, \dots \quad (3)$$

3.2.3 Adversarial Training. Given the profile pollution attack framework, a robust recommender can be built by minimizing the expected loss upon perturbed input data (i.e., adversarial training). That is, we alternate between optimizing model parameters in f and updating adversarial data based on \mathbf{x} to search for an equilibrium. The adversarial examples are generated on the fly during training and being used to minimize the training loss \mathcal{L} , such that the model f is forced to adjust parameters to resist adversarial perturbations. Formally, we define the adversarial training as a minimax optimization problem w.r.t. model f :

$$\min_f \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{X}} \left[\max_{\mathbf{x}'} \mathcal{L}(f(\mathbf{x}'), y) \right]. \quad (4)$$

4 METHODOLOGY

4.1 Profile Pollution Attack

The proposed algorithm can either attack history items or insert adversarial items to ‘fool’ recommenders. Ideally, the polluted sequences result in either performance degrade (i.e., untargeted attack) or increased exposure of the target item (i.e., targeted attack). We illustrate the proposed profile pollution attack algorithm in Figure 1, the detailed steps for both untargeted and targeted attacks are described as in Algorithm 1.

Step 1: Forward and Compute Gradients w.r.t. Input. Given input \mathbf{x} , we construct the polluted sequence \mathbf{x}' by substituting selected items in \mathbf{x} . First, we initialize \mathbf{x}' unchanged to \mathbf{x} , followed by computing the embedded sequence $\tilde{\mathbf{x}}'$ using the embedding function f_e . We then calculate the gradients of $\tilde{\mathbf{x}}'$ to construct adversarial perturbations in the embedding space. In the case of targeted attack, $\tilde{\mathbf{x}}'$ is fed through the recommender function f_m to compute the cross entropy loss \mathcal{L}_{ce} w.r.t. target item t . Then, backward propagation is performed to retrieve the gradients $\nabla_{\tilde{\mathbf{x}}'}$ of $\tilde{\mathbf{x}}'$ (i.e. $\nabla_{\tilde{\mathbf{x}}'} = \nabla_{\tilde{\mathbf{x}}'} \mathcal{L}_{ce}(f_m(\tilde{\mathbf{x}}'), t)$). For untargeted attacks, we compute the cross entropy loss w.r.t. the predicted item in the output distribution. The gradients are negated in untargeted attacks to perform gradient ascent.

Algorithm 1: Profile Pollution Attack Algorithm

```

1 Input Sequence  $\mathbf{x}$ , target  $t$ , recommender  $f$  (i.e.,  $f_e, f_m$ ), max substitutions  $z$ , similarity threshold  $\tau$ ;
2 Input Target  $t$  if targeted attack;
3 Output Polluted sequence  $\mathbf{x}'$ ;
  // Step 1: Forward and Compute Gradients
4 Initialize polluted sequence:  $\mathbf{x}' \leftarrow \mathbf{x}$ ;
5 Compute sequence embeddings:  $\tilde{\mathbf{x}}' \leftarrow f_e(\mathbf{x}')$ ;
6 if targeted then
7   | Compute gradients w.r.t.  $\tilde{\mathbf{x}}'$ :  $\nabla_{\tilde{\mathbf{x}}'} \leftarrow \nabla_{\tilde{\mathbf{x}}'} \mathcal{L}_{ce}(f_m(\tilde{\mathbf{x}}'), t)$ ;
8 else
9   | Compute gradients w.r.t.  $\tilde{\mathbf{x}}'$ :  $\nabla_{\tilde{\mathbf{x}}'} \leftarrow -\nabla_{\tilde{\mathbf{x}}'} \mathcal{L}_{ce}(f_m(\tilde{\mathbf{x}}'), \arg \max(f_m(\tilde{\mathbf{x}}')))$ ;
10 end
  // Step 2: Select Vulnerable Items
11 Compute importance ranking:  $\mathbf{r} \leftarrow \text{argsort}(\|\nabla_{\tilde{\mathbf{x}}'}\|, \text{descending})$ ;
12 Select  $z$  most vulnerable items in  $\mathbf{x}'$ :  $\mathbf{r} \leftarrow \mathbf{r}[:z]$ ;
  // Step 3: Project Attacks to Item Space
13 for  $i \in \mathbf{r}$  do
14   | Compute cosine similarity  $\mathbf{s}$ :  $\mathbf{s} \leftarrow \text{CosSim}(\tilde{\mathbf{x}}'_i - \text{sign}(\nabla_{\tilde{\mathbf{x}}'_i}, f_e(c))) \forall c \in \mathcal{I}$ ;
15   | Impose similarity constraint  $\mathbf{s}_c$ :  $\mathbf{s}_c \leftarrow \mathbb{1}(\text{CosSim}(\tilde{\mathbf{x}}'_i, f_e(c)) \geq \tau) \forall c \in \mathcal{I}$ ;
16   | Update cosine similarity  $\mathbf{s}$ :  $\mathbf{s} \leftarrow \mathbf{s} \odot \mathbf{s}_c$ ;
17   | Replace  $x'_i$  in  $\mathbf{x}'$ :  $x'_i \leftarrow \arg \max(\mathbf{s})$ ;
18 end

```

Step 2: Select Vulnerable Items to Attack. In input sequence, items are of different vulnerability. We select z vulnerable items and perform substitution on such items to achieve the best attack performance. In particular, we calculate importance scores using $\nabla_{\tilde{\mathbf{x}}'}$ from the previous step. Unlike existing works, we take l^2 norm of the last dimension in $\nabla_{\tilde{\mathbf{x}}'}$ as our importance scores (i.e., $\|\nabla_{\tilde{\mathbf{x}}'}\|$), such that the vulnerability of input items can be efficiently estimated with the steepness of the gradients. We select at most z items by choosing the first z items from importance ranking \mathbf{r} . Based on the fast gradient sign method [9], the perturbed embeddings $\tilde{\mathbf{x}}'$ can be computed as follows:

- Untargeted:

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{x}}' - \text{sign}(-\nabla_{\tilde{\mathbf{x}}'} \mathcal{L}_{ce}(f_m(\tilde{\mathbf{x}}'), \arg \max(f_m(\tilde{\mathbf{x}}')))) \quad (5)$$

