Triple Adversarial Learning for Influence based Poisoning Attack in Recommender Systems

Chenwang Wu[1], Defu Lian[1,2,*], Yong Ge[3], Zhihao Zhu[1], Enhong Chen[1,2]
[1] School of Data Science, University of Science and Technology of China
[2] School of Computer Science and Technology, University of Science and Technology of China
[3] University of Arizona
{wcw1996,zzh98}@mail.ustc.edu.cn,{liandefu,cheneh}@ustc.edu.cn,yongge@arizona.edu

ABSTRACT

As an important means to solve information overload, recommender systems have been widely applied in many fields, such as e-commerce and advertising. However, recent studies have shown that recommender systems are vulnerable to poisoning attacks; that is, injecting a group of carefully designed user profiles into the recommender system can severely affect recommendation quality. Despite the development from shilling attacks to optimization-based attacks, the imperceptibility and harmfulness of the generated data in most attacks are arduous to balance. To this end, we propose a triple adversarial learning for influence based poisoning attack (TrialAttack), a flexible end-to-end poisoning framework to generate non-notable and harmful user profiles. Specifically, given the input noise, TrialAttack directly generates malicious users through triple adversarial learning of the generator, discriminator, and influence module. Besides, to provide reliable influence for TrialAttack training, we explore a new approximation approach for estimating each fake user's influence. Through theoretical analysis, we prove that the distribution characterized by TrialAttack approximates to the rating distribution of real users under the premise of performing an efficient attack. This property allows the injected users to attack in an unremarkable way. Experiments on three real-world datasets show that TrialAttack's attack performance outperforms state-of-the-art attacks, and the generated fake profiles are more difficult to detect compared to baselines.

CCS CONCEPTS

• **Security and privacy → Web application security**.

KEYWORDS

Poisoning Attacks, Recommender Systems, Adversarial Learning.

ACM Reference Format:
Chenwang Wu, Defu Lian, Yong Ge, Zhihao Zhu, and Enhong Chen. 2021. Triple Adversarial Learning for Influence based Poisoning Attack in Recommender Systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467335

* Corresponding Author.

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467335

1 INTRODUCTION

With the rapid development of the Internet industry and information technology, information overload is a serious problem that we must face in processing massive information. Fortunately, the recommendation technology provides crucial support for alleviating the adverse effects of information overload [14, 24], and it has played an irreplaceable role in many applications such as e-commerce [29] and mobility prediction [23].

However, while the recommender system's openness and collaboration bring convenience to users, they also pose severe security problems. It has been demonstrated that recommender systems are vulnerable to poisoning attacks [12, 13, 22, 30, 34, 36]; that is, injecting a small number of well-designed fake users into the recommender system can easily deceive the recommendation model. On the one hand, such vulnerability is irrefutable, encouraging researchers to learn and explore model robustness. On the other hand, many merchants in various systems are well motivated to take advantage of data poisoning attacks to promote their products or demote their competitors' ones. For example, Sony Pictures made some specific movies widely recommended by falsifying movie reviews[1]. Therefore, from both academic research and real-world practice perspective, it is very needed to study the data poisoning attacks on recommender systems, which will reveal critical insights and strategies for developing effective defense methods.

A sophisticated poisoning attack method should not only effectively achieve the attacking target but also work in a non-notable way [12, 28]. Most extant methods for poisoning attacks focus on optimizing the defined attack objects[13, 22]; however, the generated poisoning profiles often lack diversity and seriously deviate from normal users' data [7]. This drawback causes poisoning attacks to be easily detected by defense tools or mechanisms [7, 26]. Although shilling attacks utilize the statistical information to generate users closer to real users, they have not received theoretical analysis and cannot optimize the specific model, which significantly weakens the attack's destructiveness [32].

Given the deficiency of existing attacks, a few recent works [9, 25] take advantage of generative adversarial networks (GAN) in modeling data distribution [3, 8, 15, 18, 33] and consider using GAN's generative ability to produce malicious users. For example, Christakopoulou et al. [9] used DCGAN to generate "real" users and

[1] http://news.bbc.co.uk/2/hi/entertainment/1368666.stm

then patched them based on specific attack intent. Such step-by-step processing mainly focuses on the second optimization stage, and in the worst case, it will degenerate to traditional model-based optimization attacks. Lin et al. [25] incorporated shilling loss into the generator, but it is still a shilling attack in essence, and the attack performance is formidable to guarantee.

To address the aforementioned challenges, we propose triple adversarial learning to jointly attack optimization and GAN toward generating harmful and obscure poisoning data. However, it is challenging to assess the user's reward for the attack. A straightforward approach is to retrain the contaminated dataset and then evaluate the attack performance. This approach needs the recommendation model to retrain many times in each GAN training epoch, which is computationally infeasible. This is why the current GAN-based attack is limited to adding model-agnostic shilling loss [25]. Inspired by the efficiency of influence function in tracking the effect of training samples on decision-making [19], we propose an approximate method for calculating the attack influence of poisoning users, which avoids time-consuming retraining. Furthermore, we use neural networks to model influence as an influence module and integrate it with GAN to develop a triple adversarial learning for influence based poisoning attack (TrialAttack). Similar to the TripleGAN [8] that plays a minimax game with three players, TrialAttack allows the generator, discriminator, and influence module to compete with one another through a triple adversarial mechanism. Through theoretical and experimental analysis, we confirm that TrialAttack can generate non-notable and effective poisoning data that achieve the attacking objective.

In summary, the main contributions of our work are as follows.

- We propose a flexible end-to-end framework, TrialAttack, for the poisoning attack in recommender systems. Through triple adversarial learning, TrialAttack can effectively balance the imperceptibility and harmfulness of generated users. Besides, our framework can generate fake users more efficiently compared with existing model-dependent attacks.
- We theoretically prove that under the premise of effective attack, the distribution characterized by TrialAttack could approximate the real distribution. This property allows fake users to inject unobtrusively, significantly increasing the difficulty of defense.
- In TrialAttack, we contribute a new approximation method for estimating the influence of poisoning data on the attack objective. Given any user, it can effectively evaluate the user's damage to the recommender system.
- We evaluate TrialAttack on three real-world datasets. The experimental results show that our attack is superior to the state-of-the-art methods, and the generated fake users are more difficult to detect compared to baselines.

2 RELATED WORK

In recent years, people have intensively studied poisoning attacks, attack detection, and recommender system robustness [28].

Poisoning Attack: Early shilling attacks, such as Average attack [28] and Bandwagon attack [4], were only based on the fact that similar users have similar interests and did not optimize the specific model, render the attack's damage not satisfactory. In 2016, Li et al. [22] proposed a poisoning attack on Factorization-Based collaborative recommendation to achieve integrity attacks and availability attacks by minimizing attack objectives. It laid the foundation for subsequent research on model-based attacks [12, 13, 30, 34, 36]. Yang et al. [34] modeled the attack as a constrained linear optimization problem and injected limited fake co-visitation to deceive the recommender system. Fang et al. [13] attacked graph-based recommender systems, turning the poisoning profiles into a minimization problem and solving it using projected gradient descent. TNA [12] achieves outstanding performance by selecting and evaluating a subset of influential users. Considering that malicious users are often different from normal data, [9, 25] leveraged GANs to study the distribution of real users to generate fake users that are arduous to detect, and [11] obtained profiles from different domains to execute attacks. More recently, modeling the attack trajectory as a Markov decision process and using reinforcement learning to attack has gradually captured attention [30, 36].

Attack Detection: Traditional detection methods [4, 27] mainly extract features from the rating matrix to determine whether they are malicious users. However, due to the imbalance and insufficiency of data, these methods cannot achieve sufficiently satisfactory results. Aktukmak et al.[2] established a probability model to detect malicious users. The Borderline-SMOTE method can solve data imbalance and further fine-tune the detection result for the target item [40]. Zhang et al. [38] proposed the method of label propagation to spread the known malicious user labels. Also, it is reasonable to monitor the data flow for a while [37, 39].

Robustness of Recommender Systems: Recent studies have shown that adversarial examples are not bugs but meaningful data characteristics easily ignored by the model [17]. Thus, the current work mainly focuses on adversarial training, desiring to improve the recommendation algorithms' tolerance to data changes [6, 16, 31, 35]. Deldzoo et al. [10] used regression models to analyze the relationship between URM data characteristics and the vulnerability of collaborative filtering.

3 PROBLEM DEFINITION

3.1 Threat Model

Attack goal: According to the attack's effect, the poisoning attacks can be divided into promotion attacks, demotion attacks, and availability attacks [28]. Among them, promotion attacks and demotion attacks are designed to promote or demote the recommended frequency of the target items [28], and availability attacks aim to maximize the prediction errors for unseen items [22], thereby reducing trust in the system. The proposed TrialAttack can efficiently perform these attacks by designing different attack objectives. For simplicity, we focus on promotion attacks in the paper.

