LOKI: A Practical Data Poisoning Attack Framework against Next Item Recommendations

Hengtong Zhang, Yaliang Li, Bolin Ding, and Jing Gao

Abstract—Due to the openness of the online platform, recommendation systems are vulnerable to data poisoning attacks, where malicious samples are injected into the training set of the recommendation system to manipulate its recommendation results. Existing attack approaches are either based on heuristic rules or designed against specific recommendation approaches. The former suffers unsatisfactory performance, while the latter requires strong knowledge of the target system. In this paper, we propose a practical poisoning attack approach named LOKI against blackbox recommendation systems. The proposed LOKI utilizes the reinforcement learning algorithm to train the attack agent, which can be used to generate user behavior samples for data poisoning. In real-world recommendation systems, the cost of retraining recommendation models is high, and the interaction frequency between users and a recommendation system is restricted. Thus, we propose to let the agent interact with a recommender simulator instead of the target recommendation system and leverage the transferability of the generated adversarial samples to poison the target system. We also use the influence function to efficiently estimate the influence of injected samples on recommendation results, without re-training the models. Extensive experiments on multiple datasets against four representative recommendation models show that the proposed LOKI outperformances existing method. We also discuss the characteristics of vulnerable users/items, and evaluate whether anomaly detection methods can be used to mitigate the impact of data poisoning attacks.

Index Terms—Adversarial Learning, Recommendation System, Data Poisoning

1 Introduction

In the era of big data, one of the fundamental challenges for web users is information overload, because of which users struggle in locating the information they indeed need. Recommendation systems, which suggest items (e.g., movies, products, music, etc.) that are likely to interest users based on their historical behaviors, are proposed to alleviate the information overload issue. Nowadays, recommendation systems are widely deployed by Web service platforms (e.g., YouTube, Amazon, and Taobao) and play an important role in guiding users to make decisions and choices.

It is commonly assumed that online recommendation systems are honorable and unbiased. That is to say, they recommend users the items that match their personal interests. However, the openness of recommendation systems and the potential benefit of manipulating recommendation systems offer both opportunities and incentives for malicious parties to launch attacks. Particularly, through manipulating the reviews and the recommendation systems, the merchants can promote specific items to specific users to get high sales profits or demote specific items to downplay items from competitors. Recent studies [1], [2], [3], [4], [5] have demonstrated that recommendation systems are vulnerable to poisoning attacks. In these poisoning attacks, well-crafted data is injected into a recommendation system's training set by a group of malicious users. Such poisoning attacks make the system deliver recommendations as attackers desire.

Existing poisoning attacks can be categorized into two types. The first type of work is generally based on manually designed heuristic rules. For example, [2] design rules that leverage the following intuition: items that are usually selected together by users are treated as highly correlated by recommendation systems. To promote a target item to target users, attackers utilize controlled users to fake the co-occurrence between the target item and popular items. Nevertheless, such heuristic rules cannot cover various patterns of behavior in the recommendation data. Therefore, the performance of these attack methods is usually unsatisfactory. The other line of methods is designed for certain types of recommendation methods like matrix factorization-based models [1]. However, the architecture and the parameters of the recommendation systems in real-world platforms are generally unknown to the attackers. Usually, the only information that the attackers can rely on to infer the characteristics of the recommendation systems is the recommendation results of the users they controlled, and the frequency of these interactions is often limited. Thus, there is still a noticeable gap before these attack methods can be deployed in real practice.

In this work, we propose a novel practical adversarial attack framework against blackbox recommendation systems. We focus on one of the most common next-item recommendation settings, which aims to recommend top-K potentially preferred items for each user. The proposed

Part of this work was done when the first author was an intern at Alibaba U.S. Inc.

- *H. Zhang is with Tencent AI Lab, Shenzhen, Guangdong, China E-mail: drhzhang@tencent.com.*
- *Y. Li and B. Ding are with Alibaba U.S. Inc., 500 108th Ave NE, Suite 800, Bellevue, WA 98004. E-mail: {yaliang.li, bolin.ding}@alibaba-inc.com*
- *J. Gao is with School of Electrical and Computer Engineering at Purdue University, 610 Purdue Mall, West Lafayette, IN 47907. E-mail: jinggao@purdue.edu.*

reinforcement learning-based framework *LOKI* learns an attack agent to generate adversarial user behavior sequences for the data poisoning attack.

Unlike existing attack methods designed for certain types of recommendation methods, reinforcement learning algorithms utilize the feedback from the recommendation systems to learn the agent's policy. Thus, the framework does not need comprehensive knowledge of architecture and parameters. Nevertheless, in practice, the attacker cannot control when to retrain the target recommendation system, get feedback and update the attack strategy. In addition, recommendation system service providers generally restrict feedback frequency, but a reinforcement learning-based framework requires much feedback to train a policy function. Due to this discrepancy, we cannot directly rely on the feedback from the target recommendation system to train a policy within a tolerated time period. To tackle this challenge, we propose to construct a local recommender simulator to imitate the target model and let the reinforcement framework get reward feedback from the recommender simulator instead of the target recommendation system. The local recommender simulator is constructed by constructing an ensemble of multiple representative recommendation models. The intuition behind such a design is that if two recommenders can both get similar recommendation results on a given dataset, then the adversarial samples generated for one of the recommenders can be used to attack the other. Such transferability makes the recommender simulator a good substitute for the target recommendation system to guide the attack agent.

Moreover, even with the help of a local simulator, it is still time-consuming to retrain the recommendation systems within the simulator using the contaminated data for attack outcomes. To alleviate this problem, we design a component named outcome estimator based on the influence function. The outcome estimator can efficiently estimate the influence of the injected adversarial samples on the attack outcomes. These designs ensure that the proposed adversarial attack framework for recommendation systems is practical and effective.

In the experiments, we adopt four representative recommendation models as targets and conduct attacks on multiple real-world datasets to evaluate the proposed poisoning attack framework. Experimental results show that the proposed *LOKI* outperforms baseline attack methods. We also provide further analysis of the factors that influence the attack outcome and show what kind of users are vulnerable to the data poisoning attack proposed in this paper. Finally, we conduct a case study to investigate whether existing anomaly detectors can detect the generated adversarial samples. We demonstrate that the proposed poisoning attacks are still effective even when an anomaly detector is deployed.