- Targeted:

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{x}}' - \text{sign}(\nabla_{\tilde{\mathbf{x}}'} \mathcal{L}_{ce}(f_m(\tilde{\mathbf{x}}'), t)), \quad (6)$$

Step 3: Project Attacks to Item Space. In the last step, we project the perturbed embeddings $\tilde{\mathbf{x}}'$ back to the item space. Similar to [36], we compute cosine similarity between $\tilde{\mathbf{x}}'$ and candidate items from \mathcal{I} , items with higher similarity values are favored as adversarial substitutes. The idea behind it is to find closest items in the direction of the constructed adversarial embeddings using cosine similarity. We denote \mathbf{s} as the cosine similarity scores. Here, we impose another constraint to enforce item similarity, where adversarial items are required to have a minimum cosine similarity (above τ) with the original items. We use the indicator function (i.e., $\mathbb{1}$) to select items fulfilling the similarity constraint. \mathbf{s}_c denotes the similarity constraint results and \mathbf{s}_c is element-wise multiplied (i.e., \odot) with \mathbf{s} . The final adversarial can be selected from \mathbf{s} by taking the highest similarity score. To improve the efficiency of the adversarial attack and item projection, the computation of all previous steps is performed batch-wise with matrix multiplication.

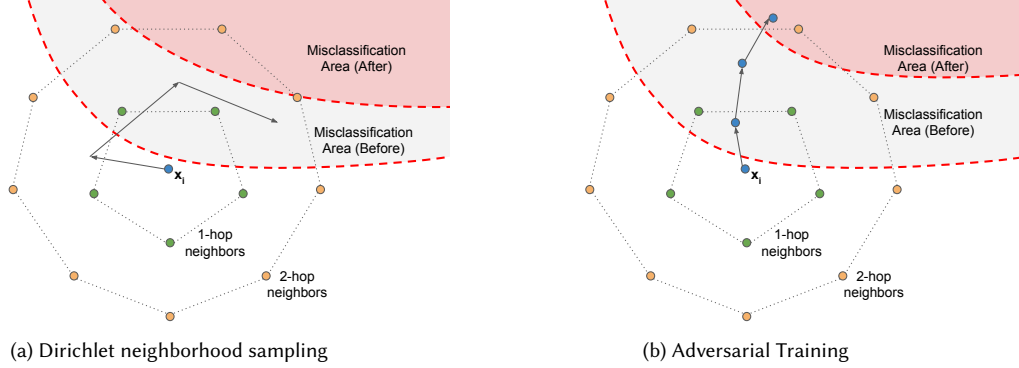


Fig. 2. The proposed defense methods: Dirichlet neighborhood sampling and adversarial training. Blue dots represent the target item, green dots represent 1-hop neighbors and yellow dots represent 2-hop neighbors.

4.2 Dirichlet Neighborhood Sampling

To efficiently train a robust sequential recommender, we first propose Dirichlet neighborhood sampling, a randomized training approach to enhance model robustness. For neighborhood sampling, a connected graph is necessary to identify item neighbors. In NLP, synonym information can be used to compute multi-hop neighbors. However, there exists no natural neighbors or synonyms in recommender systems. Therefore, we propose to construct a neighborhood for each item in item scope \mathcal{I} by computing the cosine similarity between all item pairs. We define item pair (i, j) to be neighbors if item i and item j have a cosine similarity greater or equal to τ (i.e., $\text{CosSim}(i, j) \geq \tau$). In case there exist no neighbors, items with the highest similarity values are used as 1-hop neighbors. Based on the existing 1-hop neighbors, we extend the graph to include 2-hop neighbors and construct a neighborhood dictionary to sample augmented item embeddings in training. The neighborhood dictionary is updated every k epochs (we use $k=10$ in our experiments).

For each item i , we can compute a convex hull in the embedding space spanned by the item’s multi-hop neighbors (i.e., vertices). The convex hull is used as the sampling space for the augmented item embedding. This allow us to sample from a Dirichlet distribution defined by the vertices in the convex hull to construct randomized embeddings. By sampling multiple times in the neighborhood, the recommender is less sensitive to local perturbations and learns to reject adversarial items, as illustrated in Figure 2a. Formally, for item i and the set of its multi-hop neighbors C_i ($C_i = \{c_{i,1}, c_{i,2}, \dots, c_{i,|C_i|}\}$), we represent the augmented item with η_i sampled from a Dirichlet distribution:

$$\eta_i = [\eta_{i,1}, \eta_{i,2}, \dots, \eta_{i,|C_i|}] \sim \text{Dirichlet}(\alpha_1, \alpha_2, \dots, \alpha_{|C_i|}), \quad (7)$$

where α values are hyperparameters for the Dirichlet distribution and can be empirically selected for multi-hop neighbors. Using the sampled η_i , we can compute the augmented item embeddings as follows:

$$f_e(\eta_i) = \sum_{j=1}^{|C_i|} \eta_{i,j} f_e(c_{i,j}). \quad (8)$$

Since Dirichlet distributions are multivariate probability distributions with $\sum \eta_i = 1$ and $\eta_{i,j} \geq 0, j = 1, 2, \dots$, the sampled item is represented as a linear combination of neighbors in the convex hull, see Figure 2a. We denote item i with the blue dot, 1-hop neighbors with green dots and 2-hop neighbors with yellow dots. By adjusting α values, we can change the probability distribution in the sampling space and how far the augmented embedding can travel from

item i . For example, when all α values are 0, the Dirichlet distribution only samples from nodes (i.e., neighbor items). With increasing α values, we can expect the sampled points approaching the center point of the convex hull.

During training, we randomly select and replace items in input sequence \mathbf{x} . Specifically, we select p proportion of the items and sample $\boldsymbol{\eta}$ from a Dirichlet distribution to compute augmented embeddings using the constructed neighborhoods. The virtual sequence is then used to optimize the recommender. Given the increased number of 2-hop neighbors, we select 1.0 as α for 1-hop neighbors and 0.5 for 2-hop neighbors in our experiments [44].