Attack knowledge: We first consider the full-knowledge attack that most existing works [9, 12, 13, 22, 25] care about. In a full-knowledge attack, the attacker can capture all users' historical behaviors, including product browsing records or ratings (e.g., attackers can obtain them from underground markets, internal employees). Also, the attacker knows the algorithm and parameters of the recommender system. Although many real-world recommenders are black boxes, some systems still reveal their recommendation algorithms, e.g.,

Amazon [29] and Netflix [14] have reported recommendation algorithms. More importantly, it provides the worst-case assumption for defense, revealing important insight and findings for attacking and defending many real recommenders. Thus, investigating the full-knowledge attack is practically needed.

Second, we study more practical partial-knowledge attacks. In this scenario, only partial historical behaviors are exposed to the attacker, and the attacker only knows the type of the recommender system and cannot catch the internal parameters. Its key idea is to transform this type of attack into a full-knowledge attack by estimating the target model. Specifically, given that the target model's parameters are unknown, the attacker uses the partial data obtained by web crawlers to train a local simulator that uses the same algorithm as the target model but has attacker-defined parameters. At this time, the local simulator is a white box for the attacker. Accordingly, the attacker can execute a full-knowledge attack based on the local simulator to generate fake users and inject these locally generated users into the target model to execute attacks.

Attack capability: In theory, the higher permissions an attacker has, the more vulnerable the target model is. However, too many injections of fake users are inoperable and easily perceived by detection procedures [38, 40]. Therefore, we limit the attacker to inject up to n' fake users into the recommender system.

3.2 Poisoning Attack: a Bi-level Optimization Problem

Most of the current work is still focused on the heuristic-driven shilling attack, and the research on the optimization-based attack is slightly insufficient. A daunting challenge is that it needs to solve a bi-level optimization problem [32]:

$$\max_{X'} \mathcal{L}_{atk}(X, \theta_R), \tag{1}$$

subject to

$$\theta_R = \arg\min_{\theta_R}(\mathcal{L}_{train}(X, \theta_R) + \mathcal{L}_{train}(X', \theta_R)). \tag{2}$$

Here $X \in \mathbb{R}^{n \times m}$ is the currently observed rating matrix, where $X_{i,j}$ is the rating given by user i to item j, n is the number of normal users, and m is the number of items . $X' \in \mathbb{R}^{n' \times m}$ is the rating matrix of fake users that we need to solve; θ_R represents the parameters of the recommender model. $\mathcal{L}_{atk}(X, \theta_R)$ is the objective function of the attack. For the promotion attack in the paper, we desire the target item to appear in the top-k recommendation list of as many users as possible, so we define $\mathcal{L}_{atk}(X, \theta_R)$ as follows:

$$\mathcal{L}_{atk}(X, \theta_R) = \sum_{i=1}^{n} \sum_{j \in \mathcal{V}_{i,k}} -ln\sigma(\hat{r}_{i,j} - \hat{r}_{i,t}), \tag{3}$$

where $\sigma(x) = 1/(1 + e^{-x})$, $\mathcal{V}_{i,k}$ is the top-k recommendation list of user i, and t is the target item. Here $\mathcal{V}_{i,k}$ changes dynamically in the solving process. It implies that at some stage of optimization, if $j \in \mathcal{V}_{i,k}$, $\hat{r}_{i,t} - \hat{r}_{i,j} > 0$, then the target item t will be in the top-k recommendation list. Furthermore, if the loss $\mathcal{L}_{atk}$ is larger, the target item t will tend to hold a higher rank.

However, considering the bi-level situation, the model needs to be retrained after poison, which will result in an optimized solution based on the current θ_R is only an approximation of the optimal solution. Moreover, most optimization methods use the gradient or other moments of the differentiable objective to search in the neighborhood of discrete data[41]. For discrete ratings where the gradients are undefined, it will undoubtedly increase the difficulty of optimization. Thus, similar to [12], we perform a relaxing operation on user-item ratings. Assuming that the discrete rating $r_{i,j} \in \{0, 1, ..., r_{max}\}$, we relax it to the continuous variable of $[0, r_{max}]$. After generating continuous user ratings, we use a projection operator to clip them to the rational discrete ratings.

4 TRIPLE ADVERSARIAL LEARNING FOR INFLUENCE BASED ATTACK

4.1 Overview

In the paper, we propose a triple adversarial learning for influence based poisoning attack (TrialAttack), which incorporates an attack influence module in the GAN. Inspired by TripleGAN [8], the proposed TrialAttack will extend the traditional GAN to three-player games: generator G, discriminator D, and influence module I. The specific framework is shown in Fig. 1, which mainly includes three components:

- **Influence module**: On the one hand, the influence module should appraise the generated user's destructive power and guide the generator to produce the user with influence as large as possible. To this end, we propose a new solver to approximate the attack influence of the fake user. On the other hand, influence can be regarded as the user's potential feature and used as the discriminator's input. Thus, it aims to make the predicted influence real without being detected by the discriminator.
- **Generator**: Given the input noise e, the generator is dedicated to generating close-to-real user profile $u \in \mathbb{R}^m$ with attack influence as large as possible, where u_i represents the rating for item i. To ease its learning pressure, we design a diverse noise sampling to produce input noise e.
- **Discriminator**: The discriminator should be devoted to distinguishing the generated profiles from the real. Aside from the user profile, the corresponding influence is also a key factor for judgment. Therefore, the discriminator predicts real if and only if the user profile and influence are both real. Here we take the calculated influence as real and the predicted one as fake.

More recently, some studies [9, 11, 25] have also proposed generating imperceptible poisoning profiles, but our work is quite different from them. The poisoning users in [11] are real users in different domains; in contrast, TrialAttack generates fake users through a GAN-like adversarial structure without relying on other-domain datasets. For [9, 25] that also use generative networks, they utilize the traditional GAN structure that cannot guarantee attack performance, and our TrialAttack incorporates an influence module to generate realistic users while maximizing attack damage.

4.2 Influence Module

4.2.1 Influence in Recommender Systems. The most realistic idea to understand the impact of upweighting or perturbing a training sample on the test sample is to retrain the changed dataset for effect estimation. However, it is impractical to retrain massive amounts of data in the era of big data. At this point, the influence function [19, 20] provides an efficient solution.

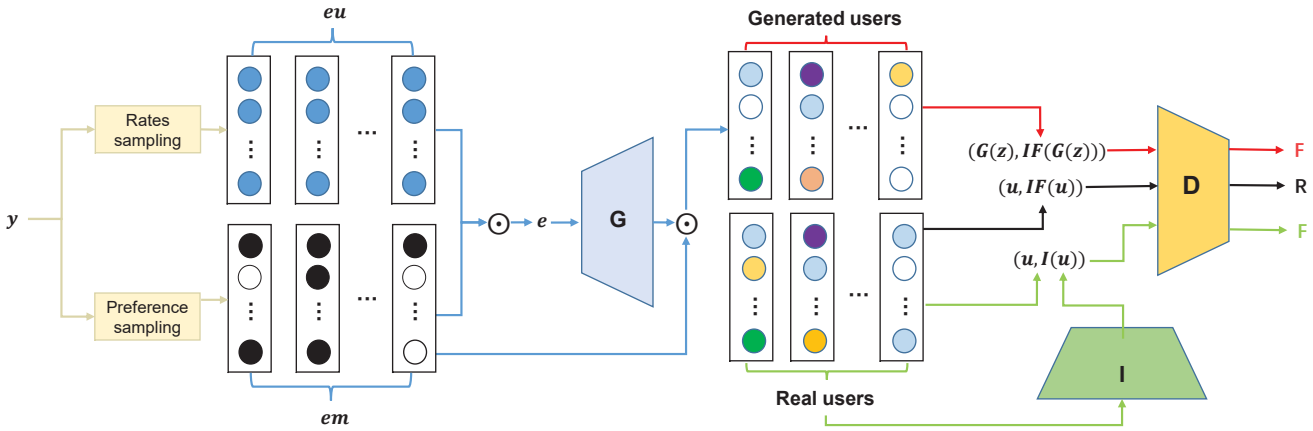


Figure 1: The framework of TrialAttack. If and only if both user and influence are real (the black lines), the discriminator D will consider it to be real (R), and the rest (red and green lines) will be considered to be fake (F).

