Attacking Recommender Systems with Augmented User Profiles

Chen Lin*
Xiamen University
chenlin@xmu.edu.cn

Si Chen
Xiamen University
sichen@stu.xmu.edu.cn

Hui Li†
Xiamen University
hui@xmu.edu.cn

Yanghua Xiao
Fudan University
shawyh@fudan.edu.cn

Lianyun Li
Xiamen University
lilianyun@stu.xmu.edu.cn

Qian Yang
Xiamen University
yangqian@stu.xmu.edu.cn

ABSTRACT

Recommendation Systems (RS) have become an essential part of many online services. Due to its pivotal role in guiding customers towards purchasing, there is a natural motivation for unscrupulous parties to spoof RS for profits. In this paper, we study the shilling attack: a subsistent and profitable attack where an adversarial party injects a number of user profiles to promote or demote a target item. Conventional shilling attack models are based on simple heuristics that can be easily detected, or directly adopt adversarial attack methods without a special design for RS. Moreover, the study on the attack impact on deep learning based RS is missing in the literature, making the effects of shilling attack against real RS doubtful. We present a novel Augmented Shilling Attack framework (AUSH) and implement it with the idea of Generative Adversarial Network. AUSH is capable of tailoring attacks against RS according to budget and complex attack goals, such as targeting a specific user group. We experimentally show that the attack impact of AUSH is noticeable on a wide range of RS including both classic and modern deep learning based RS, while it is virtually undetectable by the state-of-the-art attack detection model.

ACM Reference Format:
Chen Lin, Si Chen, Hui Li, Yanghua Xiao, Lianyun Li, and Qian Yang. 2020. Attacking Recommender Systems with Augmented User Profiles. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411884

1 INTRODUCTION

The history of **R**ecommender **S**ystems (RS) can be traced back to the beginning of e-commerce [1]. The ability of RS to assist users in finding the desirable targets makes it an important tool for alleviating information overload problem. As a result, RS has been prevalently deployed in industries (e.g., Amazon, Facebook and Netflix [1]). Not only is RS beneficial to customers, but also RS helps retail companies and producers promote their products and increase sales. Consequently, there is a strong intention for unscrupulous parties to attack RS in order to maximize their malicious objectives.

Due to RS's pivotal role in e-commerce, much effort has been devoted to studying how to spoof RS in order to give insights into the defense against malicious attacks. Various attacks, such as unorganized malicious attack (i.e., several attackers individually attack RS without an organizer) [2] and sybil attack (i.e., illegally infer a user's preference) [3], have been studied. This paper focuses on a subsistent and profitable attack, i.e., *shilling attack*, where an adversarial party produces a number of user profiles using some strategies to promote or demote an item [4] in order to have their own products recommended more often than those of their competitors. Shilling attack is also called data poisoning [5] or profile injection attack [6] in the literature. Researchers have successfully performed shilling attacks against real-world RS such as YouTube, Google Search, Amazon and Yelp in experiments [7, 8]. Large companies like Sony, Amazon and eBay have reported that they suffered from such attacks in practice [9].

Shilling attack is the specific application of *adversarial attack* [10, 11] in the domain of recommender systems. Adversarial attack uses crafted adversarial examples to mislead machine learning models. A tremendous amount of work in adversarial attack is against image classification [12], or text classification [13]. However, they cannot be directly employed in shilling attack at full power, due to the following challenges:

(1) **Data correlation in RS:** RS relies on capturing the correlations between users and items for recommendations and such relations enhance the robustness of RS. The recommendation targeting at a specific user is typically made based on the information from multiple user-item pairs (i.e., collaborative filtering [1]) instead of a single data sample. Therefore, manipulating the recommendation for one user requires to inject many related user-item pairs, which may affect the recommendation results for other non-targeting users in RS and make the attack easy to be detected. This is different compared to attacking many other learning tasks where manipulating one data sample may achieve the desirable attack goal and adversarial attacks can be directly deployed (e.g., one-pixel attack [12]).
(2) **No prior knowledge of RS:** A prevalent strategy of adversarial attack is to utilize the information of gradient decent in machine learning models to search undetectable perturbations and then combine perturbations with normal representation vectors to affect the learning system [10]. As a comparison,

*Chen Lin is supported by the National Natural Science Foundation of China (no.61972328).
†Hui Li is the corresponding author.

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.
CIKM '20, October 19–23, 2020, Virtual Event, Ireland
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411884

in shilling attack, though the data (e.g., rating matrix) of RS is generally available to all users (i.e., a user can see all other users' ratings) and thus exposed to attackers [4, 9, 14], the recommendation model is typically a *black box*. Thus, it is required that the attack must be effective against a wide range of recommendation models.

(3) **The balance of different complex attack goals:** Instead of only promoting or demoting an item to the general audience, there are usually multiple goals that the attacker desires to achieve. However, incorporating multiple attack goals together may degrade the attack performance of individual attack goal or make the attack detectable. Consequently, special designs are required to balance and achieve multiple attack goals simultaneously, while keeping the attack undetectable.

Due to the aforementioned challenges, only a few recent works [15, 16] consider directly adopting the idea of adversarial attacks for shilling attack, and they do not show satisfactory attack effects on a wide range of RS as illustrated later in our experiments. In addition to these methods, most existing shilling attack methods create injection profiles based on some global statistics, e.g., average rating value [4, 9] and rating variance [14] for each item. For instance, average attack assigns the highest rating to the target item to be promoted and an average rating to a set of randomly sampled items [9]. Although all these existing methods, including both adversarial based and simple heuristic based approaches, were proved to be effective in some cases, they still suffer from the following limitations:

(1) **Easy to detect**: Generated user profiles lack personalization (i.e., different user behavior pattern), thus the injected profiles can be easily detected, even by some simple heuristics (more details described in Sec. 5.4).
(2) **Narrow range of target models**: Depending on how the statistics are computed, conventional shilling attacks are shown to be effective only on certain traditional collaborative filtering (CF) approaches. For example, average, bandwagon and random attacks are more effective against user-based KNN, but do not work well against item-based KNN [17]. Moreover, their influence on deep learning based RS, which has attracted considerable interest and been deployed in real applications [18], has not been studied. In fact, as global statistics can not capture high-level associations among items or users, the actual effect of the existing attack approaches on modern RS is doubtable (more details described in Sec. 5.2).
(3) **Inflexibility**: It is difficult to tailor the attack for specific goals which attackers desire to achieve after the attack, e.g., to exert adverse effects on items from the competitors.

To address the above problems, a natural intuition to enhance the attack is to "augment" the templates, which are selected from existing real user profiles and are used to generate injected profiles. This way, the injected fake user profiles are diversified and it becomes difficult to distinguish them from real users. Based on this intuition, we present a novel **Au**gmented **Sh**illing Attack (AUSH) and implement it with the idea of Generative Adversarial Network (GAN) [19]. Specifically, the generator acts like an "attacker" and generates fake user profiles by augmenting the "template" of existing real user profiles. The deep neural network based generator can capture complex user-item associations better than existing attack methods using simple heuristics. Thus, it works well on modern RS which commonly deploys deep neural networks. Moreover, the generator is able to achieve secondary attack goals by incorporating a shilling loss. On the other hand, the discriminator module performs like a "defender". It distinguishes fake user profiles from real user profiles and provides guidance to train the generator to generate undetectable fake user profiles. Each of the generator and the discriminator strikes to enhance itself to beat the other one at every round of the minimax competition. It is worthy noting that, as we have explained, deploying the idea of adversarial attack in shilling attack is not a trivial task, and *directly applying the adversarial attack method (i.e., using a general GAN) in shilling attacks without our designs to tailor it for the attack will not provide satisfactory results as shown in our experiments.*

Our contributions can be summarized by three merits of AUSH. We show that AUSH resembles to the traditional *segment attack* and *bandwagon attack* [4], yet more *powerful*, *undetectable* and *flexible* than conventional shilling attack methods:

(1) AUSH is powerful on a wide range of recommendation models including both traditional CF methods and modern deep learning based approaches, while *the prior knowledge of AUSH does not exceed what the conventional shilling attack approaches require to know.*
(2) Furthermore, AUSH is virtually undetectable by the state-of-the-art attack detection method as shown in our experiments.
(3) Finally, AUSH contains more than a general GAN as it includes a reconstruction loss and a shilling loss which tailor AUSH for attacking RS and endows the AUSH with the ability of achieving secondary attack goals (e.g., promote items for a group of users who have shown preferences over a predefined set of competitors, or target on long-tail items).