In summary, the main contributions of this paper are as follows:

- We design a novel practical poisoning attack framework *LOKI* based on reinforcement learning to attack sophisticated blackbox next-item recommendation systems.
- In the framework, we propose to utilize a recommender simulator to reduce the number of feedback required from the target recommendation system, and incorporate an outcome estimator to guide the training of the attack agent efficiently.
- We conduct extensive experiments against four representative recommendation models on multiple real-world datasets to verify the advantages of the proposed framework *LOKI*.

The rest of the paper is organized as follows: In Section 2, we briefly introduce the background knowledge of the recommendation methods and detail the threat model. Then we describe the proposed framework *LOKI* in Section 3. In Section 4, we conduct a series of experiments and case studies on multiple real-world datasets. Section 5 is a survey of related work. We conclude the paper in Section 6.

2 Background & Threat Model

In this section, we first briefly review the general setting of the next-item recommendation task and a collection of representative approaches. After that, we describe the threat model, which specifies the attack goal, the attack approach, and the capability of the attacker.

2.1 Background: Next-Item Recommendation

To facilitate the discussions in the rest of this paper, we specify and formulate the next-item recommendation task as follows:

Let $\mathcal{U}$ be the set of users and $\mathcal{V}$ be the set of items, we use $\boldsymbol{x}_u = [x_u^1, x_u^2, \cdots, x_u^{m_u}]$ to denote a sequence of items that user u has chosen before in a chronological order in which $x_u^v \subseteq \mathcal{V}$. m_u denotes the number of items chosen by user u. Given existing sequences, the goal of the next-item recommendation is to output a K-sized ordered item list, which predicts the next item that the user will choose.

Generally, existing next-item recommendation methods can be categorized into two types: collaborative filtering methods and sequence-aware methods. Collaborative filtering methods are motivated by the idea that people often get the best recommendations from someone with tastes similar to themselves. The most common approaches of collaborative filtering are the neighborhood methods [6] and the factorization models [7], [8]. Recently, deep learning techniques, such as neural collaborative filtering [9] and auto-encoders [10], have been successfully applied in recommendation tasks yielding promising results. Sequence-aware recommendation is an emerging topic in the recommendation domain. This line of methods uses entire sessions as basic data units for analysis and recommendation. The major difference between collaborative filtering and sequence-aware recommendation is that the latter can utilize transactional structure information within user behavior sequences to model user preference shifts. Modern session-based recommendation models are driven by the development of machine learning techniques, especially sequence modeling approaches, such as Markov chain based models [11], recurrent neural network (RNN) based models [12], [13], [14], and attention-based models [15], [16].

2.2 Threat Model

We have briefly described the representative methods for the next-item recommendation task and discussed their major differences. Now let us detail the threat model of the attack against the next-item recommendation.

Attack Goal: An attacker's goal is to promote a set of target items to as many target users as possible. Specifically, suppose the system recommends K items to each user, *the attacker's goal is to maximize averaged display rate, which denotes the fraction of target users whose top-K recommendations results include the target items.*

Note that an attacker could also demote a target item. Demotion can be viewed as a special case of promotion as an attacker can promote other items such that the target item is demoted in recommendation lists. Thus, in this paper, we focus on promotion attacks.

Attack Approach: To achieve the attack goal, we consider the most general scenario in which the attackers can inject controlled users into the recommendation system. These controlled users visit or rate to well-selected items, which are named as *proxy items*, step-by-step. Thus, the well-crafted activities of each controlled user form a *behavior sequence*. To make the injection unnoticeable, the number of visits or ratings each controlled user conducts is limited to at most M.

The Knowledge and Capability of the Attacker: In this paper, we assume that the attacker is granted the following knowledge and capability.

1) The attacker can access the full activity history of all the users in the recommendation system. Such information is publicly available and can be easily collected by the attacker.
2) The attacker has limited resources so the attacker can merely inject a limited number of controlled users into the system. These controlled users can easily be bought from the underground market[1].
3) The attacker does not know the details about the target recommendation system, e.g., the parameters and the architecture of the recommendation model. Such setting is also known as *blackbox setting*.
4) The attacker can only receive little feedback (e.g., display rates) from the blackbox recommendation model to update and improve the proposed attack approach under blackbox setting. Such feedback can be obtained from the recommendation results of the controlled users.
5) The attacker does NOT know when the target blackbox recommendation model is retrained.

For real-world, all these knowledge and capability are available to any potential attackers. In the rest of this paper, we propose a general reinforcement learning based framework to leverage the knowledge and capability mentioned above to craft poisoning samples for the attack.

3 Methodology

In this section, we first provide an overview of the proposed reinforcement learning based framework. Then we describe the detailed design of each component of the framework. Finally, we summarize the framework and discuss how the designed components make the proposed poisoning attack feasible and practical.

1. https://www.buzzfeednews.com/article/leticiamiranda/amazon-marketplace-sellers-black-hat-scams-search-rankings

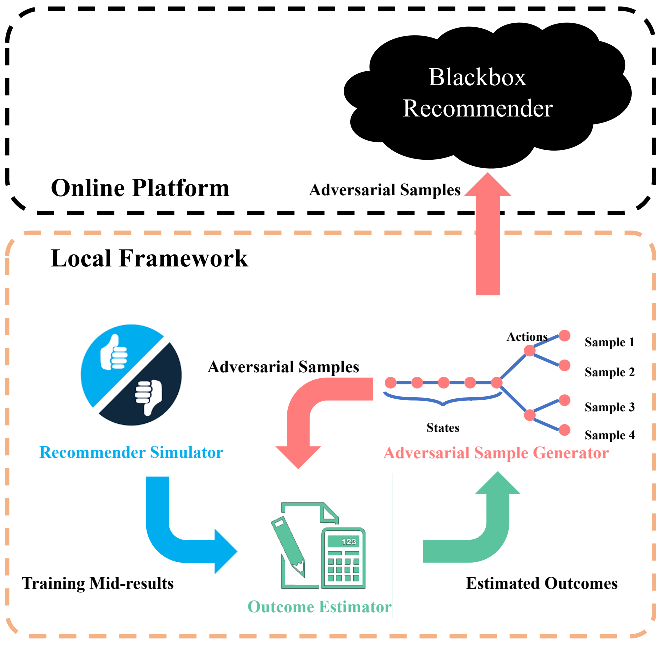


Fig. 1: Overview of the proposed framework *LOKI*.