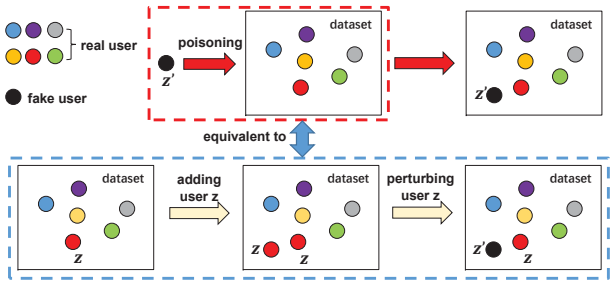


Figure 2: An example of poisoning a fake user. Poisoning a malicious user z' (black node) to the recommender system is equivalent to adding an existing normal user z (red node) first, then perturbing the added z until it is the same as z'.

For a sample z in the training set, if it is upweighted with a small value ϵ, the changed parameter $\hat{\theta}_{z,\epsilon} = \arg\min_{\theta\in\Theta} \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(z_i,\theta) + \epsilon\mathcal{L}(z,\theta)$, where $\mathcal{L}$ is the training loss of the model. And then according to [19], the influence function of upweighting z on a test sample z_{test} can be defined as follows:

$$\mathcal{I}_{up,loss}(z) := \frac{d\mathcal{L}(z_{test},\hat{\theta}_{z,\epsilon})}{d\epsilon}|_{\epsilon=0} = -\nabla_\theta\mathcal{L}(z_{test},\hat{\theta})^T H_{\hat{\theta}}^{-1}\nabla_\theta\mathcal{L}(z,\hat{\theta}),$$

where $H_{\hat{\theta}} := \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2\mathcal{L}(z_i,\hat{\theta})$ is the Hessian matrix of the training loss. Besides, consider the scenario of adding a small perturbation δ to the sample z, the influence function on the sample z_{test} can be defined as:

$$\mathcal{I}_{pert,loss}(z) := -\frac{1}{n}\nabla_\theta\mathcal{L}(z_{test},\hat{\theta})^T H_{\hat{\theta}}^{-1}\nabla_z\nabla_\theta\mathcal{L}(z,\hat{\theta})\delta.$$

The specific proof can be found in [19].

The estimation of model oscillation by the influence function can be used as an effective tool for evaluating the attack effect. Recently, there are a few related works [12, 36] that use influence to poison recommender systems. They appraise the influence of normal users [12] or items [36] on recommendations. Since such users and items exist in the system objectively, the influence function that aims to estimate the training sample's influence can be applied directly.

Unlike their research objects, we desire to calculate the destruction of a malicious user z' who has never appeared before, so simple upweighting or perturbing users is not feasible. To tackle this problem, we decompose the operation of poisoning a user into two stages: (1) choosing an existing user z and adding it to the dataset; (2) perturbing the added user z to be the same as the fake user z'. An example is illustrated in Fig. 2. Thus, the influence of a poisoning user will be the sum of these two stages:

$$\mathcal{I}_{poison}(z') = \mathcal{I}_{add,loss}(z) + \mathcal{I}_{pert,loss}(z).$$

First, for a system of n users, the weight of each training user is $\frac{1}{n}$, so adding an existing user is equivalent to upweighting this user by $\epsilon \approx \frac{1}{n}$. In particular, by computing influence on the loss of attack objective $\mathcal{L}_{atk}(X,\theta_R)$ in Eq. 3, we can linearly approximate the influence $\mathcal{I}_{add,loss}$ of adding a training user z:

$$\mathcal{I}_{add,loss}(z) \approx \frac{1}{n}\mathcal{I}_{up,loss}(z) = -\frac{1}{n}\mathcal{L}_{atk}(X,\hat{\theta}_R)^T H_{\hat{\theta}_R}^{-1}\nabla_{\theta_R}\mathcal{L}(z,\hat{\theta}_R).$$

Second, perturbing the newly added user z to make it the same as the poisoning user z'. Therefore, the size of perturbation $\delta(z \mapsto z')$ should be $\delta = z' - z$, then the influence of perturbing $\mathcal{I}_{pert,loss}(z)$ is as follows:

$$\mathcal{I}_{pert,loss}(z) = -\frac{1}{n}\mathcal{L}_{atk}(X,\hat{\theta}_R)^T H_{\hat{\theta}_R}^{-1}\nabla_z\nabla_{\theta_R}\mathcal{L}(z,\hat{\theta}_R)(z'-z).$$

To sum up, we can approximate the attack damage of any fake user z' (also applicable to the normal user):

$$\mathcal{I}_{poison}(z') = -\frac{1}{n}\nabla_{\theta_R}\mathcal{L}_{atk}(X,\hat{\theta}_R)^T H_{\hat{\theta}_R}^{-1}[\nabla_{\theta_R}\mathcal{L}(z,\hat{\theta}_R) + \nabla_z\nabla_{\theta_R}\mathcal{L}(z,\hat{\theta}_R)(z'-z)].$$

To satisfy the linear assumption [19] of the influence function, the selected user z should be as similar to z' as possible. Therefore, for a candidate fake user z', ideally, it needs to traverse the entire dataset to find a user similar to it and then calculate the influence $\mathcal{I}_{poison}(z')$. Even though implicit Hessian-vector products (IHVP) [1] can be used to approximate $H_{\hat{\theta}_R}^{-1}$ efficiently, in the worst case, it takes $O(nm + tp)$ time concerning the MF-based recommender system, where t is the number of iteration (could be large) of IHVP,

and $p = nd + md$ is the total dimension of the model parameters. For efficient calculation, we only select a user z_{min} that is expected to be close to all feasible fake users.

Proposition 4.1. *In a promotion attack where the target item is t, suppose $p(z')$ is the distribution of fake users, $\mu \in \mathbb{R}^m$, and each μ_i represents the mode of rating on item i, $e_t \in \{0, r_{max}\}^m$ whose dimension t is r_{max}, and the others are 0, then $z_{min} = \arg\min_z E_{z' \sim p(z')} \|z' - z\|_0 = \Pi(\mu + \mathbf{e_t})$, where $\Pi(z)$ project each z_i to reasonable discrete rating.*

The proof is in Appendix A.1. Therefore, we directly inject user z_{min} and then retrain the recommender system. This design avoids the evaluation of $\mathcal{I}_{add,loss}(z)$ and reduces the time to $O(p)$. Finally, the influence of poisoning user z' used in the paper is given by

$$\mathcal{I}(z') = -\frac{1}{n} \nabla_{\theta_R} \mathcal{L}_{atk}(X, \hat{\theta}_R)^T H_{\hat{\theta}_R}^{-1} \nabla_z \nabla_{\theta_R} \mathcal{L}(z_{min}, \hat{\theta}_R)(z' - z_{min}). \tag{4}$$

Proposition 4.2. *Assume that $p(z')$ is the distribution of fake users; the influence error of user z' is $Err_I(z', z)$ when selecting user z. If the perturbation $\delta = z' - z \to 0$, then $E_{z' \sim p(z')}(Err_I(z', z_{min})) \leq E_{z' \sim p(z')}(Err_I(z', z))$.*

The proof is in Appendix A.1. It tells us that when the perturbation is minimal, the estimation error of influence by choosing z_{min} is no worse than the user selected from the original dataset.

4.2.2 Influence Model. Based on the fact that the larger the influence, the higher the attack revenue, we utilize the attack influence defined in Eq. 4 to conduct the generator to produce influential users. Here, we leverage a neural network to model influence as an influence model and combine it with GAN. This design allows the generator to receive feedback from the influence model for end-to-end training. Therefore, the module's forecast of influence should be accurate and real to provide a reliable guarantee for generator training.

The accuracy can be guaranteed by employing standard supervised learning. In each epoch, we randomly sample a group of real users $u_I \sim p(u)$, and then minimize the gaps between the real influence IF_I (calculated by Eq. 4) and predicted influence $I(u_I)$:

$$\min \mathcal{L}_{sup}^I = E_{(u_I, IF_I) \sim p(u, IF)} (IF_I - I(u_I))^2.$$

For the assurance of reality, the influence model can revise its parameters through the feedback of the discriminator and attempt to confuse the discriminator into believing that the predicted influence is real ($D(u_I, I(u_I))$ is close to 1):

$$\min \mathcal{L}_{gan}^I = E_{(u_I, I(u_I)) \sim p_I(u, IF)} \log(1 - D(u_I, I(u_I))).$$

4.3 Generator

The generator is devoted to generating non-notable and malicious poisoning users. Unfortunately, discrete data in the recommender system poses severe challenges for the generator to create user profiles: (1) For a tremendous amount of items, it is not easy to learn each item's preference specifically [5]. (2) Insufficient diversity of generated users due to mode collapse [8]. Even if fake users are aggressive, they are easily detected because of low diversity [7, 26].