We conduct comprehensive experiments to verify the above merits of AUSH and its attack power against both classic and modern deep learning based recommendation algorithms. Note that attacking modern deep neural network recommendation algorithms has rarely been studied in the literature.

The rest of the paper is organized as follows: Sec. 2 illustrate the related work. Sec. 3 demonstrates the design of AUSH and Sec 4 gives one possible implementation of AUSH. In Sec. 5, we compare AUSH with other state-of-the-art shilling attacks methods and verify its effectiveness. Sec. 6 concludes our work.

2 RELATED WORK

2.1 Recommender Systems (RS)

Traditional RS typically relies on collaborative filtering methods (CF), especially matrix factorization (MF) methods [20]. MF models user preferences and item properties by factorizing the user-item interaction matrix into two low-dimensional latent matrices. Recently, numerous deep learning techniques (e.g., MLP [21], CNNs [22], RNNs [23], GNNs [24], Autoencoder [25], and the Attention Mechanism [26]) have been introduced into RS. Compared to traditional RS, deep learning based RS is able to model the nonlinearity of data correlations and learn the underlying complex feature representations [18]. Consequently, deep learning based RS has outperformed traditional RS in general.

2.2 Adversarial Attacks

Machine learning has played a vital role in a broad spectrum of applications and helped solve many difficult problems for us. However, security of machine learning systems are vulnerable to crafted adversarial examples [10], which may be imperceptible to the human eye, but can lead the model to misclassify the output. Adversaries may leverage such vulnerabilities to compromise a learning system where they have high incentives and such attacks are called as *adversarial attacks*. Adversarial attack has show its ability to manipulate the outputs of many text and image based learning systems [10, 11, 27, 28].

Adversarial examples in conventional machine learning models have been discussed since decades ago [11]. Dalvi et al. [29] find manipulating input data may affect the prediction results of classification algorithms. Biggio et al. [30] design a gradient-based approach to generate adversarial examples against SVM. Barreno et al. [31, 32] formally investigate the security of conventional machine learning methods under adversarial attacks. Roli et al. [33] discuss several defense strategies against adversarial attacks to improve the security of machine learning algorithms. In addition to conventional machine learning, recent studies have reported that deep learning techniques are also vulnerable to adversarial attacks [10, 11].

Though we have witness a great success of adversarial attacks against many learning systems, existing adversarial attacks cannot be directly adopted for the shilling attack task as explained in Sec. 1.

2.3 Generative Adversarial Network

Generative Adversarial Network (GAN) [19] has recently attracted great attention for its potential to learn real data distribution and generate text [34], images [35], recommendations [36] and many other types of data [37]. GAN performs adversarial learning between the generator and the discriminator. The generator and the discriminator can be implemented with any form of differentiable system that maps data from one space to the other. The generator tries to capture the real data distribution and generates real-like data, while the discriminator is responsible for discriminating the data generated by the generator and the real data. GAN plays a minimax game and the optimization terminates at a saddle point that is a minimum with respect to the generator and a maximum with respect to the discriminator (i.e., Nash equilibrium).

As GAN overcomes the limitations of previous generative models [37], it has been successfully applied in many applications and there is a surge of works studying how to improve GAN [37]. Follow-up works include DCGAN [38] which adopts the CNN architecture in GAN and Wasserstein GAN [39] which leverages Earth Mover distance. There also exists a direction of GAN research which utilizes GAN to generate adversarial examples. For instance, [40] propose to search the representation space of input data instead of input data itself under the setting of GAN in order to generate more natural adversarial examples. [41] design AdvGAN which can attack black-box models by training a distilled model.

2.4 Shilling Attacks against RS

O'Mahony et al. [42, 43] firstly study the robustness of user-based CF method for rating prediction by injecting some faked users. They also provide a theoretical analysis of the attack by viewing injected ratings as noises. Burke et al. [6], Lam and Riedl [9], Mobasher et al. [17], Burke et al. [44] further study the influence of some low-knowledge attack approaches to promote an item (e.g., random, average, bandwagon attack and segment attack) and to demote an item (e.g., love/hate attack and reverse bandwagon attack) on CF methods for both rating prediction and top-K recommendation. They observe that CF methods are vulnerable to such attacks. Assuming more knowledge and cost, Wilson and Seminario [45], Seminario and Wilson [46] design the power user/item attack models which leverage most influential users/items to shill RS, Fang et al. [47] study how to shill a graph based CF models, and Li et al. [5] present near-optimal data poisoning attacks for factorization-based CF. Xing et al. [7], Yang et al. [8] conduct experiments on attacking real-world RS (e.g., YouTube, Google Search, Amazon and Yelp) and show that manipulating RS is possible in practice.

Inspired by the success of GAN, a few works turn to leverage GAN for shilling attack task [15, 16]. However, directly adopting existing GAN methods for generating adversarial examples, without special designs (like AUSH) to tailor them for RS, will not provide satisfactory results in shilling attacks as shown in our experiments. Christakopoulou and Banerjee [15, 16] employ DCGAN [38] to generate faked user profiles used in shilling attacks. They formulate this procedure as a repeated general-sum game between RS and adversarial fake user generator. Compared to their work, AUSH is more specially tailored for RS instead of directly using adversarial attacks (i.e., the general GAN) against machine learning models. We consider more realistic factors (e.g., users in the segment, attack cost and undetectability) when attacking RS, which descend from previous study on attacking traditional CF models.

Note that the study on the impact of shilling attacks against deep learning based RS is limited, although there is a tremendous amount of work on attack and defense of traditional RS. Therefore, we also include an analysis of attacking deep learning based RS in the Sec. 5 of this paper.

3 AUGMENTED SHILLING ATTACK

In this section, we introduce our proposed attack framework: Augmented Shilling Attack (AUSH).

3.1 Terminology

We follow the terminology used in the literature [4] and divide the items in a fake user profile into one *target item* (i.e., the attacker wants to assign it a malicious rating), a number of *filler items* (i.e., a group of randomly sampled items which have been rated by the real user and will be used to obstruct detection of the attack), a number of *selected items* (i.e., a group of human-selected items for special treatment to form the characteristics of the attack), and *unrated items* (i.e., the rest of the items in the RS). Selected items are the same across all fake user profiles, while each fake user profile has its own filler items.

3.2 Attack Budget and Goal

Attacking RS is costly. As such, in designing a practical attack model against RS, we have to take into account the following attack budget and goal:

- **Attack budget**: we consider two factors

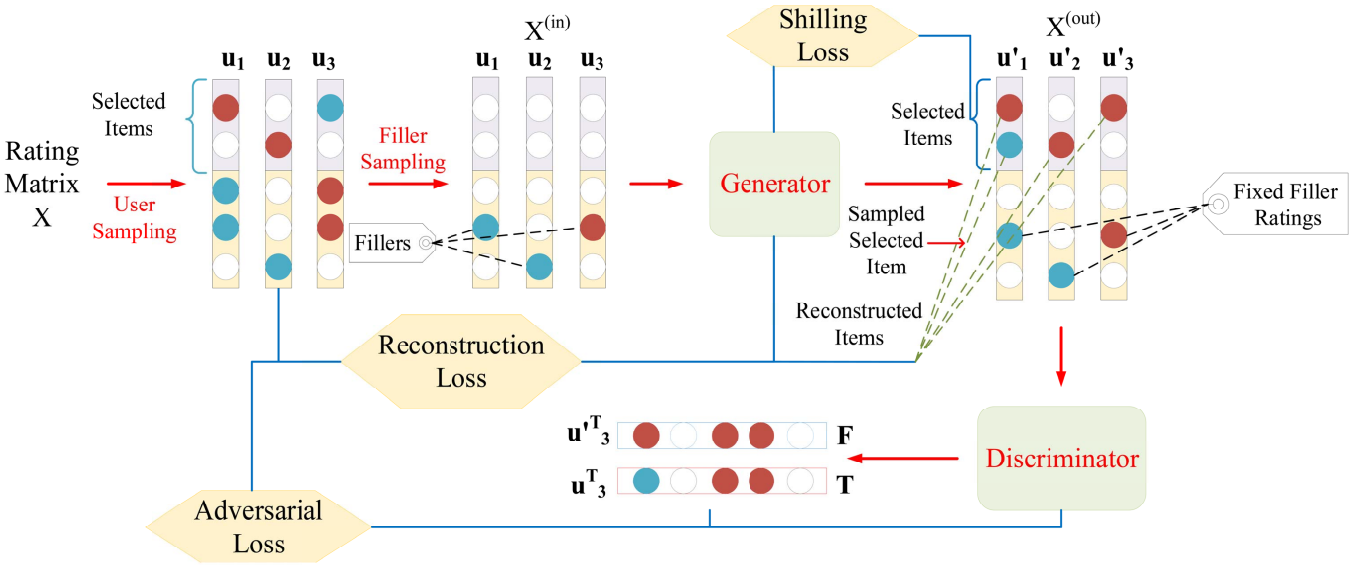

Figure 1: Pipeline of AUSH. We use binary ratings for illustration, though AUSH can handle a five-point scale. Red and blue indicate a high rating and a low rating, respectively.

- **Attack size** is the number of fake user profiles
- **Profile size** is the number of non-zero ratings. The larger the attack size / profile size is, the more effective and expensive the attack could be.

- **Attack goal**: the goal an adversarial party wants to achieve could be complex and we mainly consider the following aspects
  - **Attack type** indicates whether it is a push attack (i.e., assign a maximal rating on target item to promote it) or a nuke attack (i.e., assign a minimal rating on target item to demote it). Since the two types are similar and can be exchanged (i.e., change a maximal rating to a minimal rating), we consider push attacks in the sequel for simplicity.
  - **Target user group** is the group of users that an attack aims at.
  - **Ancillary effects** (e.g., demoting competitors, bias the ratings of a special user groups on selected items) are also desired in the attack. Such intentions will manifest in choosing selected items.

3.3 Overview of AUSH

Conventional attack models make up a fake user profile from scratch. On the contrary, our intuition is to use an existing real user profile as a "template" and **au**gment it to generate the fake user profile for **sh**illing (AUSH). The template knowledge is accessible in practice and do not exceed the requirements of recent sophisticated attack methods [5, 47], as we will show later. The benefits are two-fold. Firstly, the generated profile is indistinguishable as it is built upon real user behavior patterns. Moreover, it retains the preference diversity of the community. Unlike random or average attack, where fake users do not show specific tastes, our strategy can generate fake users who have a special taste on niche items.

Inspired by the success of adversarial learning in image generation [19], we employ a Generative Adversarial Network framework for AUSH to make the attack even more undetectable. Fig. 1 gives an overview of our pipeline, which consists of the following parts:

(1) **Sampling ("template" selection)** contains two steps. In the first step, a batch of real users are chosen as "templates". In the second step, filler items of each "template" are sampled from the rated items of the corresponding "template" user.
(2) **Generator ("patch" generation)** patches each "template" by adding ratings on selected items to generate one fake profile. Generator takes as input the sampled user-item rating submatrix (i.e., "templates") and captures the latent association between items and users. To better learn behavior patterns of the real user (i.e., the "template" user) including positive and negative preference on selected items, AUSH attempts to recover each "template" user's observed ratings on selected items and samples of unobserved selected items (i.e., to recover the rating "0") via a *reconstruction loss*. The output of generator is a set of fake user profiles, which contain ratings on selected items. We can harness a *shilling loss* to optimize secondary attack effects, including but not limited to demoting the competitors, targeting on special user groups, etc.
(3) **Discriminator** is fed with the output of the generator. It attempts to accurately classify real user profiles and fake user profiles. The *adversarial loss* is optimized to boost the performance of discriminator.

The design of AUSH is general and there are various possible implementations for the generator and the discriminator. We provide one implementation in Sec. 4.

3.4 Relation to Segment/Bandwagon Attack

Segment attack injects user profiles, each of which comprises maximal ratings on selected items and minimal ratings on filler items. For *in-segment users* (defined as users who like selected items), segment attack is one of the few attack models that work effectively on item-based CF recommendation models. The design of segment

Algorithm 1: Training procedure for AUSH

Input: rating matrix $\mathbf{X}$
Output: parameter set θ for generator G and parameter set ϕ for discriminator D
for *number of training epochs* **do**
  for k_1 *steps* **do**
    uniformly sample a minibatch of users $\mathcal{U}'$;
    foreach $u' \in \mathcal{U}'$ **do**
      sample F items to construct $\mathbf{x}_{u'}^{(in)}$;
    generate a minibatch of fake user profiles $\{\mathbf{x}_{u'}^{(out)} = G(\mathbf{x}_{u'}^{(in)}) | u' \in \mathcal{U}'\}$;
    optimize ϕ to max $H(G, D)$ with θ fixed;
  for k_2 *steps* **do**
    uniformly sample a minibatch of user rating vectors $\{\mathbf{x}_u\}$;
    foreach $u' \in \mathcal{U}'$ **do**
      sample F items to construct $\mathbf{x}_{u'}^{(in)}$;
    generate a minibatch of fake user profiles $\{\mathbf{x}_{u'}^{(out)} = G(\mathbf{x}_{u'}^{(in)}) | u' \in \mathcal{U}'\}$;
    optimize θ to min $\mathcal{L}_{AUSH}$ with ϕ fixed;

attack ensures that similarity between users in the segment and injected profiles appears high and target item becomes more likely to be recommended.

Another commonly adopted attack model is bandwagon attack. In bandwagon attack, the most popular items are regarded as selected items and are assigned with highest ratings. The filler items are randomly chosen and randomly rated. It associates the target item with popular items, so that the inserted profiles will have a high probability of being similar to many users [44].

We see that segment attack and bandwagon attack can be expressed under our framework. If we fix ratings on the fillers and selected items to be minimal rating and maximal rating respectively, then AUSH is degraded to segment attack. If we sample frequently rated items as selected items, then AUSH is degraded to bandwagon attack. Due to the architectural resemblance, AUSH is empowered by the capabilities of both segment attack and bandwagon attack. Moreover, AUSH improves over bandwagon attack by allowing the selected item to be tuned according to the rating values of fillers, making the injected profile more natural and indistinguishable. It also advances segment attack by revealing real patterns of filler ratings. In addition to the aforementioned advantages, AUSH is more flexible and able to achieve multiple goals (i.e., *Attack Goal* in Sec. 3.2) in a single attack.

4 IMPLEMENTATION

We use $\mathbf{X} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{U}|}$ to denote the rating matrix in RS, where $\mathcal{U}$ is the set of real users and $\mathcal{V}$ is the item universe. $\mathcal{V}_u = \{v \in \mathcal{V} : x_{v,u} \neq 0\}$ indicates the set of items that have been rated by u. Similarly, $\mathcal{U}_v = \{u \in \mathcal{U} : x_{v,u} \neq 0\}$ denotes the set of users that have rated v. Unless otherwise stated, we use lower-case letters for indices, capital letters for scalars, boldface lower-case letters for vectors, boldface capital letters for matrices, calligraphic letters for sets. For instance, A, P, F, $\mathcal{U}'$ and $\mathcal{S}$ are attack size, profile size, filler size, set of fake users and set of selected items, respectively.

The generator of AUSH takes $\mathbf{X}^{(in)} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{U}'|}$ as the input and generates the fake user profiles $\mathbf{X}^{(out)} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{U}'|}$, where each column has exactly P non-zero entries. As depicted in Alg. 1, AUSH comprises of the following components and steps.