3.1 Model Framework

Intuitively, data poisoning can be regarded as the creation of new sequential patterns that involve the target items in the training set of the target recommendation system. In a crafted sequential adversarial sample, the user behavior history is inherently crucial in determining the next behavior. These sequential adversarial samples together contribute to the manipulation goal. Generating adversarial samples is essentially a multi-step decision process, in which the generator ought to select specific actions for the controlled users to maximize attack outcomes. This fits the reinforcement learning setting. From the perspective of reinforcement learning, the goal is to learn a policy function to generate sequential adversarial user behavior samples, which can maximize the averaged display rate of the target users.

Based on the aforementioned motivation, we propose a reinforcement learning based framework to learn the policy function. The overall architecture of the proposed framework *LOKI* is illustrated in Figure 1. The target blackbox recommendation system is deployed on an e-commerce platform. The proposed framework consists of three components: (1) recommender simulator, (2) outcome estimator, and (3) adversarial sample generator. These components and the target recommendation system interact with each other in the following way:

1) **Recommender Simulator**: Since the frequency of querying the target recommender is restricted, we deploy a local simulator, which is an ensemble model consists of multiple recommendation models

with different weights, to simulate the recommendation preference of the target model. Intuitively, if two recommenders can both produce similar results on a dataset, the adversarial sample designed for one model can be transferred to another model for outcome manipulation. Similar transferability is studied and used in the general adversarial domain [17], [18]. To determine the weights of individual generators, we utilize the samples generated by the adversarial sample generator (will be mentioned below) to query the target recommender. The querying results are used to evaluate the difference between recommendation results produced by individual recommendation models and the remote model. Therefore, we can adjust the local simulator to better approximate the target recommender.

2) **Outcome Estimator**: When utilizing the feedback from the local recommender simulator, a practical issue is that the time for retraining the recommendation model is extremely long. Thus, we introduce outcome estimator, which is used to estimate the influence of the injected poison samples without retraining the models within the simulator. The key of this estimator is an influence function that can derive effective estimates of such an influence. Since all the individual recommendation models are trained on the same training set and the attacker can access the detailed intermediate results like gradients during the optimization. Such information is used in the outcome estimator.
3) **Adversarial Sample Generator**: The adversarial sample generator generates adversarial samples to poison the training set of the recommendation system. The adversarial sample generator interacts with the local recommender simulator to get the reward, i.e., attack outcome, and update the policy network. The outputs of the outcome estimator work as rewards for parameter updates.

In the following sections, we describe the details of these components one-by-one.

3.2 Recommender Simulator

The recommender simulator simulates the recommendation preference of the target model. It consists of multiple separated recommendation models, which are trained on the same dataset. The recommendation models used to build the simulator are detailed in Section 4. Recommendation results from these models are aggregated via weighted voting. Thus, the most crucial problem in building a recommender simulator comes to determining the weights of different recommendation models.

Concretely, suppose M different recommendation models are deployed to recommend items for user u. We use $rank_m(u, i)$ to denote the rank of the i-th item under the m-th model, and $rank_m(u)$ to denote the ranking result of the m-th model, which is an ordered list. *The higher item i ranks, the smaller $rank_m(u, i)$ is*. Since the remote recommender is considered to be the aggregation results of multiple recommendation models, we denote the weight of the m-th recommendation model as w_m. Ideally, these weights are used to adjust the simulator to match the characteristics of the target recommender.

Let the rank generated by the target recommender for user u be $rank_*(u)$, we cast the aggregation mechanism of recommender simulator as the following optimization problem:

$$\min_{\boldsymbol{w}} \sum_{m=1}^{M} w_m d(rank_*(u),\ rank_m(u)), \tag{1}$$

where $d(\cdot,\cdot)$ denotes a listwise distance function. In this paper, we use Kendall's tau distance [19] to implement d. We may trunk the ranks produced by recommendation models to top-K in practice.

The intuitions behind Eq. (1) are straightforward, i.e., the aggregated rank should be as close to the recommendation results from models with high weights as possible.

Since the aggregated rank and the individual ranks can be obtained via querying the target and the local recommendation models, we merely need to solve for the model weights $\boldsymbol{w}$. Specifically, After getting adversarial samples from the generator, we utilize them to determine model weights w_m in the recommender simulator. With these samples, we can solve for $\boldsymbol{w}$ by simply taking the derivative of Eq. (1).

The design of a recommender simulator leads to a significant reduction in the number of interactions required to train the attack agent. This is because the framework merely uses the feedback to derive the weights of individual models within the simulator instead of training the attack agent directly. For most of the experiments in this paper, we use 1% of the train data to query the remote target recommender and utilize the obtained ground truth recommendation results to adjust the weights of different recommendation models in the local recommendation simulator. Empirically, the more samples we use to adjust the local simulator, the better it can mimic the recommendation results of the target recommender and work as a good 'rivalry' to train the attack agent.

3.3 Outcome Estimator

As mentioned in Section 3.1, we need to utilize the manipulation outcome of the current adversarial samples as reward feedback to update the policy network of the adversarial sample generator. The most straightforward way to obtain the outcome is to retrain the entire model. However, retraining the online recommendation system is prohibitively slow (from a few hours to days for a single retraining). To make the attack methodology practical, we propose to use influence function for an efficient estimation of the manipulation outcome, motivated by robust statistics.

Before diving into the detailed approach, let us review the manipulation process from the perspective of optimization. Formally speaking, the parameter estimator of the recommendation models on the uncontaminated dataset is:

$$\hat{\theta} := \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(z_i; \theta), \tag{2}$$

where θ denotes the parameter vector, $\mathcal{L}$ stands for the loss function of the recommendation model. z_i denotes a sample in the dataset, and N stands for the total number of samples in the training set. For collaborative filtering models, a sample is a single user-item pair (u, v). For session-based recommendation models, given the behavior sequence $\boldsymbol{x}_u = [x_u^1, x_u^2, \cdots, x_u^m]$ of a user u, each training sample is made up of a subsequence and the ground truth next item, i.e., $([x_u^1], x_u^2)$, $([x_u^1, x_u^2], x_u^3), \cdots, ([x_u^1, x_u^2, \cdots, x_u^{n-1}], x_u^n)$.