In response to these challenges, we propose a diverse noise sampling to generate the input noise $e \in \mathbb{R}^m$ of the generator, where e_i is the rating given to item i. Diverse noise sampling mainly includes rating sampling and preference sampling, as shown in the left part of Fig. 1. The specific process is as follows: (1) The K-means algorithm is used to cluster all real users into different groups. (2) We use rating sampling to generate initial noise $eu \in \mathbb{R}^m$, where eu_i is sampled from the rating distribution of item i. (3) Since it is impractical for a user to rate all items, we use a preference sampling to select items that the user may rate. Here we generate an indicator vector $em \in \{0, 1\}^m$ to specify whether the user selects item i ($em_i = 1$) or not ($em_i = 0$). To be specific, first, we randomly select a group j, then the number of selected items $\sum_i em_i$ is set to the average number of ratings per user in group j, and the probability of item i being selected is $p(em_i = 1) = \frac{\mu_{j,i}}{\sum_k \mu_{j,k}}$, where $\mu_{j,i}$ is the mean ratings of item i in group j. Besides, we set $em_t = 1$ for target item t. (4) We only focus on the items selected by preference sampling, so the final noise $e = eu \odot em$, where $\odot$ is element-wise multiplication (note that the generated users also only care about these preferred items). On the one hand, diverse noise sampling allows the generator to focus on low-dimensional items that may be rated, thereby alleviating the learning pressure. On the other hand, diversity can be guaranteed through such a series of operations (e.g., clustering).

Addressing the above challenges brings us to focus on generating users. First, the generator needs to camouflage the generated user so that the discriminator mistakes it to be real:

$$\min \mathcal{L}_{gan}^G = E_{(u_G, IF_G) \sim P_G(u, IF)} \log(1 - D(u_G, IF_G)).$$

Besides, the generated users should be malicious and aggressive. Based on the strong correlation of influence and attack loss [19], we take the influence module's predictive feedback as attack loss $\mathcal{L}_{atk}^G$ and then guide the generator to produce users with large influence:

$$\max \mathcal{L}_{atk}^G = E_{u_G \sim p_G(u)} I(u_G).$$

Lastly, to prevent the learning distribution from deviating from the real distribution due to the incorporation of attack loss, we integrate reconstruction loss $\mathcal{L}_{rec}^G$ to the generator. Since the input noise is sampled from real distribution, the reconstruction loss is conducive to generate ratings close to reality:

$$\min \mathcal{L}_{rec}^G = E_{e_G \sim p_G(e), u_G \sim p_G(u)} \|u_G - e_G\|_2 .$$

4.4 Discriminator

The discriminator is devoted to distinguishing generated users from real users to avoid being deceived by the generator and influence module. Only when the input user and influence are real, the discriminator will treat it as real. Therefore, the adversarial loss of the discriminator is defined as follows:

$$\begin{aligned} \max \mathcal{L}_{gan}^D = & E_{(u, IF) \sim p(u, IF)} \log D(u, IF) + \\ & \alpha E_{(u_I, IF_I) \sim P_I(u, IF)} \log(1 - D(u_I, IF_I)) + \\ & (1 - \alpha) E_{(u_G, IF_G) \sim p_G(u, IF)} \log(1 - D(u_G, IF_G)). \end{aligned}$$

In summary, the utility $U(D, G, I)$ in TrialAttack can be formulated as a three-party minimax game:

$$\min_{G,I} \max_D U(D, G, I) = \mathcal{L}_{gan}^D + \gamma \mathcal{L}_{sup}^I - \beta_1 \mathcal{L}_{atk}^G + \beta_2 \mathcal{L}_{rec}^G.$$

The training process of TrialAttack is shown in Alg. 1. When the training is completed, we first sample N ($N > n'$) noises through

Algorithm 1: Training process of TrialAttack

Poison z_{min} and then retrain the recommender system;
for *T steps* **do**
  Sample a batch noise e_G of according to diverse noise sampling, and generate fake users $u_G = G(e_G)$, and calculate influence IF_G according to Eq. 4;
  Sample a batch of real users $u \sim p(u)$ and calculate influence IF according to Eq. 4;
  Sample a batch of real users $u_I \sim p(u)$, calculate influence IF_I according to Eq. 4 and predict influence $I(u_I) \sim I(u)$;
  for T_D *steps* **do**
    Update the discriminator by ascending stochastic gradient $\nabla_{\theta_D} \mathcal{L}^D_{gan}$;
  end
  for T_G *steps* **do**
    Update the generator by descending stochastic gradient $\nabla_{\theta_G}((1-\alpha)\mathcal{L}^G_{gan} - \beta_1 \mathcal{L}^G_{atk} + \beta_2 \mathcal{L}^G_{rec})$;
  end
  for T_I *steps* **do**
    Update the influence module by descending stochastic gradient $\nabla_{\theta_I}(\alpha \mathcal{L}^I_{gan} + \gamma \mathcal{L}^I_{sup})$;
  end
end

diverse noise sampling. Then we input them into the generator to obtain N malicious user profiles while getting their influence through the influence module. Finally, group-level sampling is used: (1) selecting a group according to the user proportion of each group, and (2) selecting the most influential user in that group. The process is repeated until n' users are selected.

It is worth mentioning that although this paper mainly focuses on promotion attacks, other attacks can also be achieved by modifying the attack objective defined by Eq. 3. For demotion attacks, we only need to change $\mathcal{L}_{atk}(X, \theta_R)$ into $-\mathcal{L}_{atk}(X, \theta_R)$. For availability attacks, we can define $\mathcal{L}_{atk}(X, \theta_R) = \sum_{r_{i,j} \in \Phi}(r_{i,j} - \hat{r}_{i,j})^2$, where Φ is the rating set, and $r_{i,j}$ represents the real rating of user i on item j. Alg. 1 does not require any changes.

4.5 Theoretical Analysis

This part will provide a systematic theoretical analysis of TrialAttack's learning ability under nonparametric assumptions, proving that it can approximate real users while performing an effective attack (see Appendix A.1 for all proofs).

First, we consider a simplified TrialAttack without attack loss, that is, $\beta_1 = 0$. Then the learning ability of the generator and influence module is described below.

Lemma 4.3. *when $\beta_1 = 0$, for a well-trained TrialAttack at the Nash equilibrium of $U(D, G, I)$, we can get $p(u, IF) = p_G(u, IF) = p_I(u, IF)$ with $D^*(u, IF) = 0.5$.*

The lemma tells us that if TrialAttack does not consider the attack, both the generator and the influence module can learn the real user distribution when the equilibrium is achieved. Moreover, $D^*(u, IF) = 0.5$ indicates that the discriminator is arduous to distinguish whether the user-influence pair is real or not.

Table 1: Statistics of datasets

Dataset	users	items	ratings	sparsity
ML-100K	943	1682	100000	93.70%
ML-1M	6040	3706	1000209	95.53%
FilmTrust	796	2011	30880	98.07%

Lemma 4.4. *For two distributions P and Q in the domain $\mathcal{X}$, suppose there is a small ϵ such that for any $x \in \mathcal{X}$, the probability densities $p(x)$ and $q(x)$ of P and Q satisfy $\|p(x) - q(x)\| < \epsilon$, then $JS(P|Q)|_{\epsilon=0} = 0$.*

Lemma 4.4 enlightens us that for any point x of distribution, if there always has a point with small ϵ-neighbors of x in another distribution, then the two distributions are almost identical same.

Theorem 4.5. *Assume that $p_{G,atk}(u, IF)$ is the distribution of the generated users after incorporating attack loss. When the equilibrium of $U(D, G, I)$ is achieved, TrialAttack can learn and approximate the distribution of real users, that is $p_{G,atk}(u, IF) \approx p(u, IF)$.*

From Theorem 4.5, we know that when considering the attack loss, the rating distribution learned by TrialAttack approaches the real one. Moreover, TrialAttack generates malicious users with specific attack intent by maximizing the influence. Thus, TrialAttack can produce poisoning profiles that are malicious but close to reality.

5 EXPERIMENTS

5.1 Experimental Setup

5.1.1 Dataset. We use three real-world datasets widely used in previous works [9, 13, 25, 30], including ML-100K[2] (MovieLens-100K), ML-1M[2] (MovieLens-1M), and FilmTrust[3], to evaluate TrialAttack. For FilmTrust, we filter the cold-start users (the number of ratings is less than 15) that are vulnerable [25]. Table 1 lists the specific statistics of these datasets. For each dataset, we randomly select a positive sample from each user for testing and use the rest as the training set and the validation set at a ratio of 9:1.