4.1 Sampling

In this step, AUSH samples a sub-matrix $\mathbf{X}^{(in)} \in \mathcal{R}^{|\mathcal{V}|\times|\mathcal{U}'|}$ (i.e., "templates") from $\mathbf{X}$. Each "template" is sampled randomly from real users who have sufficient ratings. Mathematically, $\forall u' \in \mathcal{U}', |\mathcal{V}_{u'}| \geq P$. In each training epoch of Alg. 1, the set $\mathcal{U}'$ is a minibatch of users. In test time (i.e., the generated fake profiles are used for attack), we sample exactly A fake user profiles $|\mathcal{U}'| = A$. We adopt different strategies as shown below to sample the filler items for each $u' \in \mathcal{U}'$ and form $\mathbf{x}_{u'}^{(in)}$. For each filler item v, $\mathbf{X}_{v,u'}^{(in)} = \mathbf{X}_{v,u'}$. For other items, $\mathbf{X}_{v,u'}^{(in)} = 0$.

(1) **Random Sample**: randomly sample items from $\mathcal{V}_u$.
(2) **Sample by Rating**: sample items based on their ratings, i.e., $P(\mathbf{X}_{v,u'}^{(in)} \neq 0) = \frac{\bar{r}_v}{\sum_{\hat{v}\in\mathcal{V}_u} \bar{r}_{\hat{v}}}$, where $\bar{r}_v$ is v's average rating.
(3) **Sample by Popularity**: items are sampled based on their popularity, i.e., $P(\mathbf{X}_{v,u'}^{(in)} \neq 0) = \frac{|\mathcal{U}_v|}{\sum_{\hat{v}\in\mathcal{V}_u} |\mathcal{U}_{\hat{v}}|}$.
(4) **Sample by Similarity**: sample items based on their similarity to the *selected items*, i.e., $P(\mathbf{X}_{v,u'}^{(in)} \neq 0) = \frac{|\mathcal{U}_v \bigcap \mathcal{U}_S|}{\sum_{\hat{v}\in\mathcal{V}} |\mathcal{U}_{\hat{v}} \bigcap \mathcal{U}_S|}$.

4.2 Generator

The generator aims to "patch" the "templates" with ratings on selected items in order to form the fake user profiles for attack.

We employ a *reconstruction loss* (i.e., MSE loss), shown in Eq. 1, to optimize the generator parameters. We will slightly abuse the notation, and define $\mathcal{S}_{u'}^{+} = \mathcal{V}_{u'} \bigcap \mathcal{S}$ as the set of observed ratings of the "template" user for user u' on selected items, and $\mathcal{S}_{u'}^{-} = (\mathcal{V} - \mathcal{V}_{u'}) \bigcap \mathcal{S}$ as random samples from the set of selected items that the "template" user has not rated in the original data. And $T_{u'}$ indicates $\mathcal{S}_{u'}^{+} \bigcup \mathcal{S}_{u'}^{-}$.

$$\mathcal{L}_{Recon} = \mathbb{E}_{u'\sim\mathcal{U}'} \sum_{j\in T_{u'}} \left(\mathbf{X}_{j,u'}^{(out)} - \mathbf{X}_{j,u'}\right)^2, \ \mathbf{X}^{(out)} = G(\mathbf{X}^{(in)}), \quad (1)$$

where $G(\cdot)$ indicates the generator which will be defined in Eq. 2.

The reconstruction loss helps to produce ratings on the selected items that are consistent with the real user's preference. Note that we use minibatch for training as shown in Alg. 1. Thus we sample m (the percentage) of unobserved selected items for all the users in a minibatch when constructing reconstructed items for these users, instead of independently sampling unobserved selected items for each user.

There is a variety of model structures for optimizing the reconstruction loss, we empirically find that towered multilayer perceptron (MLP) combined with the MSE loss on selected items works best. Let N be the number of hidden layers, the generator G is a mapping function that operates in a towered manner:

$$G(\mathbf{x}) = f_{out}\Big(f_N\Big(\cdots f_2\big(f_1(\mathbf{x})\big)\cdots\Big)\Big). \quad (2)$$

In Eq 2, $f_l(\cdot)$ with $l = 1, 2, \cdots, N$ denotes the mapping function for the l-th hidden layer. $f_l(\mathbf{x}) = \sigma(\mathbf{W}_l\mathbf{x} + \mathbf{b}_l)$, where $\mathbf{W}_l$ and $\mathbf{b}_l$ are

learnable weight matrix and bias vector for layer l. The activation function σ for each layer is sigmoid. We set the size of layers (i.e., dimensionality of $\mathbf{x}$) as one third of the previous layers. The output layer $f_{out}(\cdot)$ is similar to $f_l(\cdot)$ and its size is the number of selected items.

AUSH can be extended to achieve secondary attack goals by incorporating a *shilling loss*. In this work, we consider enhancing the attack effect on in-segment users [9]. That is, we increase the impact of the attack on users who like the selected items before the attack. Such an effect is desirable when an adversarial party (i.e., the attacker) is competing with the selected items (from its competitor). The shilling loss we adopt is shown as follows:

$$\mathcal{L}_{Shill} = \mathbb{E}_{u' \sim \mathcal{U}'} \sum_{j \in \mathcal{S}} \left(Q - \mathbf{X}^{(out)}_{j,u'}\right)^2, \tag{3}$$

where Q is the maximal possible rating in the system. The shilling loss produces fake user profiles that are more likely to associate with in-segment users. Thus in-segment users, after our attack, prefer to purchase the target item rather than the selected items (from the competitor). Through optimizing shilling loss, AUSH is able to achieve the ancillary effects.

4.3 Discriminator

The discriminator D attempts to correctly distinguish fake user profiles from real user profiles, and encourages the generator to produce realistic user profiles. We use MLP $D(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ as our discriminator, where $D(\cdot)$ estimates probabilities of its inputs been real user profiles, $\mathbf{W}$ and $\mathbf{b}$ are weight matrix and bias vector.

Inspired by the idea of GAN [19], we aim to unify the different goals of generator and discriminator by letting them play a *minimax game* via optimizing the following *adversarial loss*:

$$\min_{\theta} \max_{\phi} H(G, D) = \mathbb{E}_{u \sim \mathcal{U}}[\log D(\mathbf{x}_u)] + \mathbb{E}_{z \sim p_\theta}[\log\left(1 - D(z)\right)], \tag{4}$$

where θ and ϕ are model parameters of G and D, respectively. $\mathbf{x}_u$ is a real user profile. z is a fake user profile from the generator distribution p_θ.

4.4 Learning

Finally, the complete objective considers adversarial loss, reconstruction loss and shilling loss, and leads to the following formulation:

$$\mathcal{L}_{AUSH} = \min_{\theta} \max_{\phi} \Big(H(G, D) + \mathcal{L}_{Shill} + \mathcal{L}_{Recon}\Big). \tag{5}$$

As shown in Alg. 1, in each round of the optimization, each of the "attacker" (i.e., generator) and "defender" (i.e., discriminator) endeavors to improve itself to defeat the other part. The generator attempts to generate "perfect" fake user profiles that are difficult to detect, while the discriminator tries to accurately identify fake profiles. During this procedure, the generator learns to produce fake profiles similar to real profiles via optimizing the reconstruction loss. At the same time, optimizing the shilling loss endows the fake profiles with the capability to exert ancillary influence (e.g., demote competitors or bias in-segment users).

Table 1: Statistics of data

Data	#Users	#Items	#Ratings	Sparsity
ML-100K	943	1,682	100,000	93.70%
FilmTrust	780	721	28,799	94.88%
Automotive	2,928	1,835	20,473	99.62%

Data	Attack Size	Filler Size	#Selected Items	Profile Size
ML-100K	50	90	3	94
FilmTrust	50	35	2	38
Automotive	50	4	1	6

5 EXPERIMENT

In this section, we conduct experiments in order to answer the following research questions:

- **RQ1**: Does AUSH have better attack performance on both traditional and deep learning based RS, than other shilling attack methods?
- **RQ2**: If adversarial attack methods are directly used (i.e., using a general GAN) for shilling attack, what are the attack impacts?
- **RQ3**: Is AUSH able to achieve secondary attack goals at the same time?
- **RQ4**: How much does each component in AUSH contribute to the attack effects?
- **RQ5**: Is it more difficult for attack detector to recognize the attack launched by AUSH, compared to shilling attack methods?