Now let us move on to the discussion of the influence function. Suppose we upweight a sample z_δ by a small ϵ in the training set, the new estimation of θ is given as:

$$\hat{\theta}_{z_\delta} := \arg\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(z_i;\theta) + \epsilon\mathcal{L}(z_\delta;\theta). \tag{3}$$

When $\epsilon \rightarrow 0$, according to the classic results in [20], the influence of upweighting z_δ on the parameter θ is given by:

$$\hat{\theta}_{z_\delta} - \hat{\theta} \approx -H_{\hat{\theta}}^{-1}\nabla_\theta\mathcal{L}(z_\delta;\hat{\theta}), \tag{4}$$

where $H_{\hat{\theta}} := \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta^2 L(z_i,\theta)$, denotes the Hessian matrix of the loss function. Given the fact that the number of users is large in the recommendation datasets, injecting a data sample is the same as upweighting the sample by $\epsilon \approx \frac{1}{N}$.

Here, the key computation bottleneck lies in the calculation of the huge inverse Hessian matrix H_θ^{-1}. Given a sample z_j, we use implicit Hessian-vector products (HVPs) [21], [22] to efficiently approximate $H_\theta^{-1}\nabla_\theta L(z_j,\hat{\theta})$. The adapted approximation algorithm (LiSSA) [22] is shown in Algorithm 1. We choose T to be large enough such that the HVP stabilizes. Here, the second order derivatives, i.e., $\nabla_\theta^2 L(\boldsymbol{x}_i,\theta)$ in Algorithm 1, can be cached to further improve the efficiency. For a detailed derivation of the approximation of $\nabla_\theta^2 L(\boldsymbol{x}_i,\theta)$, please refer to [22].

```
Input: Number of samples to estimate the Hessian,
       i.e., $T$. Iterations needed to approximate the
       inverse of Hessian, i.e., $S$. Gradient of the loss
       function with perturbations: $\nabla_\theta L(\boldsymbol{z}_j,\theta)$.
Output: Unbiased estimation of $H_\theta^{-1}\nabla_\theta L(\boldsymbol{z}_j,\theta)$.
Initialize $H_{\theta,(0)}^{-1}\nabla_\theta L(\boldsymbol{z}_j,\theta) = \nabla_\theta L(\boldsymbol{z}_j,\theta)$ ;
for $t = 1, \cdots, T$ do
    Draw a sample $\boldsymbol{z}_i$ from the training set;
    Update: $H_{\theta,(t)}^{-1}\nabla_\theta L(\boldsymbol{z}_j,\theta) \leftarrow \nabla_\theta L(\boldsymbol{z}_j,\theta) + \big(I -$
      $\nabla_\theta^2 L(\boldsymbol{z}_t,\theta)\big)H_{\theta,(t-1)}^{-1}\nabla_\theta L(\boldsymbol{z}_j,\theta)$;
end
return $H_{\theta,(T)}^{-1}\nabla_\theta L(\boldsymbol{z}_j,\theta)$.
```

Algorithm 1: Estimation of $H_\theta^{-1}\nabla_\theta L(\boldsymbol{z}_j,\theta)$.

Based on an approximate estimate of the sample upweight's influence on parameter $\hat{\theta}$, we further calculate the influence on the prediction scoring function w.r.t. the perturbation. Specifically, suppose we want to promote an product v' to user u', we can treat this as a *target sample* $z_{u'v'}^{test}$in the test set. The influence on the prediction scoring function w.r.t. can be written as:

$$\begin{aligned}\frac{df_{test}(z_{u'v'}^{test};\hat{\theta})}{d\epsilon} &= \frac{df_{test}(z_{u'v'}^{test};\hat{\theta})}{d\hat{\theta}_{z_\delta}} \cdot \frac{d\hat{\theta}_{z_\delta}}{d\epsilon} \\ &\approx -\nabla_\theta f_{test}(z_{u'v'}^{test};\hat{\theta})H_{\hat{\theta}}^{-1}\nabla_\theta\mathcal{L}(z_\delta;\hat{\theta}),\end{aligned} \tag{5}$$

where f_{test} is the prediction scoring function used by the recommender system in the test phase. This result is further used to design rewards for efficient agent policy training.

3.4 Adversarial Sample Generator

The adversarial attack against a local recommender simulator is essentially interpreted as a multi-step decision problem. In this section, we translate this decision problem into a Markov Decision Process (MDP), which is a general mathematical framework for modeling decision making. The MDP can be solved via reinforcement learning. We will first show the design of the MDP, in which we utilize *coarse action space* to improve the efficiency of agent training. Then we specify the architecture of the Q-network, which is needed to solve the MDP. Finally, we briefly discuss the training details and strategies.

3.4.1 Design of the MDP

MDP is defined as a tuple $(\mathcal{S},\mathcal{A},\mathcal{P},\mathcal{R},\gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P}$ is the transition probabilities, $\mathcal{R}$ is the immediate reward, and γ is the discount factor. In the context of this paper, the MDP can be specified as follows:

Action space $\mathcal{A}$: As mentioned in Section 3.1, the attacker determines specific items organized in a proper sequence for each controlled user. Instead of taking the set of all the possible items as action space, we divide the item set into groups and use the set of all the groups as action space. The main reason for using coarse groups instead of items for action space is due to the concern in learning efficiency. Learning action strategies for every single item is not only costly but also unnecessary for the attack goal. This is because adversarial samples do not need to follow the exact sample pattern. Here we define one of the groups as the collection of all the target items. The remaining groups are obtained by item clustering, in which each group represents items with similar properties. This item clustering takes the feature vectors of all the items and an integer c as input and divides the items into c clusters. Here, we utilize non-negative matrix factorization [23] to extract item features and use K-means [24] algorithm for clustering.