5.1.2 Parameter Settings. We consider the MF-based recommender system as the target model, and we set the latent factor dimension d to 64 unless otherwise specified. For the training of TrialAttack in Alg. 1, we set training epochs to 1000, all three modules use the Adam optimizer, the learning rates of generator and discriminator are both 0.0001, and the learning rate of the influence module is set to 0.001, $\alpha = 0.5$, $\beta_2 = 100$, $\gamma = 1$, $T_D = 1$, $T_G = 2$, $T_I = 1$. For ML-100K, ML-1M, and FilmTrust, β_1 is set to 2000, 4000, 400, N is set to 500, 2000, 500, respectively. Besides, we set the number of filler items m' in baselines to the average number of ratings per user and ensure that the average number of filler items sampled by diverse noise sampling is not larger than m'. The source code of TrialAttack is available in https://github.com/Daftstone/TrialAttack.

[2]https://grouplens.org/datasets/movielens/
[3]https://www.librec.net/datasets/flmtrust.zip

Table 2: Attack performance for full-knowledge attack. * indicates that the improvements compared with baselines are statistically significant for $p < 0.001$.**

Metric		HR@10													
		Unpopular Items							Random Items						
Dataset	Attack Size	None	Random	Average	AUSH	PGA	TNA	TrialAttack	None	Random	Average	AUSH	PGA	TNA	TrialAttack
ML-100K	1%	0.0000	0.0001	0.0060	0.0065	0.0008	0.0313	**0.3078*****	0.0072	0.0088	0.0264	0.0011	0.0085	0.0251	**0.0537*****
	2%	0.0000	0.0038	0.0512	0.0877	0.0120	0.3172	**0.8232*****	0.0072	0.0195	0.0491	0.0012	0.0147	0.0764	**0.2113*****
	3%	0.0000	0.0258	0.1898	0.3714	0.0387	0.4789	**0.9417*****	0.0072	0.0191	0.0626	0.0859	0.0205	0.1806	**0.2789*****
ML-1M	1%	0.0000	0.0045	0.1490	0.2500	0.0001	0.2587	**0.8629*****	0.0002	0.0022	0.0051	0.0023	0.0008	0.0056	**0.0266*****
	2%	0.0000	0.0167	0.2400	0.2687	0.0024	0.3863	**0.9258*****	0.0002	0.0038	0.0193	0.0077	0.0014	0.0244	**0.0701*****
	3%	0.0000	0.0313	0.2702	0.5377	0.0029	0.2656	**0.9455*****	0.0002	0.0061	0.0272	0.0121	0.0022	0.0632	**0.0845*****
FilmTrust	1%	0.0000	0.0003	0.0002	0.0000	0.0006	0.0052	**0.0118*****	0.0268	0.0385	0.0288	0.0341	0.0412	0.1040	**0.1578*****
	2%	0.0000	0.0024	0.0014	0.0000	0.0018	0.0471	**0.1377*****	0.0268	0.0273	0.0199	0.0351	0.0512	0.2002	**0.3889*****
	3%	0.0000	0.0037	0.0034	0.0065	0.0023	0.0559	**0.2752*****	0.0268	0.0299	0.0267	0.0440	0.0417	0.2357	**0.4708*****
Metric		NDCG@10													
		Unpopular Items							Random Items						
Dataset	Attack Size	None	Random	Average	AUSH	PGA	TNA	TrialAttack	None	Random	Average	AUSH	PGA	TNA	TrialAttack
ML-100K	1%	0.0000	0.0000	0.0022	0.0023	0.0003	0.0123	**0.1498*****	0.0027	0.0034	0.0110	0.0004	0.0032	0.0093	**0.0241*****
	2%	0.0000	0.0016	0.0224	0.0354	0.0043	0.1427	**0.4677*****	0.0027	0.0080	0.0225	0.0004	0.0056	0.0322	**0.1060*****
	3%	0.0000	0.0099	0.0797	0.1639	0.0145	0.2195	**0.5455*****	0.0027	0.0080	0.0256	0.0372	0.0077	0.0770	**0.1484*****
ML-1M	1%	0.0000	0.0020	0.0726	0.1161	0.0000	0.1296	**0.4848*****	0.0001	0.0011	0.0025	0.0010	0.0003	0.0024	**0.0147*****
	2%	0.0000	0.0078	0.1125	0.1239	0.0010	0.1903	**0.5287*****	0.0001	0.0018	0.0100	0.0035	0.0006	0.0118	**0.0412*****
	3%	0.0000	0.0142	0.1259	0.2632	0.0011	0.1229	**0.5435*****	0.0001	0.0031	0.0140	0.0056	0.0009	0.0275	**0.0489*****
FilmTrust	1%	0.0000	0.0001	0.0001	0.0000	0.0003	0.0019	**0.0047*****	0.0107	0.0138	0.0104	0.0131	0.0151	0.0426	**0.0702*****
	2%	0.0000	0.0010	0.0006	0.0000	0.0007	0.0212	**0.0553*****	0.0107	0.0107	0.0076	0.0133	0.0181	0.0915	**0.1770*****
	3%	0.0000	0.0016	0.0015	0.0025	0.0009	0.0225	**0.1249*****	0.0107	0.0120	0.0102	0.0160	0.0157	0.1079	**0.2324*****

5.1.3 Comparison Attacks. We compare TrialAttack with the classic shilling attacks and the state-of-the-art attacks based on model optimization, including Random Attack [21], Average Attack [21], PGA [22], TNA [12], and AUSH [25] (we put their details in Appendix A.3). Notably, baselines' performance under default settings is poor due to the use of different datasets, so we use grid search to obtain the optimal parameters under the validation set.

5.1.4 Evaluation Metric. We first use the average hit ratio (HR@k) and Normalized Discounted Cumulative Gain (NDCG@k) as evaluation metrics. HR@k measures the average fraction of the target items appearing in the user's top-k recommendation list. NDCG@k is adopted because the optimization of the attack objective $\mathcal{L}_{atk}(X, \theta_R)$ is to promote target items' ranking. For these two metrics, we truncate the ranked list k to 10.

Second, we study two types of target items: (1) Unpopular items, which are rarely accessed and are more likely to be targeted by attackers [25]. Here we randomly select unpopular items with the number of ratings less than 5. (2) Random items are randomly selected from all items. The number of target items is set to 5. Besides, we perform paired t-test for significance tests when necessary.

5.2 Full-knowledge Attack

5.2.1 Comparison with Baselines. We do 30 independent repeated experiments and summarize the average attack performance in Table 2. Firstly, the results reveal that TrialAttack is significantly better than the baselines with respect to HR@10 and NDCG@10. Compared with the state-of-the-art method TNA, TrialAttack has more than 33.7% improvements concerning HR@10 when attacking random items, and the improvement is more significant in the attack of unpopular items, with the improvements are up to 72.0% for HR@10. Secondly, for the AUSH, also based on GAN, we observe that its performance is almost the same as Average attack and other shilling attacks. That is because it incorporates in-segment loss, which is still a heuristic shilling attack in nature. Finally, we notice that in ML-1M and ML-100K, the attack damage on unpopular items is more obvious than that of random items. More than 80% of users are successfully attacked when the attack size is 2%. We suspect that they are denser and have relatively fewer behavior features than FilmTrust, making them more vulnerable to attack.

Table 3: The effect of Influence (Attack Size 3%)

Attack	ML-100K		ML-1M		FilmTrust	
	HR@10	NDCG@10	HR@10	NDCG@10	HR@10	NDCG@10
TrialAttack-NIF	0.0045	0.0016	0.0023	0.0010	0.3260	0.1629
TrialAttack	**0.2789**	**0.1484**	**0.0845**	**0.0489**	**0.4708**	**0.2324**

5.2.2 Effectiveness of Influence. The key of TrialAttack is to combine the influence function to guide the generator to produce influential users. In this part, we evaluate the impact of the influence function on the attack performance. Let TrialAttack-NIF be the attack without attack influence loss, namely $\beta_1 = 0$. Table 3 reports the comparison between TrialAttack-NIF and the original TrialAttack attacking random items when the attack size is 3%. It reveals that the advantage of TrialAttack using influence is remarkable. In the best case of ML-100K, HR@10 increases by about 62 times, while NDCG@10 increases by 92 times. These results emphasize the positive effect of influence on improving attack performance.

5.2.3 Performance in the defense model. We test the performance in the defense model. Here we use the most extensive adversarial training in recent studies[6, 16, 31, 35]. Table 4 lists the performance after adversarial training when the attack size is 3%. To emphasize the defense effect, we use HR@50 here. The result shows that even if the defense program is configured, TrialAttack is still effective and in the lead, revealing the powerful destructiveness of TrialAttack.