In the following, we first demonstrate our experiment setup in Sec. 5.1. Then, the attack effect of AUSH is verified on three well-known recommendation benchmarks and is compared with both heuristic based and general GAN based attack models in Sec. 5.2 (RQ1, RQ2, RQ3). After that, we investigate the role of each component in AUSH on the attack impact (RQ4). Finally, we show that AUSH can not be detected by supervised and unsupervised attack detection methods in Sec. 5.4 and it generates indistinguishable profiles in terms of similarity measurements (RQ5).

5.1 Experimental Setup

We use three benchmark data sets for RS in our experiments: ML-100K[1], FilmTrust[2] and Amazon Automotive[3]. Most of the previous work [14] only uses ML-100K as the single data set. We use its default training/test split. In addition, we use FilmTrust and Automotive, which are larger and sparser, to testify the competence of AUSH in different settings. We randomly split them by 9:1 for training and testing, respectively. To exclude cold-start users (as they are too vulnerable), we filter users with less than 15 ratings and items without ratings.

We inject 50 user profiles (i.e., roughly 5% of the population which can manifest the differences among attack models [6]) in each attack. The number of fillers in each injected user profile equals to the average number of ratings per user in the data set. For each target item in ML-100K, we select a small number of items that are most frequently rated under the same tag/category of the target item as the selected items. For each target item in FilmTrust and Automotive which do not have information of tag/category, we sample items from global popular items as the selected items. Tab. 1 illustrates the statistics of the data.

[1] https://grouplens.org/datasets/movielens/100k/
[2] https://www.librec.net/datasets/filmtrust.zip
[3] http://jmcauley.ucsd.edu/data/amazon/

We use TensorFlow for the implementation. The generator of AUSH has 5 hidden layers (i.e., $N = 5$) with 400, 133, 44, 14 and 4 neurons for each layer. We use random sampling as the default strategy for sampling filler items in AUSH as it requires the least effort. The output layer size is the number of selected items. We use Adam [48] for optimization with an initial learning rate of 0.01. The maximal number of adversarial iterations is set to be 150.

5.2 Attack Performance (RQ1, RQ2, RQ3)

To answer RQ1, we investigate the attack performance of AUSH and compare it with other baselines on several classic and deep learning based RS models including NMF [49], NNMF [50] and AutoEncoder [25] in our experiments. Note that AUSH is designed for attacking rating based RS and we need to estimate the exact values of ratings in the experiment. Thus we exclude methods such as NCF [21] which is designed for implicit feedback. We compare AUSH with several shilling attack models:

(1) **Random attack** assigns a rating $r \sim \mathcal{N}(\mu, \sigma)$ to a filler, where μ and σ are the mean and the variance of all ratings in the system, respectively.
(2) **Average attack** assigns a rating $r \sim \mathcal{N}(\mu, \sigma)$ to a filler, where μ and σ are the mean and the variance of ratings on this filler in the system, respectively.
(3) **Segment attack** assigns maximal ratings to the selected items and minimal ratings to the filler items.
(4) **Bandwagon attack** uses the most popular items as the selected items and assigns maximal ratings to them, while fillers are assigned ratings in the same manner as random attack.
(5) **DCGAN** is an adversarial network [38] adopted in a recent shilling attack method [15, 16], where the generator takes the input noise and output fake user profiles through convolutional units. We use the default settings in [16].
(6) **WGAN** is similar to DCGAN, but we replace the GAN used in the shilling attack with Wasserstein GAN [39] which has a good empirical performance [51].

In all methods, the highest rating is assigned to the target item. We train each RS and AUSH until convergence. The required information (e.g., mean and variance) is obtained from the training set. Note the prior knowledge of AUSH does not exceed what the baselines require to know. Then we inject user profiles generated by the attack models to the training set and train the RS again on the polluted data. We evaluate the attack performance on the test set using prediction shift (PS) and Hit Ratios at K (HR@K). PS is the difference of ratings that the RS makes before and after the attack. HR@K is the hit ratio of target items in the top-K recommendations after the attack. As we are performing a push attack, the PS and HR@K need to be positive to indicate an effective attack. The larger their values are, the more effective the attack is. In the evaluation, we use $K = 10$ for HR@K.

Overall Performance (RQ1). We randomly select five items as *random targets*. As indicated in the literature [9], unpopular items (i.e., long-tail items) are likely to be the targets of an attack. Therefore, we additionally sample five target items with the number of ratings no more than a threshold as *random long-tail targets*. The threshold number of ratings is one in ML-100K, two in FilmTrust, and three in Automotive. We report the average attack performance on the three data sets for random targets and random long-tail targets, when the complete loss $\mathcal{L}_{AUSH}$ (i.e., Eq. 5) is used in AUSH, in Tabs. 2, 3, 4 and 5 for attacking NMF, NNMF, U-AutoEncoder and I-AutoEncoder, respectively. We highlight the best performance in each category. AUSH will also be highlighted if it achieves the second best performance.

From experimental results, we can see that *AUSH generally achieves attractive attack performance against all recommendation models* on target items. It generates the largest PS and HR@10 in most categories including both random targets and random long-tail targets, showing that AUSH is a practical method for attackers who want to promote their products. *Conventional attack models do not show a robust attack performance like AUSH, even though they may exceed AUSH in a few cases.*

Comparisons between AUSH and General GANs (RQ2). We can observe from Tabs. 2, 3, 4 and 5 that *directly adopting the idea of adversarial attacks (i.e., using general GANs) does not give a satisfactory performance.* Particularly, both DCGAN which is adopted in the recent shilling attack [15, 16] and WGAN [39] which aims at stabilizing GAN training do not show better performance than simple heuristic based attack approaches like Average attack and Random attack. In some cases, attacks launched by DCGAN and WGAN even give opposite effects (i.e., negative PS and HR@K). It validates our assumption that a tailored GAN framework is necessary for shilling attack.

Secondary Attack Goals (RQ3). As explained in Sec. 4.2, incorporating a shilling loss helps AUSH to achieve secondary attack goal, i.e., increasing the impact of the attack on users who like the selected items before the attack. We call such users *in-segment users* [9] and they are target population to certain attackers. We define in-segment users as users who have assigned high ratings (i.e., 4- or 5-stars) on all selected item in our experiments. In Tabs. 2, 3, 4 and 5, we already report the attack results for in-segment users and all users, and list random targets and random long-tail targets separately. In Tab. 6, we further show the attack performance on I-AutoEncoder in different settings and the results are reported by averaging attack impacts on random targets and random long-tail targets. Note that AUSH$_{rand}$ in Tab. 6 indicates the complete loss (i.e., Eq. 5) and the default random sampling are used, i.e., it is equivalent to AUSH in Tabs. 2, 3, 4 and 5. For example, AUSH$_{rand}$ has a PS of 1.6623 for in-segment users in Tab. 6 which is the average of AUSH's attack performances for in-segment users on random targets and random long-tail targets of ML-100K (i.e., 1.3746 and 1.9499) in Tab. 5. Due to space limit, we only report attack results in ML-100K in Tab. 6, but we observe similar results in other settings.

We can observe that AUSH (in Tabs. 2, 3, 4 and 5) and AUSH$_{rand}$ (in Tab. 6) enhance the power of segment attack – a much more significant attack impact on in-segment users than on all users, while other baselines are not that flexible and they are unable to achieve such a secondary attack goal. This property of AUSH is desirable if the attacker wants to demote the items from competitors. Note AUSH$_{rand}$ uses the complete loss. For a further study on impacts of the different loss components, please refer to the next section.

Table 2: Attack performance against NMF. Best results are marked in bold, and AUSH results are also marked in bold if they are the second best in each category.