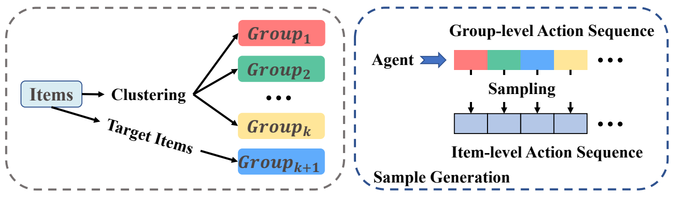


Fig. 2: Generation of adversarial samples.

After item groups are obtained, during the phases of training and testing the agent, *group-level actions* are sampled step-by-step from the policy, forming a *group-level action sequence*. Then sequential poisoning samples are sampled step-by-step from the corresponding group indicated by the current step of the *group-level action sequence*. This process is illustrated in Figure 2. The left side of Figure 2 shows the

process of clustering items into groups and the right side illustrates the process of generating the poisoning samples.
State $\mathcal{S}$: The state is defined as the action subsequence before current step t and the actions all come from the action space mentioned above.

Reward $\mathcal{R}$: As aforementioned, the purpose of the attacker is to manipulate the local recommender simulator and further the target recommender. Thus, we define the reward as the weighted aggregation of the prediction scoring function of different recommendation models in the recommender simulator. Formally, the reward of step t is defined as:

$$r(s_t, a_t) = \frac{1}{M}\sum_{m=1}^{M} w_m \cdot \nabla_\theta f_{test}(z^{test}_{u'v'};\hat{\theta}) H^{-1}_{\hat{\theta}} \nabla_\theta \mathcal{L}(z_\delta;\hat{\theta}), \tag{6}$$

where f_{test} is the prediction scoring function used by the recommender system in the test phase. The second term in the reward definition is identical to Eq. (5)

The objective of an agent in an MDP is to find the optimal policy that maximizes the expected accumulative rewards from any initial state s_0. At each step $t = 0, 1, \cdots, T$, the MDP is in some state s_t, and the agent may choose any action a_t that is available in state s_t according to the policy. Then the MDP moves into a new state s' and gives the agent a corresponding reward. In this paper, we apply deep Q-learning to learn the MDPs. Q-learning is an off-policy optimization method which fits the Bellman optimality equation of the Q action-value function directly as below:

$$Q^*(s_t, a_t) = r(s_t, a_t) + \gamma Q^*_{a'}(s_{t+1}, a'), \tag{7}$$

which implicitly suggests a greedy policy:

$$\pi(a_t|s_t) = \arg\max_{a_t} Q^*(s_t, a_t). \tag{8}$$

In this paper, we apply Deep Q-Network (DQN) to estimate the action-value function.

3.4.2 Architecture of the Q-Network

The proposed Q-network structure is shown in Figure 3. The representation of the existing sequence, i.e., state, is modeled via a GRU (Gated Recurrent Unit) layer, and the representation of each type of *actions* is extracted via a DNN (Deep Neural Network) layer. Here, we utilize the Dueling architectures [25], which divides the Q-function into value function $V(s)$ and advantage function $A(s,a)$, where $V(s)$ is only determined by the state representations, and $A(s,a)$ is determined by both the state and the action representations.

3.4.3 Training Process

In each iteration of a training session, there are two stages, replay memory generation stage and parameters update stage. In replay memory generation stage, the agent generates a group-level action a_t according to an ϵ-greedy policy and current state s_t. Then the item-level sequences are generated by sampling items from the corresponding group suggested by each step in the group-level sequence. After that, the agent observes the reward r_t from the outcome estimator and updates the state.

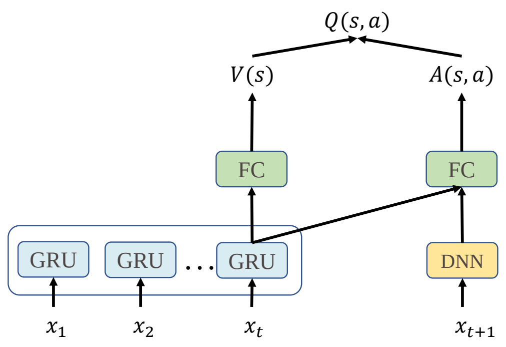


Fig. 3: Architecture of the Q-Network. GRU denotes Gated Recurrent Unit; DNN denotes Deep Neural Network; and FC stands for Fully-Connected layer.

For parameter update stage: the agent samples a (s_t, a_t, r_t, s_{t+1}) from replay memory, and then updates the parameters by minimizing a loss defined in the following equation:

$$\mathcal{L}_\theta = \mathbb{E}_{s,a,r,s'}\left[\left(r + \gamma Q^{target}(s,a;\theta) - Q^{policy}(s,a;\theta)\right)^2\right]. \tag{9}$$

Here, $Q^{target}(s,a;\theta)$ is only used to calculate the target value and $Q^{policy}(s,a;\theta)$ is used to generate new actions. These two networks have the same architecture but different parameters. Every T steps, $Q^{target}(s,a;\theta)$ synchronizes its parameters with $Q^{policy}(s,a;\theta)$. This strategy can help stabilize the learning procedure and avoid the divergence of parameters.

Finally, it is worth mentioning that we enforce a balanced sampling strategy during the Q-network training stage, as it is difficult for the training algorithm to obtain positive samples during the early training stage. Maintaining a balanced sample can make full use of these valuable positive samples and avoid model collapse.

3.5 Summary

As mentioned in Section 1, the major challenge of attacking a blackbox recommender system is two-fold. First, the architecture and the parameters are unknown. Second, the frequency of interactions between the attacker and the target system is restricted. The framework presented in this section is designed to tackle these challenges. With the help of the proposed simulator, the framework does rely heavily on direct feedback from the target model to train the attack agent. Instead, the feedback is merely used to approximate the local recommender towards the target recommender so that the adversarial samples generated for the simulator can be transferred to attack the target model. Thus, the number of interactions required between the attack agent and the target recommendation systems is significantly decreased. Besides, the time cost for re-training the local simulator to get the feedback is extremely high. To further decrease the time cost for the local training of the attack agent, we utilize the influence function to estimate attack outcomes and utilize coarse group-level action space to shrink the exploration space of the agent.

TABLE 1: Data statistics.