Table 4: Attack performance after defense (HR@50)

Target item	Dataset	Random	Average	AUSH	PGA	TNA	TrialAttack
Random	ML-100K	0.1505	0.3439	0.4126	0.2159	0.3896	**0.5174**
	ML-1M	0.0489	0.1937	0.1161	0.0405	**0.2051**	0.1686
	FilmTrust	0.1219	0.1290	0.2831	0.2679	0.7849	**0.8064**
Unpopular	ML-100K	0.3406	0.8464	0.8926	0.4812	0.9398	**0.9894**
	ML-1M	0.5581	0.9265	0.9571	0.3207	0.9281	**0.9791**
	FilmTrust	0.0091	0.0082	0.0596	0.6068	0.1709	**0.8304**

5.3 Partial-knowledge Attack

As introduced in Section 3.1, we build a local simulator to transform the partial-knowledge attack into a full-knowledge attack and execute TrialAttack and baselines on the local simulator.

For the sampling of observed users, we adopt the operation [12] that selects the nearest users based on the distance to target items. The local simulator's dimension is set to 128, which is different from the target model. Fig. 3 compares HR@10 against unpopular items when training with only partial users (we have similar results on random items). We notice that TrialAttack is still effective, and the lack of knowledge does not significantly reduce the attack performance. By contrast, we find that the more knowledge acquired, the better the attack is not necessarily (e.g., PGA in the FilmTrust achieves the best performance when 60% of user's ratings are observed). We suspect that it can focus on high-quality attacks on partial users under the premise of obtaining enough knowledge.

We also study the attack sensitivity of dimension d of the recommender system. We set the local simulator's dimension d to 64, while the target system uses different dimensions. The results under unpopular items are illustrated in Fig. 4. It shows that the dimension has minimal impact, and TrialAttack can still produce high-quality fake users in different dimensions. The excellent characteristic that the attack is invariably effective gives us a wake-up call that solving the security problem of recommender systems is imperative.

5.4 Fake User Detection

Following [28], we use the state-of-the-art peer-reviewed detection method based on fraudulent action propagation [38]. For ML-1M, 3% of fake users attacking different random items are generated, and for ML-100K and FilmTrust, which have a small number of users, generate 10% of fake users. To study the effectiveness of diverse noise sampling, we define our approach based on random sampling as TrialAttack-rand. Fig. 5 reports the F1 scores of detecting fake users under different parameters. The larger the score, the better the detection performance. Firstly, most of the users generated by TrialAttack-rand are easily detected, which indicates that diverse noise sampling is effective in learning real profiles. Secondly, it can be seen that TrialAttack is not easy to be perceived in these datasets. Especially on the ML-100K and ML-1M, the fake users almost entirely cheat the detector, which clearly verifies the superiority of TrialAttack in generating imperceptible users. Finally, we find that the detection performance of most attacks on FilmTrust has significantly decreased. We suspect that the FilmTrust is more sparse and easier to be injected fake users similar to normal users, which is also in line with the phenomenon observed in [13]. Please see Appendix A.5 for more experiments about the reality verification of the generated users.

6 CONCLUSION

In the paper, we propose an end-to-end framework, TrialAttack, for poisoning attacks in the recommender system. Through the triple adversarial learning of the generator, discriminator, and influence module, TrialAttack can efficiently produce non-notable and harmful users. Besides, the flexibility of TrialAttack allows us to design different attack influence functions and generate fake users with different attacking intents via the optimization of adversarial learning. Through theoretical analysis and extensive experiments under real-world datasets, we prove that TrialAttack can approximate real users while performing an efficient attack. Compared with the state-of-the-art poisoning approaches, the performance of TrialAttack is not inferior. Theoretically speaking, as long as the non-MF-based model is second-order differentiable after rating relaxation, our solution can work by replacing θ_R of Eq. 4 with the non-MF-based model's parameters. In the future, we plan to extend TrialAttack to other recommendation models.

ACKNOWLEDGMENTS

The work was supported by grants from the National Natural Science Foundation of China (No. 62022077 and 61976198), JD AI Research and the Fundamental Research Funds for the Central Universities (No. WK2150110017).

REFERENCES

[1] Naman Agarwal, Brian Bullins, and Elad Hazan. 2017. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research* 18, 1 (2017), 4148–4187.
[2] Mehmet Aktukmak, Yasin Yilmaz, and Ismail Uysal. 2019. Quick and accurate attack detection in recommender systems through user attributes. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 348–352.
[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875* (2017).
[4] Robin Burke, Bamshad Mobasher, Chad Williams, and Runa Bhaumik. 2006. Classification features for attack detection in collaborative recommender systems. In *Proceedings of KDD'06*. ACM, 542–547.
[5] Dong-Kyu Chae, Jin-Soo Kang, Sang-Wook Kim, and Jung-Tae Lee. 2018. Cfgan: A generic collaborative filtering framework based on generative adversarial networks. In *Proceedings of CIKM'18*. 137–146.
[6] 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.
[7] Zunping Cheng and Neil Hurley. 2009. Effective diverse and obfuscated attacks on model-based recommender systems. In *Proceedings of the Third ACM Conference on Recommender Systems*. 141–148.
[8] LI Chongxuan, Taufik Xu, Jun Zhu, and Bo Zhang. 2017. Triple generative adversarial nets. In *Proceedings of NIPS'17*. 4088–4098.
[9] Konstantina Christakopoulou and Arindam Banerjee. 2019. Adversarial attacks on an oblivious recommender. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 322–330.
[10] Yashar Deldjoo, Tommaso Di Noia, Eugenio Di Sciascio, and Felice Antonio Merra. 2020. How Dataset Characteristics Affect the Robustness of Collaborative Recommendation Models. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 951–960.
[11] Wenqi Fan, Tyler Derr, Xiangyu Zhao, Yao Ma, Hui Liu, Jianping Wang, Jiliang Tang, and Qing Li. 2020. Attacking Black-box Recommendations via Copying Cross-domain User Profiles. *arXiv preprint arXiv:2005.08147* (2020).
[12] 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.
[13] Minghong Fang, Guolei Yang, Neil Zhenqiang Gong, and Jia Liu. 2018. Poisoning attacks to graph-based recommender systems. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 381–392.
[14] Carlos A Gomez-Uribe and Neil Hunt. 2015. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)* 6, 4 (2015), 1–19.
[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial

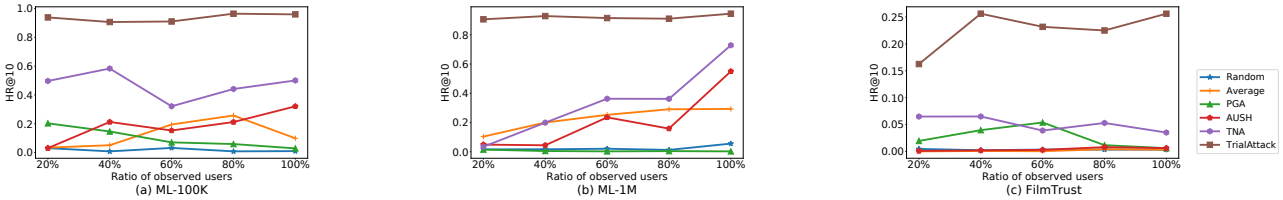


Figure 3: Comparison of attacks under obtaining partial user profiles (attack size 3%).

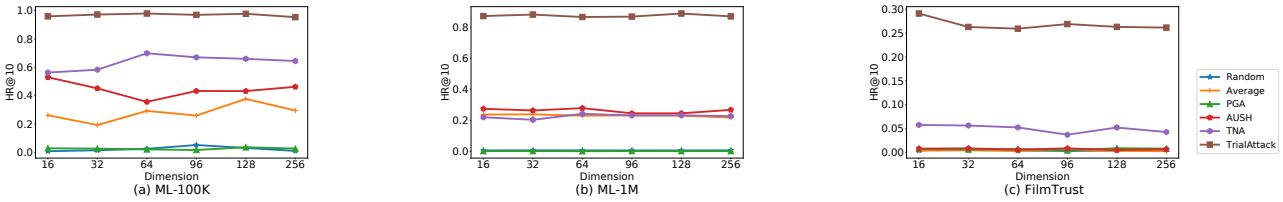


Figure 4: The impact of different dimension d of the recommender system on attack performance (attack size 3%).