4.3 Adversarial Training with Mixed Representations

While Dirichlet neighborhood sampling is efficient for training sequential recommender, it does not exploit training examples by computing the worst case augmentation. To further improve model robustness, we design an additional defense method called adversarial training with mixed representations. Different from existing adversarial training methods [31, 45] and Dirichlet neighborhood sampling, the proposed method searches for the worst case linear combination of items $\boldsymbol{\eta}$ in \mathcal{I} instead of sampling from the local neighborhood alone. The idea behind searching for item combinations is to introduce mixed representations of the original items and potential adversarial items. Then, we can compute the worst case linear combinations by updating $\boldsymbol{\eta}$ using gradient ascent, such that both the original and adversarial item embeddings are jointly optimized for robust item representations.

For item x_i in input \mathbf{x} , we first initialize x_i with one-hot vector $\boldsymbol{\eta}_i = \text{Onehot}(x_i) \in \mathcal{R}^{|\mathcal{I}|}$. Similar to Dirichlet neighborhood sampling, $f_e(\boldsymbol{\eta}_i)$ is computed as the weighted sum of the embedding matrix (i.e., $f_e(\boldsymbol{\eta}_i) = \sum_{j=1}^{|\mathcal{I}|} \boldsymbol{\eta}_{i,j} f_e(j)$). Next, we perform gradient ascent and compute the gradients w.r.t. $\boldsymbol{\eta}_i$. Specifically, the embedded sequence $\tilde{\mathbf{x}}$ for input \mathbf{x} is first computed using the embedding function f_e . Similar to Section 4.1, we then forward $\tilde{\mathbf{x}}$ using the sequential model f_m and compute the gradients based on the optimization loss to update $\boldsymbol{\eta}_i$:

$$\boldsymbol{\eta}_i = \boldsymbol{\eta}_i + \epsilon \frac{\nabla_{\boldsymbol{\eta}_i} \mathcal{L}(f_m(\tilde{\mathbf{x}}), y)}{\|\nabla_{\boldsymbol{\eta}_i} \mathcal{L}(f_m(\tilde{\mathbf{x}}), y)\|}, \quad (9)$$

where ϵ is the step size. In each update, we normalize the gradients by dividing the gradients by their l^2 norm. Then, step size ϵ is multiplied with the gradients before performing gradient ascent. As such, $\boldsymbol{\eta}_i$ is perturbed towards the misclassification area of the sequential recommender, see Figure 2b. As $\boldsymbol{\eta}_i$ is a linear combination of all items in \mathcal{I} , we require $\sum_j^{|\mathcal{I}|} \boldsymbol{\eta}_{i,j} = 1$ and $\boldsymbol{\eta}_{i,j} \geq 0, j = 1, 2, \dots$ and ensure that the updated $\boldsymbol{\eta}_i$ follows the same requirements within the embedding space. Therefore, we clip negative elements and rescale $\boldsymbol{\eta}_i$ after each update²:

$$\boldsymbol{\eta}_i = \frac{\text{Clip}_{[0,\infty]}(\boldsymbol{\eta}_i, 0, 1)}{\text{Sum}(\text{Clip}_{[0,\infty]}(\boldsymbol{\eta}_i, 0, 1))}. \quad (10)$$

We illustrate the proposed adversarial training with mixed representations in Figure 2b. We denote item i with the blue dot, the paths of the blue dot represent the update iterations of adversarial training, in which the adversarial example moves towards the misclassification area, and thereby forcing the recommender to resist against worst case perturbations. Unlike Dirichlet sampling, where the augmentation directions are sampled from a distribution within a local convex hull, adversarial training finds the worst case embeddings using a linear combination of items to propagate gradients to the item representations. Such worst case augmentation extends the model robustness to a larger area outside the local neighborhood, and therefore effectively reduces the area of misclassification. Compared to Dirichlet neighborhood sampling, adversarial training with mixed representations requires increased computational costs, since adversarial examples are updated multiple times in each iteration. To reduce the training costs, we select p proportion of

²We considered using softmax function but eventually discarded the idea as softmax only produces non-zero $\boldsymbol{\eta}$ values.

Datasets	Users	Items	Avg. len	Sparsity
Beauty	22,363	12,101	9	99.9%
ML-1M	6,040	3,416	165	95.2%
ML-20M	138,493	18,345	144	99.2%
LastFM	988	57,638	9739	83.1%
Steam	334,289	12,012	13	99.9%

Table 1. Dataset details.

Model	Basic Block	Training
NARM [18]	RNN	Autoregressive
SASRec [15]	Transformer	Autoregressive
BERT4Rec [30]	Transformer	Masked Training
Locker [13]	Transformer	Masked Training

Table 2. Model Architectures.

items in each sequence and update η of the selected items three times in our experiments, we optimize the recommender with both clean input and perturbed examples to enhance the model robustness.

5 EXPERIMENTS

5.1 Setup

5.1.1 Dataset. In our experiments, we adopt five datasets to validate the proposed attack and defense methods (Details presented in Table 1). We use Amazon Beauty [23], Movielens (both Movielens 1M and Movielens 20M) [11], LastFM [2] and Steam [21]. We adopt 5-core version of all datasets and preprocess the data following [30, 36].

5.1.2 Model. We select four different recommender architectures to evaluate the proposed methods, see Table 2. In particular, we adopt NARM [18], SASRec [15], BERT4Rec [30] and Locker [13] in our experiments:

- *Neural Attentive Recommendation Machine (NARM)* is a RNN-based sequential recommender comprising of a global encoder and a local encoder. NARM utilizes an attention module to compute item representations and outputs the predictions using a similarity layer [18].
- *Self-Attentive Sequential Recommendation (SASRec)* leverages transformer blocks to forward items autoregressively. Transformer blocks compute attentive representations with one-directional self-attention. SASRec consists of an embedding function, transformer blocks and an output layer [15].
- *Bidirectional Encoder Representations from Transformers for Sequential Recommendation (BERT4Rec)* has a similar architecture to SASRec. BERT4Rec utilizes bidirectional self-attention and adopts masked training (i.e., masked language modeling) to optimize the model [30].
- *Locally Constrained Self-attentive Recommender (Locker)* is similar to BERT4Rec, but proposes additional local constraints to improve self-attention. In our implementation, we use the convolution layers to model local dynamics. Locker is also optimized via masked training [13].