Dataset	Amazon	Steam	Gowalla
# Users	22,363	25,038	13,591
# Items	12,101	11,474	14,322
# Activities	176,139	709,022	227,193
Avg. Activities / User	7.88	28.31	36.71
Avg. Activities / Item	14.56	61.79	35.86

4 Experiments

In this section, we conduct extensive experiments on three real-world datasets to verify the attack effectiveness of *LOKI* when facing four victim recommendation methods. The experimental results clearly demonstrate that *LOKI* consistently outperforms the existing attack approaches given practical attack budgets. We also design various experiments to answer practical questions like (1) How efficient is the outcome estimator? (2) Whether the attack is still effective when the target recommendation system is equipped with an anomaly detector? and (3) How many queries do we need to adjust the simulator.

4.1 Datasets

To demonstrate the effectiveness of the proposed poisoning attack framework, we adopt the following real-world recommendation datasets.

- **Amazon**: This dataset is one category of the widely used recommendation dataset series named *Amazon* [26]. The dataset used in this paper mainly focuses on skin care and beauty.
- **Steam Dataset**: This is the benchmark dataset collected by [16]. The dataset is crawled from a large online video game store named Steam[2] from Oct. 2010 to Jan. 2018.
- **Gowalla Dataset**: This is the dataset from [27]. Gowalla is a location-based social networking website where users share their locations by checking-in. Unlike the *Amazon* and the *Steam* dataset mentioned above, the task on this dataset is to recommend the next location that a user may check in.

For all the datasets, we followed the same preprocessing procedure introduced in [16], [28]. We treat the presence of an action (e.g., click or rating) as an implicit feedback and utilize the appended timestamp to form the behavior sequence of each user. For each user u in the dataset, suppose the length of u's sequence is T_u, we hold the first $T_u - 2$ actions in the sequence as the training set and use the next one action as the validation set to search for the optimal hyperparameter settings for these recommendation models. The attack methods aim to manipulate the prediction of the next item, i.e. item T_u. Moreover, we filter out the users with less than five actions and items with less than five feedbacks. The statistics of these datasets are shown in Table 1.

4.2 Experimental Settings

4.2.1 Baseline Attack Methods

As aforementioned, there is no existing work solving exactly the same task considered in this paper. Thus, we compare LOKI with multiple representative heuristic-based attack strategies and optimization-based strategies:

- **None**: This denotes the circumstance when no attack is conducted.
- **Random**: In this baseline method, the attacker mixes the target items and the randomly picked items to form a repository for each controlled account. In each step, the controlled user picks items at random from the item repository without repetition.
- **Popular**: In this baseline method, attackers inject fake co-visitations between the popular items and the target items, to promote the target items.
- **Co-visit**: This is a variant of [3]. The attackers inject fake co-visitations between the items already chosen by the target users and the target items, to promote the target items.

4.2.2 Target Recommendation Methods

In this section, we consider the following target methods for performance comparisons.

- **BPRMF** [8] is a factorization based personalized ranking approach. It is a state-of-the-art method for non-sequential item recommendation on implicit feedback data.
- **FPMC** [11] is a classic hybrid model combing Markov chain and matrix factorization for next-basket recommendation. FPMC can model the user's long-term preference and the short-term item-to-item transition.
- **GRU4REC** [12] is a representative session-based recommendation model, which uses GRU to capture sequential dependencies and make predictions.
- **TransRec** [29] is one of the highly cited method, which models the high-order relationships for recommendation. The method embeds items into a 'transition space' where users are modeled as translation vectors operating on item sequences.

For all the recommendation methods mentioned above, we use the implementation from the NeuRec[3] library. The parameters of these target methods are set following the suggestion in the original papers.

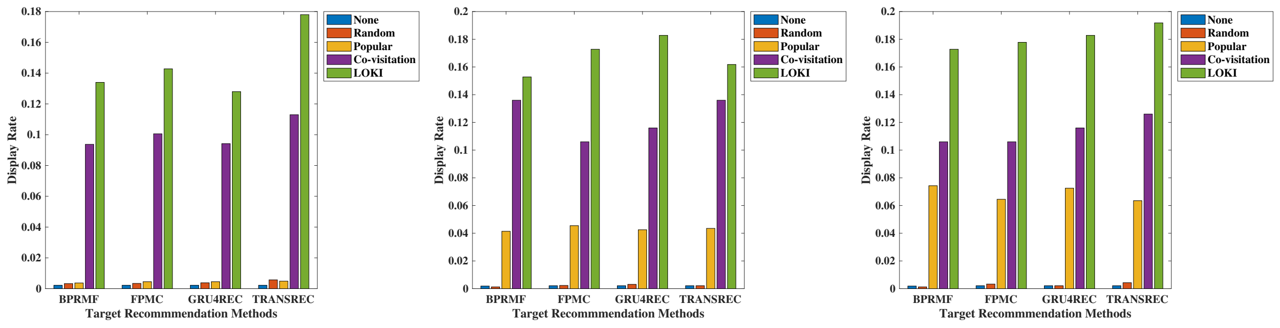
To simulate the interactions between the target blackbox recommendation system and the recommender simulator, we adopt the "leave-one-out strategy". Specifically, we design four scenarios. In each scenario, one of the four recommendation methods is used as the target blackbox recommendation system, and the remaining ones together form a local recommender simulator. For instance, we use BPRMF as the target recommendation system that is blind to the attacker, and use the other methods to form a recommender simulator. The number of target items and target users in this paper are both fixed to be 20.

4.2.3 Evaluation Metric

We use the averaged *display rate*, which denotes the fraction of target users whose top-K recommendation results include the target items, as our evaluation metric. The larger the display rate is, the better the attack approach performs.

2. https://store.steampowered.com/

3. https://github.com/wubinzzu/NeuRec



(a) Amazon (b) Steam (c) Gowalla

Fig. 4: Overall performance of all the attack methods on real-world datasets (Better view with color).