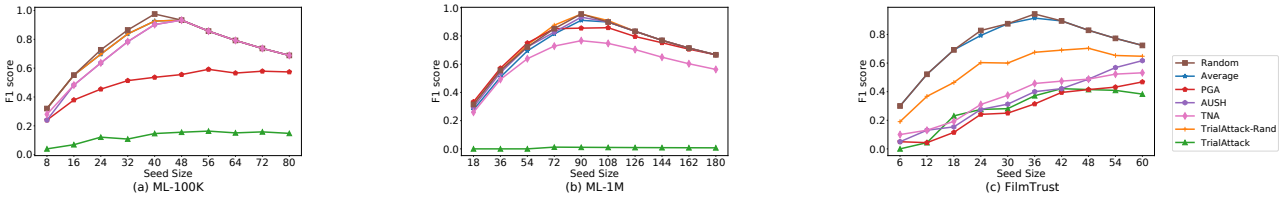


Figure 5: Comparison of F1 score for different attack detection.

nets. In *Advances in Neural Information Processing Systems*. 2672–2680.
[16] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial personalized ranking for recommendation. In *Proceedings of SIGIR'18*. 355–364.
[17] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. 2019. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*. 125–136.
[18] Binbin Jin, Defu Lian, Zheng Liu, Qi Liu, Jianhui Ma, Xing Xie, and Enhong Chen. 2020. Sampling-Decomposable Generative Adversarial Recommender. *Advances in Neural Information Processing Systems* 33 (2020).
[19] Pang Wei Koh and Percy Liang. 2017. Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning*. 1885–1894.
[20] Pang Wei W Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. 2019. On the accuracy of influence functions for measuring group effects. In *Advances in Neural Information Processing Systems*. 5254–5264.
[21] Shyong K Lam and John Riedl. 2004. Shilling recommender systems for fun and profit. In *Proceedings of WWW'04*. ACM, 393–402.
[22] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. In *Advances in Neural Information Processing Systems*. 1885–1893.
[23] Defu Lian, Yongji Wu, Yong Ge, Xing Xie, and Enhong Chen. 2020. Geography-aware sequential location recommendation. In *Proceedings of KDD'20*. ACM, 2009–2019.
[24] Defu Lian, Xing Xie, and Enhong Chen. 2019. Discrete matrix factorization and extension for fast item recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2019).
[25] Chen Lin, Si Chen, Hui Li, Yanghua Xiao, Lianyun Li, and Qian Yang. 2020. Attacking Recommender Systems with Augmented User Profiles. *arXiv preprint arXiv:2005.08164* (2020).
[26] Bhaskar Mehta. 2007. Unsupervised shilling detection for collaborative filtering. In *AAAI*. 1402–1407.
[27] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. 2007. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM TOIT* 7, 4 (2007), 23–es.
[28] Mingdan Si and Qingshan Li. 2020. Shilling attacks against collaborative recommender systems: a review. *Artificial Intelligence Review* 53, 1 (2020), 291–319.
[29] Brent Smith and Greg Linden. 2017. Two decades of recommender systems at Amazon. com. *Ieee internet computing* 21, 3 (2017), 12–18.
[30] 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.
[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] Jiaxi Tang, Hongyi Wen, and Ke Wang. 2020. Revisiting Adversarially Learned Injection Attacks Against Recommender Systems. In *Fourteenth ACM Conference on Recommender Systems*. 318–327.
[33] Jun Wang, Lantao Yu, Weinan Zhang, Yu Gong, Yinghui Xu, Benyou Wang, Peng Zhang, and Dell Zhang. 2017. Irgan: A minimax game for unifying generative and discriminative information retrieval models. In *SIGIR'17*. ACM, 515–524.
[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 SIGIR'19*. ACM, 1065–1068.
[36] 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.
[37] Sheng Zhang, Amit Chakrabarti, James Ford, and Fillia Makedon. 2006. Attack detection in time series for recommender systems. In *Proceedings of KDD'06*. ACM, 809–814.
[38] Yongfeng Zhang, Yunzhi Tan, Min Zhang, Yiqun Liu, Tat-Seng Chua, and Shaoping Ma. 2015. Catch the black sheep: unified framework for shilling attack detection based on fraudulent action propagation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
[39] Wei Zhou, Junhao Wen, Qiang Qu, Jun Zeng, and Tian Cheng. 2018. Shilling attack detection for recommender systems based on credibility of group users and rating time series. *PloS one* 13, 5 (2018), e0196533.
[40] Wei Zhou, Junhao Wen, Qingyu Xiong, Min Gao, and Jun Zeng. 2016. SVM-TIA a shilling attack detection method based on SVM and target item analysis in recommender systems. *Neurocomputing* 210 (2016), 197–205.
[41] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2847–2856.

A APPENDIX

A.1 Detailed Proofs

Proof of Proposition 4.1. Let $\mathcal{R} = E_{z' \sim p(z')} \|z - z'\|_0$, then

$$\mathcal{R} = \int p(z') \left\|z - z'\right\|_0 dz'.$$

It is difficult to derive the L_0 norm. Fortunately, for the discrete rating, we know that $\mathcal{R}$ gets the minimum when $z = M(z')$, where $M(z') \in \{0, ..., r_{max}\}^m$ denotes the mode vector of fake ratings. In a promotion attack, the target item t is given the maximum r_{max}, so $M(z'_t) = r_{max}$. In addition, TrialAttack is committed to learning the distribution of real users, then $M(z'_i) \approx \mu_i$ for non-target items. Therefore, the user z_{min} with sparse ratings is given by

$$z_{min} = \Pi(\mu + \mathbf{e_t}).$$

□

Proof of Proposition 4.2. For the influence function based on the first-order Taylor approximation [20], assuming $\delta_1 < \delta_2 < \epsilon$, when $\epsilon \to 0$, $Err_{pert(\delta_1)}(z') < Err_{pert(\delta_2)}(z')$. From Proposition 4.1 we know that $E(\|z' - z_{min}\|) < E(\|z' - z\|)$, let $\|z' - z_{min}\| = \delta_1$ and $\|z' - z\| = \delta_2$. Since z_{min} is directly poisoned to the recommender system, $Err_{add}(z', z_{min}) = 0$. For $z' \sim p(z')$, if $z' - z \to 0$, the following will hold

$$\begin{aligned}
E(Err_I(z', z_{min})) =& E(Err_{add}(z', z_{min}) + Err_{pert}(z', z_{min})) \\
=& E(Err_{pert}(z', z_{min})) \\
\leq& E(Err_{pert}(z', z)) \\
\leq& E(Err_{add}(z', z) + Err_{pert}(z', z)) \\
=& E(Err_I(z', z)).
\end{aligned}$$

□

Proof of Lemma 4.3. For convenience, we define $U'(D, G, I)$ as follows:

$$\begin{aligned}
U'(D, G, I) =& [E_{(u,IF) \sim p_D(u,IF)} \log D(u, IF) + \\
& \alpha E_{(u,IF) \sim p_I(u,IF)} \log(1 - D(u, IF))] + \\
& (1 - \alpha) E_{(u,IF) \sim p_G(u,IF)} \log(1 - D(u, IF)).
\end{aligned}$$

Similar to the proof of TripleGAN [8], we fix the discriminator and rewrite the minimax problem $U'(D, G, I)$ as follows:

$$\begin{aligned}
U'(D, G, I) =& \iint p(u, IF) \log D(u, IF) dIF du + \\
& \alpha \iint p(u) p_I(IF|u) \log(1 - D(u, IF)) dIF du + \\
& (1 - \alpha) \iint p_G(e) p(IF|G(e)) \log(1 - D(G(e), IF)) dIF de \\
=& \iint p_\alpha(u, IF) \log(1 - D(u, IF)) dIF du + \\
& \iint p(u, IF) \log D(u, IF) dIF du,
\end{aligned}$$

where $p_\alpha(u, IF) = (1 - \alpha) p_G(u, IF) + \alpha p_I(u, IF)$. Now fix the generator and influence module and take the derivative of $U'(D, G, I)$, we can find that when $U'(D, G, I)$ reaches the maximum value, $D^*(u, IF) = \frac{p(u,IF)}{p(u,IF) + p_\alpha(u,IF)}$.

Now bring $D^*(u, IF)$ into $U'(D, G, I)$ and rewrite the minimization problem as follows:

$$\begin{aligned}
\min_{G,I} U'(D, G, I) =& \min_{G,I} \max_{D} U'(D, G, I) \\
=& \min_{G,I} \iint p_\alpha(u, IF) \log(\frac{p_\alpha(u, IF)}{p(u, IF) + p_\alpha(u, IF)}) dIF du \\
& + \iint p(u, IF) \log \frac{p(u, IF)}{p(u, IF) + p_\alpha(u, IF)} dIF du \\
=& \min_{G,I} 2JS(p(u, IF) \| p_\alpha(u, IF)) - 4 \log 4,
\end{aligned}$$

where $JS(\cdot)$ is Jensen-Shannon divergence[3]. It takes the minimum value 0 when the two distributions are the same. Therefore, $U'(D, G, I)$ gets the optimal solution $-4 \log 4$ if and only if:

$$p(u, IF) = (1 - \alpha) p_G(u, IF) + \alpha p_I(u, IF).$$

Then we consider the supervised learning loss $\mathcal{L}^I_{sup}$ and reconstruction loss $\mathcal{L}^G_{rec}$. Minimizing $\mathcal{L}^G_{rec}$ is to make the generated distribution close to reality. By minimizing $\mathcal{L}^I_{sup}$, $p_I(u, IF) = p(u, IF)$ holds when the influence module reaches the optimum. They do not conflict with $U(D, G, I)$ reaching the optimum, that is, $p(u, IF) = p_\alpha(u, IF)$ is still established for $U(D, G, I)$.