5.1.3 Evaluation. We follow [15, 30, 36] to perform evaluation. We adopt leave-one-out method and use the last two items in each sequence for validation and testing. We adopt normalized discounted cumulative gain $NDCG@10$ and $Recall@10$ as evaluation metrics. In the attack experiments, 15 items are selected as target items across different popularity groups for targeted attacks, we evaluate on the polluted profiles and compute the average results.

5.1.4 Implementation. We follow the original papers to implement sequential recommenders [13, 15, 18, 30]. Unless explicitly mentioned, hyperparameters are from the original works. All models are trained without warmup using Adam optimizer with learning rate of 0.001, weight decay 0.01 and batch size of 64. Similar to [15, 30, 36], we set the maximum sequence lengths of ML-1M to be 200 and 50 for other datasets. For profile pollution, we use $z = 2$ as the maximum substitution for ML-1M and $z = 1$ for other datasets. The minimum cosine similarity of $\tau = 0.5$ is imposed for both attack and defense methods. In our defense experiments, we set substitution probability p to be 0.5 and adopt

Model (U/T)	Dataset	Beauty	ML-1M	ML-20M	LastFM	Steam
	Method	N@10 / R@10	N@10 / R@10	N@10 / R@10	N@10 / R@10	N@10 / R@10
NARM (U)	Before	0.334 / 0.493	0.624 / 0.815	0.770 / 0.940	0.659 / 0.804	0.617 / 0.836
	SimAlter	0.326 / 0.483	0.622 / 0.813	0.769 / 0.939	0.654 / 0.795	0.611 / 0.831
	Ours	0.209 / 0.332	0.154 / 0.326	0.630 / 0.892	0.565 / 0.727	0.437 / 0.714
NARM (T)	Before	0.064 / 0.125	0.074 / 0.150	0.093 / 0.157	0.099 / 0.147	0.088 / 0.141
	SimAlter	0.094 / 0.170	0.084 / 0.169	0.098 / 0.165	0.100 / 0.148	0.101 / 0.158
	Ours	0.239 / 0.420	0.285 / 0.436	0.140 / 0.223	0.262 / 0.407	0.181 / 0.293
SASRec (U)	Before	0.325 / 0.498	0.546 / 0.790	0.762 / 0.953	0.679 / 0.823	0.599 / 0.820
	SimAlter	0.320 / 0.491	0.545 / 0.788	0.760 / 0.952	0.680 / 0.824	0.591 / 0.813
	Ours	0.185 / 0.309	0.144 / 0.299	0.640 / 0.900	0.589 / 0.771	0.407 / 0.641
SASRec (T)	Before	0.072 / 0.139	0.092 / 0.165	0.104 / 0.173	0.103 / 0.152	0.093 / 0.149
	SimAlter	0.184 / 0.312	0.109 / 0.192	0.124 / 0.211	0.145 / 0.231	0.161 / 0.281
	Ours	0.187 / 0.335	0.236 / 0.360	0.139 / 0.226	0.191 / 0.296	0.179 / 0.297
BERT4Rec (U)	Before	0.353 / 0.530	0.542 / 0.744	0.760 / 0.941	0.663 / 0.796	0.574 / 0.795
	SimAlter	0.340 / 0.516	0.540 / 0.742	0.759 / 0.940	0.660 / 0.796	0.563 / 0.788
	Ours	0.215 / 0.349	0.155 / 0.316	0.636 / 0.892	0.637 / 0.778	0.305 / 0.566
BERT4Rec (T)	Before	0.072 / 0.147	0.064 / 0.133	0.086 / 0.155	0.106 / 0.157	0.087 / 0.139
	SimAlter	0.084 / 0.170	0.072 / 0.149	0.088 / 0.156	0.106 / 0.158	0.093 / 0.150
	Ours	0.155 / 0.298	0.245 / 0.395	0.114 / 0.184	0.119 / 0.181	0.144 / 0.247
Locker (U)	Before	0.357 / 0.526	0.602 / 0.796	0.760 / 0.941	0.662 / 0.799	0.631 / 0.852
	SimAlter	0.353 / 0.520	0.601 / 0.796	0.759 / 0.941	0.660 / 0.804	0.627 / 0.852
	Ours	0.248 / 0.389	0.164 / 0.346	0.653 / 0.902	0.617 / 0.768	0.507 / 0.780
Locker (T)	Before	0.065 / 0.130	0.073 / 0.147	0.084 / 0.150	0.104 / 0.157	0.097 / 0.159
	SimAlter	0.133 / 0.242	0.084 / 0.167	0.090 / 0.159	0.109 / 0.166	0.135 / 0.230
	Ours	0.134 / 0.250	0.234 / 0.356	0.107 / 0.180	0.126 / 0.200	0.131 / 0.225

Table 3. Evaluation results of the clean performance and performance under profile pollution attacks. Untargeted attack results are denoted with U (lower the better), targeted attacks are denoted with T (higher the better). Before denotes clean performance, while SimAlter and Ours represent the model performance under attack.

2-hop neighborhood for Dirichlet sampling. To update perturbations in adversarial training, an update step size ϵ from [0.01, 0.1, 1.0] and three adversarial update iterations are used in our implementation³.

5.2 RQ1: Can We Attack Recommenders using Substitution-based Profile Pollution?

We first evaluate the attack performance of the proposed profile pollution algorithm on all model architectures and datasets, the results are presented in Table 3. Since we restrain from substituting more than two items, we adopt SimAlter as the baseline method, which utilizes neighbor items as adversarial items to perform attacks [36]. The table includes several parts: (1) Each row denotes a model name and the setting (i.e., U for untargeted and T for targeted attacks, Before denotes the clean performance); (2) Each column represents the results from one dataset; (3) For targeted attacks, we select a total of 15 items across different popularity groups and report the average results; (4) *Lower results indicate better untargeted attack performance while higher results indicate better targeted attack performance*; and (5) The presented results are in *NDCG@10* and *Recall@10* (i.e., N@10/R@10). Best results are marked in bold.