Metric	In-segment Users						All Users					
	Prediction Shift			HR@10			Prediction Shift			HR@10		
Data Set	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive
Model	Random Targets											
AUSH	**1.8857**	**0.8937**	**0.2778**	**0.2538**	**0.2822**	**0.0539**	**1.7503**	**0.9650**	**0.2585**	**0.1849**	**0.2821**	**0.0541**
Segment	1.0157	0.6832	0.2313	0.0372	0.3214	**0.1545**	0.7061	0.4504	**0.2649**	0.0380	0.1978	**0.1132**
Average	1.8478	0.8721	0.1972	0.2147	0.2208	0.0239	1.7754	0.9522	0.2100	0.1787	0.2241	0.0241
Random	1.7220	0.8667	0.2332	0.1253	0.2708	0.0380	1.6285	0.9570	0.2391	0.0995	0.3140	0.0406
Bandwagon	1.7199	0.8184	0.2380	0.1791	0.2380	0.0294	1.6194	0.8508	0.2327	0.1257	0.2048	0.0300
DCGAN	-0.0112	0.1082	0.1002	0.0000	0.0833	0.0086	-0.0096	0.1005	0.1065	0.0000	0.0751	0.0046
WGAN	0.0774	0.1966	0.0473	0.0000	0.0469	0.0040	0.0723	0.1923	0.0396	0.0000	0.0374	0.0055
	Random Long-tail Targets											
AUSH	**2.9387**	**1.4263**	**0.2575**	**0.6007**	**0.1571**	0.0055	**2.8949**	**1.4758**	0.2456	**0.5057**	**0.1961**	**0.0091**
Segment	2.7918	0.9993	0.1719	0.5175	0.2197	**0.0669**	2.5726	0.7095	**0.2961**	0.3450	0.1265	**0.0541**
Average	2.9427	1.4084	0.2508	0.5044	0.0941	0.0066	2.9038	1.4723	0.2544	0.4420	0.1247	0.0041
Random	2.8994	1.4084	**0.2618**	0.6661	0.1568	0.0050	2.8401	1.4718	0.2724	0.5276	0.2159	0.0091
Bandwagon	2.8752	1.3426	0.1385	0.6232	0.1412	0.0000	2.8100	1.3561	0.1628	0.4900	0.1501	0.0011
DCGAN	-0.1479	0.1753	-0.0731	0.0000	0.0008	0.0000	-0.1374	0.1836	-0.0383	0.0000	0.0088	0.0002
WGAN	1.2299	0.4455	-0.0509	0.0000	0.0332	0.0000	1.2473	0.4071	-0.0416	0.0000	0.0298	0.0016

Table 3: Attack performance against NNMF. Best results are marked in bold, and AUSH results are also marked in bold if they are the second best in each category.

Metric	In-segment Users						All Users					
	Prediction Shift			HR@10			Prediction Shift			HR@10		
Data Set	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive
Model	Random Targets											
AUSH	**1.2225**	**0.9092**	0.2507	**0.1170**	**0.3027**	0.0242	**1.4009**	**1.1156**	0.3017	**0.1704**	**0.3614**	0.0254
Segment	0.0500	0.4423	0.1745	0.0156	0.1330	0.0213	-0.4469	0.4486	0.1701	0.0069	0.1240	0.0242
Average	0.8749	0.7795	**0.3016**	0.0665	0.2220	0.0279	1.1468	0.9129	**0.3491**	0.1112	0.2340	0.0392
Random	0.5837	0.7634	0.2815	0.0431	0.1568	**0.0399**	0.8732	0.9334	0.3005	0.0411	0.2083	**0.0426**
Bandwagon	0.6517	0.7333	0.2716	0.0388	0.1945	0.0223	0.5153	0.8634	0.3157	0.0309	0.2168	0.0260
DCGAN	-0.0611	-0.2444	0.0468	0.0012	0.0010	0.0000	0.0885	-0.1889	0.0274	0.0013	0.0034	0.0010
WGAN	-0.0543	0.0786	0.0093	0.0000	0.0600	0.0100	-0.0649	0.1085	-0.0041	0.0007	0.0457	0.0037
	Random Long-tail Targets											
AUSH	**1.5956**	**0.9002**	**0.8406**	**0.2654**	**0.2957**	**0.0257**	**1.7413**	**1.1241**	**0.8343**	**0.3420**	**0.3799**	**0.0206**
Segment	-0.4232	0.3003	0.5454	0.0011	0.1360	0.0116	-0.8599	0.3996	0.5150	0.0011	0.1242	0.0162
Average	1.4323	0.7883	0.8203	0.1503	0.1532	0.0188	1.5251	0.9430	0.7721	0.2236	0.1841	0.0158
Random	1.3755	0.8430	0.8023	0.1432	0.2011	**0.0307**	1.4984	1.0222	0.7878	0.2255	0.2648	**0.0402**
Bandwagon	1.3315	0.6977	0.5278	0.1296	0.1143	0.0102	1.4923	0.8026	0.5131	0.1877	0.1306	0.0056
DCGAN	0.1487	-0.4251	0.1673	0.0000	0.0010	0.0000	0.2164	-0.3518	0.1438	0.0000	0.0008	0.0028
WGAN	0.0555	0.1383	0.2385	0.0000	0.0021	0.0000	0.0266	0.2591	0.1144	0.0000	0.0033	0.0081

5.3 Impacts of Sampling Strategies and Each Loss (RQ4)

To answer RQ4, we remove or change some components of AUSH and investigate the performance changes.

Impacts of Sampling Strategies. We report the impacts of different sampling strategies in Tab. 6. AUSH$_{rand}$, AUSH$_{rating}$, AUSH$_{pop}$ and AUSH$_{sim}$ indicate random sample, sample by rating, sample by popularity and sample by similarity, respectively. All the four variations of AUSH adopt the complete loss (i.e., Eq. 5). We can observe that sample by rating is the best strategy for AUSH. The reason may be that it is easy to bias people with items having high ratings, as customers tend to trust such "power" items [46]. Nevertheless, all the variations have more significant attack impacts than other baselines.

Impacts of Each Loss. To study the contributions of each loss term, in Tab. 6, we also report the results of AUSH$_{adv}$, AUSH$_{rec}$ and AUSH$_{rec+shill}$, which denote using adversarial loss only, using reconstruction loss only, and using reconstruction and shilling losses, respectively. In these three methods, random sampling is employed. An ordinary neural network (i.e., AUSH$_{rec}$) is outperformed by the complete AUSH (i.e., AUSH$_{rand}$, AUSH$_{rating}$, AUSH$_{pop}$ and AUSH$_{sim}$), showing the effectiveness of our design of tailoring GAN for use in shilling attacks. AUSH$_{adv}$ has the worst attack performance compared to other variations of AUSH, showing that the reconstruction loss also contributes to the attack.

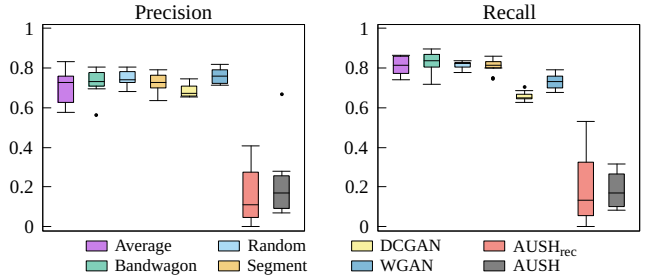

Figure 2: Attack detection of injected profiles on ML-100K. Lower value suggests a better attack model.

5.4 Attack Detection (RQ5)

We apply a state-of-the-art unsupervised attack detector [52] on the injected user profiles generated by different attack models and report the precision and recall on 10 random selected target items. Fig. 2 depicts the detection results on ML-100K. We can observe that the detector performs the worst in terms of precision and recall

Table 4: Attack performance against U-AutoEncoder. Best results are marked in bold, and AUSH results are also marked in bold if they are the second best in each category.