4.3 Result Analysis

Fig 4 summarizes the results for all the attack methods in four scenarios on the two real-world datasets. Here, we fix the percentage of controlled users to 3% and the number of actions per user to 15. The number of returned items is fixed to 40. In terms of attack outcome, the proposed *LOKI* achieves the best performance on every dataset, and the improvement is significant. For example, on the Amazon dataset, compared with the best baseline Co-visitation, the proposed *LOKI*'s display rate increases by almost 50% on average. On the Steam dataset, the proposed *LOKI* increases the display rate by 27% on average. Among the baseline methods, *Random* simply lets the target items occur in the poisoning sequences without actually creating any new pattern that favors the recommendation of the target items. Thus, *Random* has the worst performance. *Popular* fakes the co-visitations between the popular items and the target items without considering whether these popular items indeed overlap with the preferences of the target users. Thus, they have the second-worst performance. *Co-visitation* considers the preference of the target user to fake the co-visitations that involve the target items, and thus gets better performance than the other two baselines.

Compared with these baselines, the proposed *LOKI* takes advantage of the feedback from the local simulator to train an attack agent. The learned agent is capable of creating more complex patterns to achieve data poisoning goals. We also notice that the more complicated the target model is, the higher increase in performance metric the proposed *LOKI* achieves. For instance, the performance gap is larger when attacking GRU4REC or TransRec than attacking BPRMF. This is because GRU4REC or TransRec is able to capture more complicated user patterns within user behavior sequences. The capability to capture various user patterns leads to better prediction performance in general, but at the same time, enables more room for the attack improvement by the proposed *LOKI* with its ability of creating new patterns to poison the recommendation. That is to say, in some circumstances, the proposed *LOKI* can create certain sequential patterns. However, since relatively simple methods cannot capture these crafted patterns, these methods are less sensitive to the adversarial samples generated by the proposed *LOKI*.

With the above experimental results confirming the effectiveness of the proposed *LOKI* framework, we dig further to reveal the answer to an interesting question: *What kind of users are more vulnerable to the proposed attack?*

To answer this question, we extract several critical characteristics of the target users and compare the ones that are successfully manipulated with the rest. For each user, we calculate the following statistics as the characteristics:

- *Item per User*: The number of behavior carried out by the user;
- *Item Popularity*: The averaged occurrence number in the entire dataset of the items chosen by the user;
- *Targets Chosen*: The number of target items that have been chosen by the user before the recommendation.

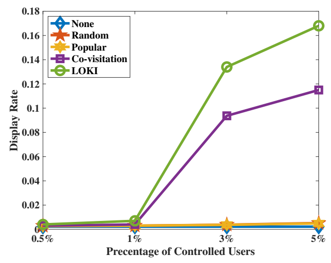
We first calculate these factors from experiments in all the four scenarios on the Gowalla dataset, and report the averaged values for both the ones that are successfully manipulated and the remaining target users, separately. The comparison is shown in Table 2.

TABLE 2: Analysis of vulnerable users on the Gowalla dataset.

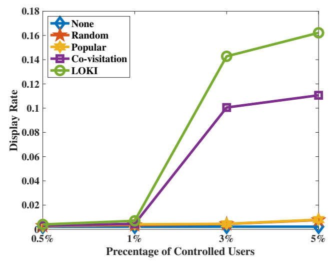
Characteristics		Average	Variance
Items per User	Succeed	35.21	34.73
	Failed	34.77	30.32
Item Popularity	Succeed	32.35	27.89
	Failed	37.72	30.51
Targets Chosen	Succeed	2.01	0.92
	Failed	0.12	0.45

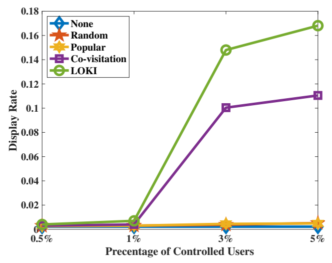
From the table, we can find that the more popular items a user chooses, the less vulnerable the user is. This is because the behavior patterns that involve popular items are supported by more samples in the training set and thus are more stable. Moreover, if the target user has chosen a target item before the recommendation, it is more likely for the user to choose the item for the second time, as the patterns of favoring target items already exist. The averaged number of activities seems to be less related to the vulnerability of a user, compared with the two aforementioned characteristics.

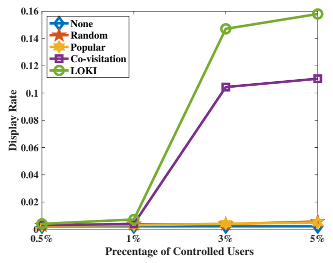
After discussing the overall experimental results and the characteristics of vulnerable users, we demonstrate the impact of two attack budgets when conducting the data poisoning attack against recommendation systems, i.e. (1) the percentage of controlled users recruited by the attacker; (2) the number of activities that each controlled user conducts.