³The implementation of our framework is publicly available at <https://github.com/Yueeeeeee/RecSys-Substitution-Defense>.

Model (U/T)	Dataset	Beauty	ML-1M	ML-20M	LastFM	Steam
	Method	$\Delta N@10/R@10$	$\Delta N@10/R@10$	$\Delta N@10/R@10$	$\Delta N@10/R@10$	$\Delta N@10/R@10$
NARM (U)	Before	-0.125 / -0.161	-0.470 / -0.489	-0.140 / -0.048	-0.094 / -0.077	-0.180 / -0.122
	Dirichlet	-0.100 / -0.120	-0.366 / -0.335	-0.136 / -0.060	-0.031 / -0.018	-0.160 / -0.098
	AdvTrain	-0.093 / -0.120	-0.331 / -0.314	-0.100 / -0.035	-0.053 / -0.043	-0.159 / -0.102
NARM (T)	Before	0.175 / 0.295	0.211 / 0.286	0.047 / 0.066	0.163 / 0.260	0.093 / 0.152
	Dirichlet	0.127 / 0.215	0.203 / 0.270	0.059 / 0.079	0.040 / 0.078	0.071 / 0.114
	AdvTrain	0.092 / 0.156	0.153 / 0.201	0.020 / 0.024	0.050 / 0.080	0.062 / 0.093
SASRec (U)	Before	-0.140 / -0.189	-0.402 / -0.491	-0.122 / -0.053	-0.090 / -0.052	-0.192 / -0.179
	Dirichlet	-0.127 / -0.180	-0.299 / -0.307	-0.091 / -0.031	-0.038 / -0.030	-0.122 / -0.096
	AdvTrain	-0.093 / -0.121	-0.228 / -0.215	-0.067 / -0.021	-0.031 / -0.017	-0.120 / -0.085
SASRec (T)	Before	0.115 / 0.196	0.144 / 0.195	0.035 / 0.053	0.088 / 0.144	0.086 / 0.148
	Dirichlet	0.089 / 0.148	0.097 / 0.122	0.023 / 0.032	0.027 / 0.051	0.044 / 0.087
	AdvTrain	0.061 / 0.096	0.080 / 0.109	0.021 / 0.035	0.041 / 0.082	0.050 / 0.093
BERT4Rec (U)	Before	-0.138 / -0.181	-0.387 / -0.428	-0.124 / -0.049	-0.026 / -0.018	-0.269 / -0.229
	Dirichlet	-0.071 / -0.095	-0.258 / -0.228	-0.081 / -0.031	-0.011 / -0.009	-0.148 / -0.094
	AdvTrain	-0.119 / -0.152	-0.249 / -0.234	-0.074 / -0.029	-0.006 / -0.008	-0.151 / -0.106
BERT4Rec (T)	Before	0.083 / 0.151	0.181 / 0.262	0.028 / 0.029	0.013 / 0.024	0.057 / 0.108
	Dirichlet	0.041 / 0.069	0.129 / 0.162	0.021 / 0.027	0.005 / 0.010	0.026 / 0.049
	AdvTrain	0.056 / 0.096	0.132 / 0.172	0.020 / 0.019	0.001 / 0.009	0.035 / 0.068
Locker (U)	Before	-0.109 / -0.137	-0.438 / -0.450	-0.107 / -0.039	-0.045 / -0.031	-0.124 / -0.072
	Dirichlet	-0.087 / -0.106	-0.301 / -0.262	-0.093 / -0.034	-0.040 / -0.027	-0.125 / -0.072
	AdvTrain	-0.086 / -0.106	-0.314 / -0.279	-0.084 / -0.031	-0.028 / -0.019	-0.117 / -0.066
Locker (T)	Before	0.069 / 0.120	0.161 / 0.209	0.023 / 0.030	0.022 / 0.043	0.034 / 0.066
	Dirichlet	0.066 / 0.100	0.121 / 0.157	0.021 / 0.026	0.019 / 0.037	0.021 / 0.043
	AdvTrain	0.044 / 0.071	0.131 / 0.166	0.024 / 0.027	0.012 / 0.023	0.015 / 0.022

Table 4. Evaluation results of the defense methods under profile pollution attacks. Untargeted attack results are denoted with U (higher the better), targeted attacks are denoted with T (lower the better). Before denotes evaluation results on the clean model, while Dirichlet and AdvTrain represent the adopted defense methods.

From the results we observe the following: (1) By comparing clean performance (i.e., Before) and metric numbers under attack (i.e., SimAlter and Ours), recommender models demonstrate vulnerability against profile pollution attacks. For example, recommenders suffer from over 50% performance deterioration in untargeted attacks on ML-1M. (2) The proposed method performs the best for both untargeted and targeted attacks, successfully biases the recommender models towards the targets and outperforms the baseline method in all cases with the only exception of Locker on Steam. (3) We observe different vulnerabilities across model architectures and training datasets. For example, we observe RNN-based model (i.e., NARM) to be more vulnerable against targeted attacks with an average increase of over 175% on NDCG@10. On the contrary, Locker only suffers from 82.1% increase in targeted attacks. Additionally, we observe that dense datasets are generally more robust against profile pollution attacks. For instance, we only observe 9.5% performance drop for untargeted attacks on LastFM dataset, while the same setting yields 37.5% deterioration on Beauty. Overall, the results suggest that the proposed algorithm can effectively perform profile pollution attacks.

5.3 RQ2: How the Proposed Defense Methods Perform under Profile Pollution Attacks?

We study the effectiveness of defense methods by evaluating profile pollution attacks on recommenders trained with the defense methods. In particular, we adopt identical training conditions and apply the proposed methods (i.e., Dirichlet neighborhood sampling and adversarial training). Then, we evaluate the recommender performance on clean test data and polluted test data. As each defense method results in a new model, we only compute the **performance changes** (Δ) between clean and polluted evaluation data on the model. Ideally, a robust recommender demonstrates little performance variations, therefore lower absolute values of Δ indicate better defense performance. In other words, higher values indicate better results for untargeted attacks, as model degrades after attack (i.e., Δ is negative). Lower values indicate better results for targeted attacks, as metric values w.r.t. target items increase after attack (i.e., Δ is positive), see Table 4. Similar to Table 3, we separate different settings by model, dataset and attack type. We adopt Δ of $NDCG@10$ and $Recall@10$ as metrics and mark best results in bold.