Therefore, at the equilibrium of utility $U(D, G, I)$, $p(u, IF) = p_I(u, IF) = p_G(u, IF)$ with $D^*(u, IF) = \frac{p(u,IF)}{p(u,IF) + p(u,IF)} = 0.5$. □

Proof of Lemma 4.4. According to the definition of Jensen-Shannon divergence, we have

$$\begin{aligned}
JS(P \| Q) =& \frac{1}{2} \int p(x) \log \frac{p(x)}{(p(x) + q(x))/2} dx + \\
& \frac{1}{2} \int q(x) \log \frac{q(x)}{(p(x) + q(x))/2} dx.
\end{aligned}$$

Suppose there is another distribution K with probability density $k(x)$ such that $k(x) = (p(x) + q(x))/2$. Let $f(x) = p(x) - q(x)$, then $\|f(x)\| < \epsilon$, and then we could get $p(x) = k(x) + f(x)/2$ and $q(x) = k(x) - f(x)/2$. Now the $JS(P \| Q)$ can be rewritten as

$$\begin{aligned}
JS(P \| Q) =& \frac{1}{2} \int p(x) \log \frac{p(x)}{k(x)} dx + \frac{1}{2} \int q(x) \log \frac{q(x)}{k(x)} dx \\
=& \frac{1}{2} \int (k(x) + \frac{f(x)}{2}) \log \frac{p(x)}{k(x)} dx + \\
& \frac{1}{2} \int (k(x) - \frac{f(x)}{2}) \log \frac{q(x)}{k(x)} dx \\
=& \frac{1}{2} \int \frac{f(x)}{2} \log \frac{p(x)}{k(x)} dx - \frac{1}{2} \int k(x) \log \frac{k(x)}{p(x)} dx - \\
& \frac{1}{2} \int \frac{f(x)}{2} \log \frac{q(x)}{k(x)} dx - \frac{1}{2} \int k(x) \log \frac{k(x)}{q(x)} dx \\
=& \frac{1}{2} \int \frac{f(x)}{2} \log \frac{p(x)}{q(x)} dx - \frac{1}{2} KL(K \| P) - \frac{1}{2} KL(K \| Q) \\
\leq& \frac{1}{2} \int \frac{\epsilon}{2} \log(1 + \frac{\epsilon}{q(x)}) dx.
\end{aligned}$$

It is easy to know $\int \frac{\epsilon}{2} \log(1 + \frac{\epsilon}{q(x)}) dx|_{\epsilon=0} = 0$, so $JS(P|Q)|_{\epsilon=0} = 0$ is proved. □

Proof of Theorem 4.5. After adding the reconstruction loss $\mathcal{L}^G_{rec}$ in TrialAttack, the user produced by the generator is similar to the input, which is equivalent to minimizing $\|p_G(u) - p_G(e)\|$,

where $p_G(e)$ is the distribution of input noise. When the system reaches equilibrium under ideal training conditions, $p_G(u)$ and $p_G(e)$ are close enough, that is, $\|p_G(u) - p_G(e)\| < \epsilon_1$. Similarly, after adding the attack loss $\mathcal{L}^G_{atk}$, $\left\|p_{G,atk}(u) - p_G(e)\right\| < \epsilon_2$. Then

$$\begin{aligned}\left\|p_{G,atk}(u) - p_G(u)\right\| &= \left\|p_{G,atk}(u) - p_G(e) - (p_G(u) - p_G(e))\right\| \\ &\le \left\|p_{G,atk}(u) - p_G(e)\right\| + \|p_G(u) - p_G(e)\| \\ &= \epsilon_2 + \epsilon_1.\end{aligned}$$

Here ϵ_1 and ϵ_2 are small values tending to 0, so according to Lemma 4.4, the distribution of $p_{G,atk}(u)$ and $p_G(u)$ is close to the same, that is, $p_{G,atk}(u) \approx p_G(u)$. Furthermore, applying the Lemma 4.3, we know that when $U(D, G, I)$ at the equilibrium, $p_{G,atk}(u, IF) \approx p(u, IF)$. □

A.2 Architecture of TrialAttack

Table 5: Architecture of TrialAttack

Generator	Discriminator	Influence module
FC(512)	FC(512)	FC(512)
ReLU	LeakyReLU	LeakyReLU
FC(64)	FC(256)	FC(512)
ReLU	LeakyReLU	LeakyReLU
FC(512)	FC(64)	FC(64)
ReLU	LeakyReLU	LeakyReLU
FC(m)	FC(1)	FC(1)
tanh	sigmoid	

We describe the architecture of TrialAttack in Table 5.

A.3 Comparison Attacks

The specific details of the baseline method are as follows.

Random Attack [21]: The attacker assigns the maximum rating to the target items and randomly selects m' items as the filler items for rating. The rating score of each item is sampled according to the rating distribution of the dataset.

Average Attack [21]: It is similar to the Random attack. The only difference is that the rating score of the filler item i is randomly selected according to the rating distribution of item i.

PGA Attack [22]: The method constructs an objective function that increases the target item's prediction ratings and uses the projection gradient ascent to optimize the objective.

TNA Attack [12]: It first uses gradient descent to minimize the rate gap between the target item and top-k items. Then assign the maximum rating to the target items. Finally, select m' items with the largest value after optimization, and generate each filler rating according to the corresponding distribution. Note that we only use $\mathcal{U}$-TNA here. Because $\mathcal{S}$-TNA's performance is similar to $\mathcal{U}$-TNA, it takes an expensive time to search for influential users.

AUSH Attack [25]: This attack randomly samples ratings from the item repository and then uses the generative adversarial network with an in-segment loss to correct fake user ratings.

A.4 Running Time

In the real scene, the recommender system is updated in real-time, which requires attackers to instantly make attack decisions based on the system's current state. Table 6 shows the time required for attacks to generate 3% of fake users. It can be seen that model-independent shilling attacks such as Random attack can quickly generate fake users, while TrialAttack can achieve similar efficiency. Compared with TNA, the computing efficiency of TrialAttack has improved by a staggering 50000 times on ML-1M, which shows the powerful superiority of GAN in generation efficiency.

Table 6: Running time (s)

Dataset	Random	Average	AUSH	PGA	TNA	TrialAttack
ML-100K	0.12	0.14	1.17	448.63	2423.32	1.52
ML-1M	1.06	1.25	1.85	3038.50	129941.40	2.52
FilmTrust	0.03	0.04	1.01	401.86	1948.56	1.31

A.5 Additional Results for Reality Verification of Generated Users

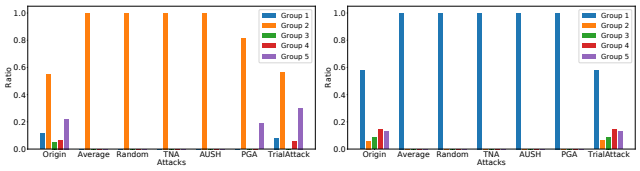
Table 7: Comparison of Wasserstein distance

Dataset	Random	Average	AUSH	PGA	TNA	TrialAttack
ML-100K	0.0509	0.0452	0.0417	0.0404	0.0463	**0.0198**
ML-1M	0.0338	0.0307	0.0311	0.0404	0.0313	**0.0125**
FilmTrust	0.0252	0.0249	0.022	0.0221	0.0225	**0.0092**

We use Wasserstein distance [3] to measure the similarity of rating distribution between real users and generated ones:

$$W(p, q) = \inf_{\gamma \sim \Pi(p,q)} E_{(x,y)\sim\gamma}[\|x - y\|],$$

where $\Pi(p, q)$ is the set of all feasible joint distributions that combine p and q. We use each attack method to generate 3% of fake users, and the comparison is reported in Table 7. The fake users generated by TrialAttack have the smallest Wasserstein distance in the three datasets, and the improvement has more than doubled. This gratifying result further confirms the detection's difficulty in our approach.



(a) ML-100K **(b) ML-1M**

Figure 6: Group distribution of users

Besides, the existing detection algorithms leverage the low diversity of fake users [7, 26]. We use K-means to cluster the real users and classify the fake users to study the diversity of generated users. Fig. 6 reports the distribution of fake users on ML-100K and ML-1M. We can reveal that the user ratings produced by the baselines are mostly concentrated in one group and lack diversity, while the group distribution of users in TrialAttack is similar to real users. This verifies from the perspective of user diversity why users generated by TrialAttack are challenging to detect.