Metric	In-segment Users						All Users					
	Prediction Shift			HR@10			Prediction Shift			HR@10		
Data Set	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive
Model	Random Targets											
AUSH	**1.7661**	**1.3406**	**0.2206**	**0.2465**	**0.5596**	**0.0168**	**1.6184**	**1.1550**	**0.0382**	**0.2006**	**0.3549**	**0.0050**
Segment	0.4721	1.0875	**0.4700**	0.0036	0.5371	**0.7789**	0.3098	0.8886	0.0121	0.0050	**0.3719**	**0.2166**
Average	0.9297	0.9024	0.1311	0.0144	0.1490	0.0000	1.0187	0.9731	0.1514	0.0231	0.1481	0.0000
Random	0.4624	0.7527	0.1262	0.0027	0.0807	0.0000	0.6284	0.8271	0.1200	0.0059	0.1023	0.0000
Bandwagon	0.5501	0.6026	0.0896	0.0012	0.0316	0.0000	0.6311	0.6382	0.0686	0.0062	0.0335	0.0000
DCGAN	-1.064	0.0076	-0.2258	0.0000	0.0000	0.0000	-0.2215	-0.0326	-0.2415	0.0000	0.0000	0.0000
WGAN	1.3940	0.0923	0.1813	0.0000	0.0000	0.1583	1.2985	0.1095	**0.1630**	0.0212	0.0000	0.0928
	Random Long-tail Targets											
AUSH	**3.2274**	**1.7384**	0.3898	**0.6657**	**0.6896**	**0.0000**	**2.9440**	**1.5602**	**-0.0424**	**0.4894**	**0.5149**	**0.0000**
Segment	**3.3397**	1.4665	0.2109	0.6364	0.5570	**0.3733**	3.0081	1.2709	-0.4654	**0.5423**	0.4175	0.0098
Average	3.1671	1.2961	0.2915	0.3897	0.0425	0.0000	**3.0299**	1.3290	**0.2930**	0.4439	0.0851	0.0000
Random	2.5778	1.0348	0.0466	0.1508	0.0259	0.0000	2.5229	1.1324	0.0275	0.1575	0.0815	0.0000
Bandwagon	2.5466	0.8524	0.1227	0.1581	0.0073	0.0000	2.4444	0.9117	0.0509	0.1242	0.0198	0.0000
DCGAN	-0.3896	0.3782	-0.0539	0.0000	0.0000	0.0000	-0.3813	0.4132	0.0496	0.0000	0.0000	0.0000
WGAN	1.3940	0.0923	0.1813	0.0000	0.0000	0.1583	1.2985	0.1095	0.1630	0.0212	0.0000	**0.0928**

Table 5: Attack performance against I-AutoEncoder. Best results are marked in bold, and AUSH results are also marked in bold if they are the second best in each category.

Metric	In-segment Users						All Users					
	Prediction Shift			HR@10			Prediction Shift			HR@10		
Data Set	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive	ML-100K	FilmTrust	Automotive
Model	Random Targets											
AUSH	**1.3746**	**1.4280**	**0.9913**	**0.1488**	**0.9155**	**0.9141**	**1.2180**	**1.3059**	**0.8870**	**0.0990**	**0.8333**	**0.8965**
Segment	0.5137	1.2035	0.4927	0.0086	0.6423	0.7266	0.3232	0.8689	1.6986	0.0274	0.4371	**0.9777**
Average	1.0117	1.4203	0.5394	0.0487	**0.9187**	0.6490	1.1044	**1.3589**	0.5470	**0.1025**	**0.8520**	0.6500
Random	0.6304	1.2210	0.5492	0.0585	0.8307	0.6732	0.7634	1.1630	0.5391	0.0918	0.7477	0.6483
Bandwagon	0.5978	1.2788	0.9718	0.0287	0.8299	0.8608	0.5960	1.2297	**1.8099**	0.0430	0.7825	0.9309
DCGAN	0.0243	-0.0633	0.0046	0.0000	0.0010	0.0050	0.0213	-0.0600	0.0054	0.0000	0.0010	0.0054
WGAN	0.1131	-0.1228	0.0412	0.0000	0.0000	0.0050	0.1045	-0.1142	0.0465	0.0002	0.0008	0.0047
	Random Long-tail Targets											
AUSH	**1.9499**	**1.7052**	**0.9820**	**0.2974**	**0.9239**	**0.8821**	**1.7822**	**1.6019**	**0.8150**	0.2369	0.8396	**0.8623**
Segment	0.5188	1.4510	0.3969	0.0072	0.5835	0.5938	0.3249	1.1385	**1.5154**	0.0344	0.4165	**0.9629**
Average	1.3898	1.6790	0.4245	0.1019	0.9041	0.3726	1.3793	**1.6318**	0.4478	0.1104	**0.8483**	0.3846
Random	0.9227	1.4590	0.46697	0.0401	0.7768	0.4368	1.0349	1.4076	0.4740	0.0900	0.6910	0.4477
Bandwagon	0.6220	1.5672	0.2814	0.0091	0.8390	0.4267	0.7456	1.5190	0.6489	0.0346	0.7728	0.6593
DCGAN	0.0241	0.0119	0.0056	0.0000	0.0000	0.0000	0.0348	0.0114	-0.0029	0.000	0.0005	0.0086
WGAN	0.1096	0.0718	-0.0428	0.0000	0.0000	0.0000	0.1374	0.0728	-0.0364	0.0006	0.0003	0.0018

Table 6: Attack performance on I-AutoEncoder using different sampling strategies and losses in ML-100K. Best results are marked in bold.

Attack Method	In-segment Users		All Users	
	Prediction Shift	HR@10	Prediction Shift	HR@10
AUSH_{rand}	1.6623	0.2231	1.5001	0.1679
AUSH_{rating}	**1.7310**	**0.2735**	**1.5695**	**0.2243**
AUSH_{pop}	1.7252	0.2699	1.5620	0.2212
AUSH_{sim}	1.6752	0.2300	1.5383	0.1992
AUSH_{adv}	1.2960	0.0640	1.3162	0.0881
AUSH_{rec}	1.4980	0.1450	1.4411	0.1441
$\text{AUSH}_{rec+shill}$	1.6569	0.2349	1.5033	0.1849
Segment	0.5163	0.0079	0.3241	0.0309
Average	1.2008	0.0753	1.2419	0.1065
Random	0.7766	0.0493	0.8992	0.0909
Bandwagon	0.6099	0.0189	0.6708	0.0388
DCGAN	0.0242	0.0000	0.0281	0.0000
WGAN	0.1114	0.0000	0.1210	0.0004

Table 7: Two distance measures between injected user profiles and real user profiles in ML-100K.

Measure	AUSH	Average	Bandwagon	Random	Segment	DCGAN	WGAN
TVD	**0.01210**	0.05450	0.05762	0.05704	0.08010	0.11302	0.11598
JS	**0.00215**	0.01162	0.01398	0.01353	0.03461	0.04363	0.04601

against AUSH and AUSH_{rec}, i.e., it fails to distinguish the injected user profiles generated by these two approaches. On the contrary, most of the injected user profiles from conventional attack models can be easily detected. Compared to AUSH, the detection performance of an ordinary neural network such as AUSH_{rec} is unstable over the 10 target items. In the worst case, the injections generated by AUSH_{rec} will be more likely to be detected compared to those produced by AUSH. This observation further verifies the ability of our special designed AUSH to generate virtually undetectable injections in shilling attack.

Additionally, we run a set of similarity tests to further demonstrate the undetectability of AUSH. We generate as many fake user profiles as the population of real users, i.e., 943 fake users for ML-100K. We compute the distribution $\mathbf{p}(v) \in \mathcal{R}^6$ for each item v, where $\mathbf{p}_i$ is the percentage of real ratings on v with value $i, i = \{0, 1, 2, 3, 4, 5\}$. We also compute the distribution $\mathbf{q}(v)$ in the injected user profiles. Following Christakopoulou and Banerjee [15, 16], we compute two distance measures, i.e., Total Variation Distance and Jensen-Shannon divergence:

$$TVD = \sum_{v \in \mathcal{V}} |\mathbf{p}(v) - \mathbf{q}(v)| \;/\; |\mathcal{V}|$$

$$JS = \sum_{v \in \mathcal{V}} \Big(KL\big(\mathbf{p}(v) \big\| \mathbf{m}(v)\big) + KL\big(\mathbf{q}(v) \big\| \mathbf{m}(v)\big) \Big) \Big/ |\mathcal{V}|$$

where $\mathbf{m}(v) = \big(\mathbf{p}(v) + \mathbf{q}(v)\big)/2$ and $KL(\cdot)$ represents the Kullback-Leibler divergence, between fake profiles and real profiles. As shown

in Tab. 7, the fake profiles generated by AUSH have the smallest TVD and JS. Since TVD and JS measure the difference of overall rating distributions, we can see that AUSH can preserve the distribution patterns and diversity of the original rating space.