The defense results indicate the following: (1) Although the recommenders are vulnerable against profile pollution attacks, we find both defense methods to be helpful against polluted test data. For example, the proposed adversarial training reduces $NDCG@10$ deterioration in untargeted attacks by 34.2% on ML-1M. (2) The results indicate that the proposed adversarial training performs the best, achieving the best performance in 30 out of 40 scenarios. Overall, adversarial training reduces $NDCG@10$ variations by 34.4% in untargeted attacks, compared to 28.5% of the proposed Dirichlet neighborhood sampling method. The results suggest adversarial training considerably improves model robustness and can outperform randomized method (i.e., Dirichlet sampling) in most cases. This may be attributed to the generated adversarial examples, which forces the model to reduce reliance on vulnerable features. (3) Defense methods perform differently depending on the recommender and datasets. For example, the proposed adversarial training demonstrate better performance on NARM and SASRec, consistently outperforming Dirichlet neighborhood sampling on four of the five datasets. A potential reason may be the autoregressive training approach, which increases model vulnerability by learning shortcuts and local features [8]. (4) Based on the defense results, LastFM demonstrates the highest robustness on all datasets against both types of attacks, while Locker performs the best across all recommender architectures. This suggest that increasing data density and capturing user dynamics contribute to robustness, possibly by providing more informative item representations and transition patterns. Moreover, learning via autoencoding (i.e., masked training) might improve recommender robustness against adversarial attacks.

Popularity	Popular	Middle	Bottom
	$\Delta N@10 / R@10$	$\Delta N@10 / R@10$	$\Delta N@10 / R@10$
Beauty Before	0.178 / 0.252	0.087 / 0.177	0.067 / 0.143
Beauty AdvTrain	0.126 / 0.177	0.041 / 0.075	0.021 / 0.049
ML-1M Before	0.409 / 0.478	0.057 / 0.118	0.057 / 0.119
ML-1M AdvTrain	0.318 / 0.368	0.037 / 0.081	0.022 / 0.051
ML-20M Before	0.081 / 0.089	0.015 / 0.036	0.004 / 0.009
ML-20M AdvTrain	0.066 / 0.078	0.007 / 0.017	0.001 / 0.003
LastFM Before	0.105 / 0.152	0.058 / 0.107	0.053 / 0.094
LastFM AdvTrain	0.058 / 0.086	0.015 / 0.034	0.010 / 0.025
Steam Before	0.138 / 0.226	0.041 / 0.084	0.023 / 0.047
Steam AdvTrain	0.110 / 0.186	0.010 / 0.029	0.004 / 0.009

Table 5. Averaged results of different popularity groups in targeted attacks.

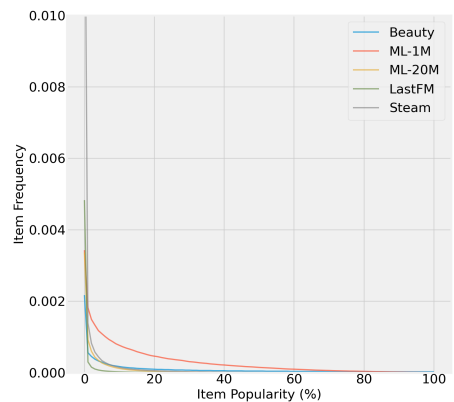


Fig. 3. Distribution of Item Popularity.

Popularity	Performance	Untargeted	Popular	Middle	Bottom
Model	N@10 / R@10	Δ N@10 / R@10	Δ N@10 / R@10	Δ N@10 / R@10	Δ N@10 / R@10
NARM Before	0.601/0.778	-0.202 / -0.179	0.241 / 0.303	0.099 / 0.188	0.073 / 0.144
NARM AdvTrain	0.576/0.766	-0.148 / -0.124	0.187 / 0.238	0.032 / 0.076	0.011 / 0.027
SASRec Before	0.582/0.777	-0.189 / -0.193	0.182 / 0.239	0.063 / 0.127	0.037 / 0.095
SASRec AdvTrain	0.585/0.779	-0.108 / -0.092	0.110 / 0.157	0.027 / 0.055	0.018 / 0.040
BERT4Rec Before	0.578/0.761	-0.189 / -0.181	0.157 / 0.218	0.025 / 0.054	0.036 / 0.072
BERT4Rec AdvTrain	0.579/0.771	-0.121 / -0.105	0.127 / 0.172	0.015 / 0.023	0.010 / 0.022
Locker Before	0.602/0.783	-0.165 / -0.146	0.149 / 0.196	0.020 / 0.048	0.016 / 0.037
Locker AdvTrain	0.593/0.778	-0.129 / -0.102	0.117 / 0.148	0.015 / 0.035	0.007 / 0.018

Table 6. Influence of recommender architecture on profile pollution attacks.

We additionally study the performance variation w.r.t. item popularity. Specifically, we study targeted attacks on sequential recommenders and select target items of different popularity. Target items are divided into three subgroups upon their popularity (i.e., number of total occurrences) in the dataset. *Popular* items denote the $\frac{1}{3}$ most popular items within the item scope. *Middle* and *Bottom* items refer to the next $\frac{1}{3}$ and the last $\frac{1}{3}$ of the items in \mathcal{I} . We present the item distribution of all datasets in Figure 3, we present profile pollution results of the three popularity subgroups averaged across all recommender models. Similarly, performance variations are report with Δ of $NDCG@10$ and $Recall@10$ in Table 5. We observe the following: (1) Popular items are of the highest vulnerability with Δ N@10 of 0.182 before attack, showing the largest performance variations in all cases; (2) Middle and bottom items are generally ‘harder’ to attack. For example, middle items has a much lower Δ N@10 of 0.051 before attack. (3) Popular items are also ‘harder’ to defend, with 27.0% reduced Δ N@10 after defense, compared to 58.2% and 73.8% of middle and bottom items. The results suggest that popular items exhibit the highest vulnerability against profile pollution attacks, a potential reason for such vulnerability is the higher number of neighbors, which increases the number of possible adversarial substitutes.

5.4 RQ3: How Recommender Performance and Robustness Change with Different Model Architectures?

We study the vulnerability of different recommender models and performance of the proposed defense methods. In particular, we study the attacks on all datasets and compare the original performance and performance changes of different recommender architectures under profile pollution attacks. We present the averaged results of each recommender architecture in Table 6 for a direct comparison. The visualized results are presented in Figure 4a and Figure 4b. Each sub-graph represents a recommender model, where the results on all datasets with different defense methods are compared and visualized with bars. As $NDCG@10$ and $Recall@10$ provide similar trends, we visualize the recall results in the graphs. B, M-1, M-20, L, S represent Beauty, ML-1M, ML-20M, LastFM and Steam datasets.

Here are our observations: (1) We observe that adversarial training has limited influence on recommendation performance. For example, our results on transformer-based models demonstrate slight performance changes, possibly due to the improved item representations. (2) By comparing performance variations before and after profile pollution attacks in Table 6, we observe that NARM is more vulnerable and demonstrates higher performance variations both before and after applying the defense methods. (3) Locker demonstrates the highest robustness on average. Without adopting defense methods, we observe 27.4% $NDCG@10$ variation on Locker in untargeted attacks, compared to 32.7%, 32.4% and 33.6% of BERT4Rec, SASRec and NARM. (4) Although SASRec and BERT4Rec have similar transformer

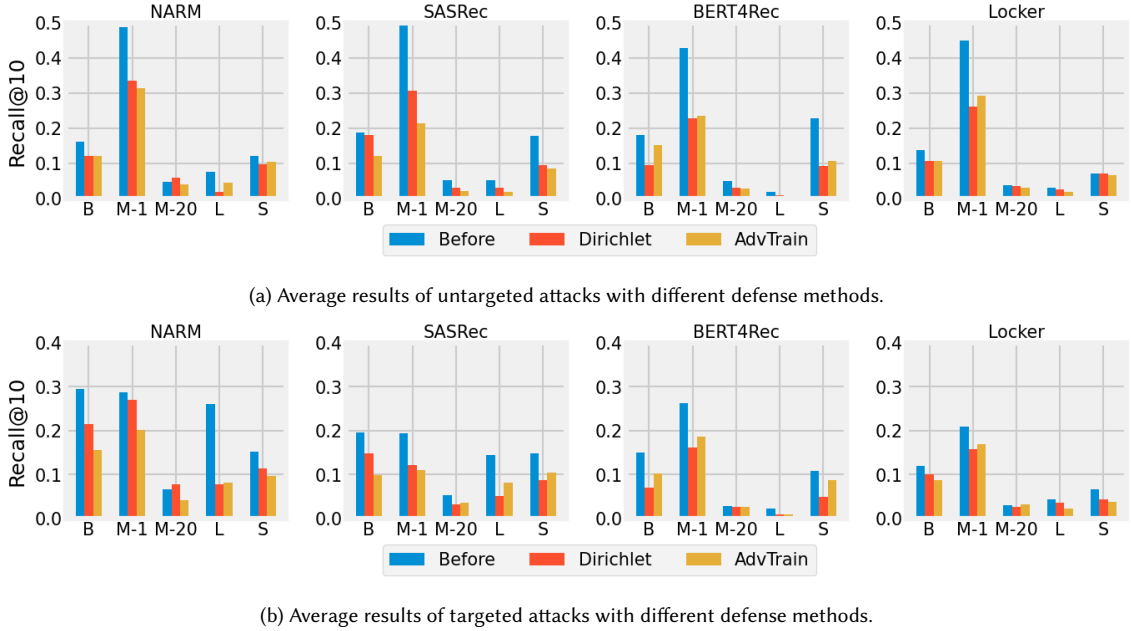


Fig. 4. Visualization of profile pollution performance on different recommenders / datasets. B, M-1, M-20, L, S represent Beauty, ML-1M, ML-20M, LastFM and Steam respectively.

blocks to Locker, SASRec and BERT4Rec are more vulnerable and suffer higher performance deterioration even when adversarially trained. For instance, targeted attacks on popular items increase NDCG@10 variation by 31.3% on SASRec and 27.2% on BERT4Rec, compared to only 24.8% of Locker. Overall, transformer-based sequential recommenders (i.e., SASRec, BERT4Rec and Locker) demonstrate improved robustness of similar magnitude compared to RNN-based model (i.e., NARM). Comparable trends can be observed when we apply adversarial training on sequential recommenders. The reason for such robustness may be traced back to the introduction of global attention in transformer models. Additionally, BERT4Rec and Locker utilize masked training to improve the capture of global context. In contrast, NARM relies on local temporal patterns and can be more sensitive to substitution-based perturbations.

6 CONCLUSION AND FUTURE WORK

In this work, we study the effectiveness of substitution-based profile pollution attacks on sequential recommenders. We design a profile pollution attack algorithm and evaluate on multiple datasets and state-of-the-art recommenders, where the proposed method successfully poses threats to sequential recommenders for both the untargeted and targeted attack scenarios. Then, we explore the defense methods by sampling augmented item representations from a Dirichlet distribution within multi-hop neighborhoods. We additionally design an adversarial training method to search for the worst-case augmentation and enhance model robustness. To the best of our knowledge, this is the first work on defending substitution-based profile pollution attacks for sequential recommenders. Our experimental results suggest that by applying the proposed defense methods, sequential recommender systems can learn robust item representations and demonstrate significantly reduced performance variations under profile pollution attacks.