6 CONCLUSION

In this paper, we present a novel shilling attack framework AUSH. We design a minimax game to let each of the attack profile generator and fake profile discriminator iteratively strikes to improve itself and beat the other one. We additionally employ a reconstruction loss and a shilling loss to help generate "perfect" fake profiles and achieve secondary attack goals. The experimental results show the superiority of AUSH. In the future, we plan to design more sophisticated mechanisms for learning selected items instead of selection by human. This way, the ultimate goal of the attack can not be easily inferred from the selected items and AUSH can become even more undetectable.

REFERENCES

[1] Charu C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016.
[2] Ming Pang, Wei Gao, Min Tao, and Zhi-Hua Zhou. Unorganized malicious attacks detection. In *NeurIPS*, pages 6976–6985, 2018.
[3] Joseph A. Calandrino, Ann Kilzer, Arvind Narayanan, Edward W. Felten, and Vitaly Shmatikov. "you might also like:" privacy risks of collaborative filtering. In *IEEE Symposium on Security and Privacy*, pages 231–246, 2011.
[4] Ihsan Gunes, Cihan Kaleli, Alper Bilge, and Huseyin Polat. Shilling attacks against recommender systems: a comprehensive survey. *Artif. Intell. Rev.*, 42(4): 767–799, 2014.
[5] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. Data poisoning attacks on factorization-based collaborative filtering. In *NIPS*, pages 1885–1893, 2016.
[6] Robin D. Burke, Bamshad Mobasher, Runa Bhaumik, and Chad Williams. Segment-based injection attacks against collaborative filtering recommender systems. In *ICDM*, pages 577–580, 2005.
[7] Xinyu Xing, Wei Meng, Dan Doozan, Alex C. Snoeren, Nick Feamster, and Wenke Lee. Take this personally: Pollution attacks on personalized services. In *USENIX Security Symposium*, pages 671–686, 2013.
[8] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. Fake co-visitation injection attacks to recommender systems. In *NDSS*, 2017.
[9] Shyong K. Lam and John Riedl. Shilling recommender systems for fun and profit. In *WWW*, pages 393–402, 2004.
[10] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey. *arXiv Preprint*, 2018. URL https://arxiv.org/abs/1810.00069.
[11] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Networks Learn. Syst.*, 30(9): 2805–2824, 2019.
[12] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Trans. Evolutionary Computation*, 23(5): 828–841, 2019.
[13] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani B. Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. In *EMNLP*, pages 2890–2896, 2018.
[14] Jeff J. Sandvig, Bamshad Mobasher, and Robin D. Burke. A survey of collaborative recommendation and the robustness of model-based algorithms. *IEEE Data Eng. Bull.*, 31(2):3–13, 2008.
[15] Konstantina Christakopoulou and Arindam Banerjee. Adversarial recommendation: Attack of the learned fake users. *arXiv Preprint*, 2018. URL https://arxiv.org/abs/1809.08336.
[16] Konstantina Christakopoulou and Arindam Banerjee. Adversarial attacks on an oblivious recommender. In *RecSys*, pages 322–330, 2019.
[17] Bamshad Mobasher, Robin D. Burke, Runa Bhaumik, and Chad Williams. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Trans. Internet Techn.*, 7(4):23, 2007.
[18] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38, 2019.
[19] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.
[20] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *SIGMOD Conference*, pages 835–850, 2017.
[21] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, pages 173–182, 2017.
[22] Trinh Xuan Tuan and Tu Minh Phuong. 3d convolutional networks for session-based recommendation with content features. In *RecSys*, pages 138–146, 2017.
[23] Peijie Sun, Le Wu, and Meng Wang. Attentive recurrent social recommendation. In *SIGIR*, pages 185–194, 2018.
[24] Shaohua Fan, Junxiong Zhu, Xiaotian Han, Chuan Shi, Linmei Hu, Biyu Ma, and Yongliang Li. Metapath-guided heterogeneous graph neural network for intent recommendation. In *KDD*, pages 2478–2486, 2019.
[25] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, and Lexing Xie. Autorec: Autoencoders meet collaborative filtering. In *WWW (Companion Volume)*, pages 111–112, 2015.
[26] Hui Li, Yanlin Wang, Ziyu Lyu, and Jieming Shi. Multi-task learning for recommendation over heterogeneous information network. *IEEE Trans. Knowl. Data Eng.*, 2020.
[27] Wenqi Wang, Lina Wang, Run Wang, Zhibo Wang, and Aoshuang Ye. Towards a robust deep neural network in texts: A survey. *arXiv Preprint*, 2019. URL https://arxiv.org/abs/1902.07285.
[28] Wei Emma Zhang, Quan Z Sheng, Ahoud Alhazmi, and Chenliang Li. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Trans. Intell. Syst. Technol.*, 11(3), 2020.
[29] Nilesh N. Dalvi, Pedro M. Domingos, Mausam, Sumit K. Sanghai, and Deepak Verma. Adversarial classification. In *KDD*, pages 99–108, 2004.
[30] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *ECML/PKDD (3)*, volume 8190, pages 387–402, 2013.
[31] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In *AsiaCCS*, pages 16–25, 2006.
[32] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Mach. Learn.*, 81(2):121–148, 2010.
[33] Fabio Roli, Battista Biggio, and Giorgio Fumera. Pattern recognition systems under attack. In *CIARP (1)*, volume 8258, pages 1–8, 2013.
[34] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858, 2017.
[35] Ming-Yu Liu and Oncel Tuzel. Coupled generative adversarial networks. In *NIPS*, pages 469–477, 2016.
[36] Cheng Wang, Mathias Niepert, and Hui Li. Recsys-dan: Discriminative adversarial networks for cross-domain recommender systems. *IEEE Trans. Neural Netw. Learning Syst.*, 2019.
[37] Yongjun Hong, Uiwon Hwang, Jaeyoon Yoo, and Sungroh Yoon. How generative adversarial networks and their variants work: An overview. *ACM Comput. Surv.*, 52(1):10:1–10:43, 2019.
[38] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.
[39] Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. *arXiv Preprint*, 2017. URL https://arxiv.org/abs/1701.07875.
[40] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *ICLR*, 2018.
[41] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. In *IJCAI*, pages 3905–3911, 2018.
[42] Michael P. O'Mahony, Neil J. Hurley, and Guenole C. M. Silvestre. Recommender systems: Attack types and strategies. In *AAAI*, pages 334–339, 2005.
[43] Michael P. O'Mahony, Neil J. Hurley, Nicholas Kushmerick, and Guenole C. M. Silvestre. Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Techn.*, 4(4):344–377, 2004.
[44] Robin Burke, Bamshad Mobasher, and Runa Bhaumik. Limited knowledge shilling attacks in collaborative filtering systems. In *ITWP@IJCAI*, 2005.
[45] David C. Wilson and Carlos E. Seminario. When power users attack: assessing impacts in collaborative recommender systems. In *RecSys*, pages 427–430, 2013.
[46] Carlos E. Seminario and David C. Wilson. Attacking item-based recommender systems with power items. In *RecSys*, pages 57–64, 2014.
[47] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. Poisoning attacks to graph-based recommender systems. In *ACSAC*, pages 381–392, 2018.
[48] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.
[49] Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2000.
[50] Gintare Karolina Dziugaite and Daniel M. Roy. Neural network matrix factorization, 2015. URL https://arxiv.org/abs/1511.06443.
[51] Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. In *NIPS*, pages 5767–5777, 2017.
[52] Yongfeng Zhang, Yunzhi Tan, Min Zhang, Yiqun Liu, Tat-Seng Chua, and Shaoping Ma. Catch the black sheep: Unified framework for shilling attack detection based on fraudulent action propagation. In *IJCAI*, pages 2408–2414, 2015.