The proposed methods have certain limitations. For example, our attack and defense methods are designed for sequential recommenders and might not be applicable for other recommender systems. Additionally, the profile pollution experiments are performed under a white-box assumption, rendering the proposed method less accessible in real-world scenarios. Despite having introduced Dirichlet neighborhood sampling, we have not discussed different choices of α values for multi-hop neighbors to exploit the potential benefits of the Dirichlet distribution. For future work, we plan to relax our assumptions and explore certified defense methods for sequential recommenders. For example, it would be interesting to explore interval bound propagation to provide guaranteed defense. Additionally, we can design model-agnostic adversarial attack and defense algorithms to extend our framework to other architectures.

ACKNOWLEDGMENTS

This research is supported in part by the National Science Foundation under Grant No. IIS-2202481, CHE-2105032, IIS-2008228, CNS-1845639, CNS-1831669. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

- [1] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998* (2018).
- [2] O. Celma. 2010. *Music Recommendation and Discovery in the Long Tail*. Springer.
- [3] Huiyuan Chen and Jing Li. 2019. Adversarial tensor factorization for context-aware recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 363–367.
- [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] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. 2017. Hotflip: White-box adversarial examples for text classification. *arXiv preprint arXiv:1712.06751* (2017).
- [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] Siddhant Garg and Goutham Ramakrishnan. 2020. Bae: Bert-based adversarial examples for text classification. *arXiv preprint arXiv:2004.01970* (2020).
- [8] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. 2020. Shortcut learning in deep neural networks. *Nature Machine Intelligence* 2, 11 (2020), 665–673.
- [9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).
- [10] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Relja Arandjelovic, Timothy Mann, and Pushmeet Kohli. 2018. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715* (2018).
- [11] 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.
- [12] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial personalized ranking for recommendation. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 355–364.
- [13] Zhankui He, Handong Zhao, Zhe Lin, Zhaowen Wang, Ajinkya Kale, and Julian McAuley. 2021. Locker: Locally Constrained Self-Attentive Sequential Recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3088–3092.
- [14] 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).
- [15] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.
- [16] Sunggho 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.

- [17] Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *arXiv preprint arXiv:1812.05271* (2018).
- [18] 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.
- [19] Linyang Li, Ruotian Ma, Qipeng Guo, Xiangyang Xue, and Xipeng Qiu. 2020. Bert-attack: Adversarial attack against bert using bert. *arXiv preprint arXiv:2004.09984* (2020).
- [20] Jermaine Marshall and Dong Wang. 2016. Mood-sensitive truth discovery for reliable recommendation systems in social sensing. In *Proceedings of the 10th ACM conference on recommender systems*. 167–174.
- [21] 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.
- [22] Wei Meng, Xinyu Xing, Anmol Sheth, Udi Weinsberg, and Wenke Lee. 2014. Your online interests: Pwned! a pollution attack against targeted advertising. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 129–140.
- [23] 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>
- [24] Sejoon Oh and Srijan Kumar. 2022. Robustness of Deep Recommendation Systems to Untargeted Interaction Perturbations. *arXiv preprint arXiv:2201.12686* (2022).
- [25] Michael P O’Mahony, Neil J Hurley, and Guérolé CM Silvestre. 2005. Recommender systems: Attack types and strategies. In *AAAI*. 334–339.
- [26] 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.
- [27] Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*. 1085–1097.
- [28] Motoki Sato, Jun Suzuki, Hiroyuki Shindo, and Yuji Matsumoto. 2018. Interpretable adversarial perturbation in input embedding space for text. *arXiv preprint arXiv:1805.02917* (2018).
- [29] 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.
- [30] 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.
- [31] Jinhui Tang, Xiaoyu Du, Xiangnan He, Fajie Yuan, Qi Tian, and Tat-Seng Chua. 2019. Adversarial training towards robust multimedia recommender system. *IEEE Transactions on Knowledge and Data Engineering* 32, 5 (2019), 855–867.
- [32] Jiayi Tang, Hongyi Wen, and Ke Wang. 2020. Revisiting Adversarially Learned Injection Attacks Against Recommender Systems. In *Fourteenth ACM Conference on Recommender Systems*. 318–327.
- [33] 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.
- [34] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. 2017. Fake Co-visitation Injection Attacks to Recommender Systems.. In *NDSS*.
- [35] Feng Yuan, Lina Yao, and Boualem Benatallah. 2019. Adversarial collaborative neural network for robust recommendation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1065–1068.
- [36] 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*. 44–54.
- [37] William Zeller and Edward W Felten. 2008. Cross-site request forgeries: Exploitation and prevention. *Bericht, Princeton University* (2008).
- [38] Huimin Zeng, Jiahao Su, and Furong Huang. 2021. Certified Defense via Latent Space Randomized Smoothing with Orthogonal Encoders. *arXiv preprint arXiv:2108.00491* (2021).
- [39] Huimin Zeng, Chen Zhu, Tom Goldstein, and Furong Huang. 2021. Are adversarial examples created equal? A learnable weighted minimax risk for robustness under non-uniform attacks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 10815–10823.
- [40] Daniel Zhang, Bo Ni, Qiyu Zhi, Thomas Plummer, Qi Li, Hao Zheng, Qingkai Zeng, Yang Zhang, and Dong Wang. 2019. Through the eyes of a poet: Classical poetry recommendation with visual input on social media. In *2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 333–340.
- [41] Daniel Yue Zhang, Qi Li, Herman Tong, Jose Badilla, Yang Zhang, and Dong Wang. 2018. Crowdsourcing-based copyright infringement detection in live video streams. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 367–374.
- [42] 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.
- [43] Yubao Zhang, Jidong Xiao, Shuai Hao, Haining Wang, Sencun Zhu, and Sushil Jajodia. 2019. Understanding the manipulation on recommender systems through web injection. *IEEE Transactions on Information Forensics and Security* 15 (2019), 3807–3818.
- [44] Yi Zhou, Xiaoqing Zheng, Cho-Jui Hsieh, Kai-wei Chang, and Xuanjing Huang. 2020. Defense against adversarial attacks in nlp via dirichlet neighborhood ensemble. *arXiv preprint arXiv:2006.11627* (2020).

- [45] Chen Zhu, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2019. Freelb: Enhanced adversarial training for language understanding. (2019).