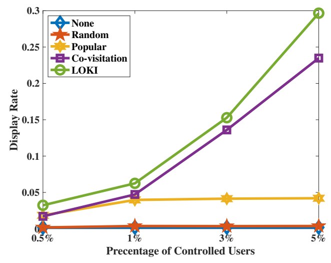


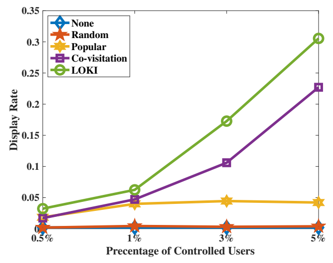
(a) VS BPRMF on Amazon

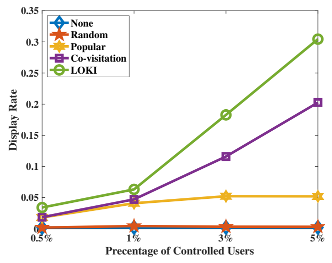

(b) VS FPMC on Amazon

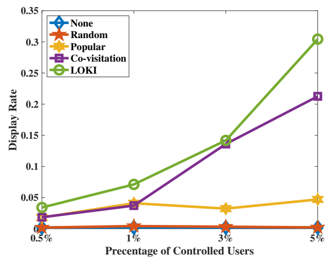

(c) VS GRU4REC on Amazon

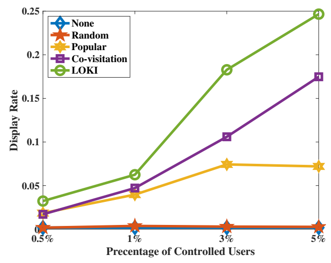

(d) VS TransRec on Amazon

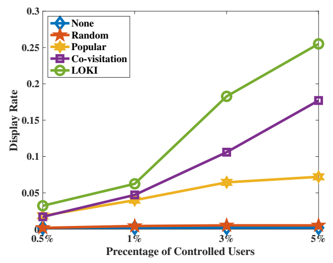

(e) VS BPRMF on Steam

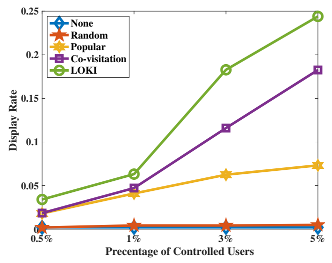

(f) VS FPMC on Steam


(g) VS GRU4REC on Steam


(h) VS TransRec on Steam


(i) VS BPRMF on Gowalla


(j) VS FPMC on Gowalla


(k) VS GRU4REC on Gowalla

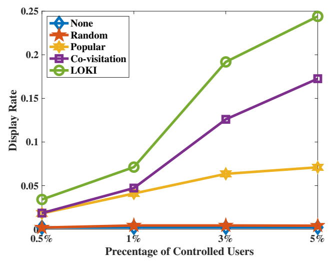

(l) VS TransRec on Gowalla

Fig. 5: Impact of the percentage of the controlled users.

4.4 Impact of the Percentage of the Controlled Users

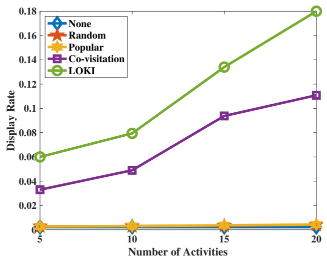
In this experiment, we consider the case where the percentage of the controlled user is low (less than 5%), and evaluate the performance of *LOKI* when this percentage is varied. Here "*percentage*" is calculated as the number of the controlled users over the number of normal users. The number of activities per controlled user is fixed to 15 and the recommendation system returns the top 10 items. The *display rate* for the real-world datasets is shown in Fig 5. From the figure, we can clearly see that the proposed *LOKI* outperforms the baselines in all cases and can successfully promote the target items. For instance, for the Steam dataset, when attacking against GRU4REC, the display rate increases to 0.07 even when the percentage of the controlled users is as low as 3%. Thus, we can conclude that the attack proposed in this paper is effective even with a scant attack budget.

4.5 Impact of the Number of Activities per Controlled User

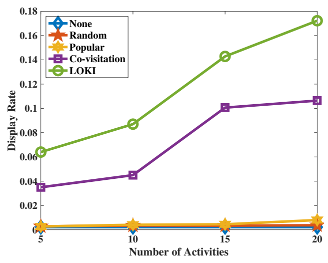
When the percentage of controlled users is given, the number of activities each controlled user conducts is another important attack factor. In this experiment, we fix the percentage of the controlled users to be 3% and vary the number of activities each controlled user conducts from 5 to 20 for all the datasets. The recommendation system returns the top 10 items. The results are shown in Figure 6. These results show that the proposed *LOKI* outperforms the baseline methods in all cases. With the number of activities per user increasing, the display rates also increase. This is because a larger number of activities grant the controlled users more capability to inject the manipulated bias information into the system. The results also show that the proposed *LOKI* is effective even with as few as $10 \sim 15$ activities per controlled user. If we look at the distribution of the length sequential user behavior samples in these two datasets, for example, the distribution derived from the Amazon dataset in Fig 7, we can see that the distributions before and after the injection are similar. Hence, injecting such generated sequential samples into the dataset is unnoticeable from the perspective of the online platform operators in practice.

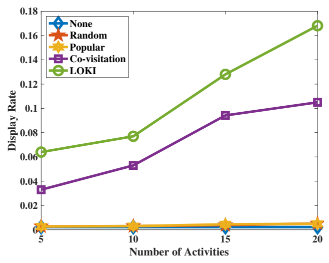
4.6 Efficiency Analysis of the Outcome Estimator

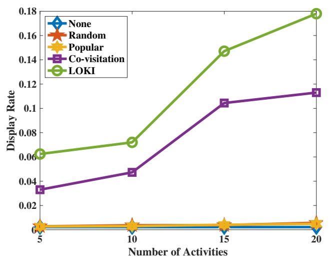
As mentioned in Sec. 3.3, retraining the recommendation model to get feedback for attack agent training is prohibitively slow (it takes hours or even days for a single retraining). Hence, we have proposed an outcome estimator based on the theory of influence function to rapidly get the intermediate attack outcome for agent training. In this section, we compare the time cost of promoting different numbers of target items to the target users, using an outcome estimator and vanilla model retraining. We report

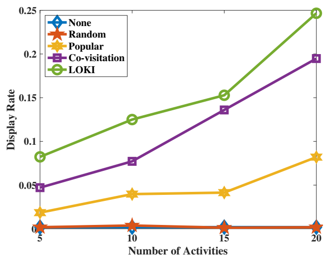


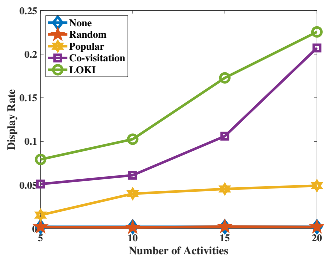
(a) VS BPRMF on Amazon

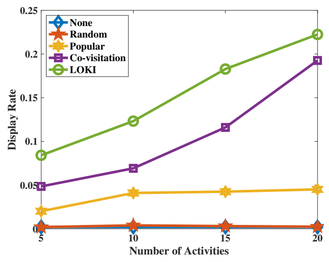

(b) VS FPMC on Amazon

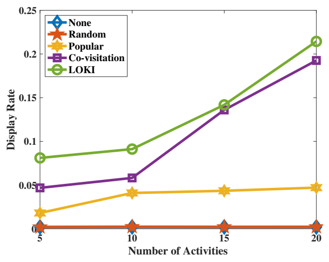

(c) VS GRU4REC on Amazon

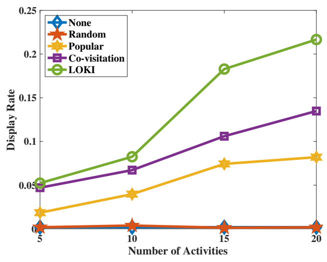

(d) VS TransRec on Amazon

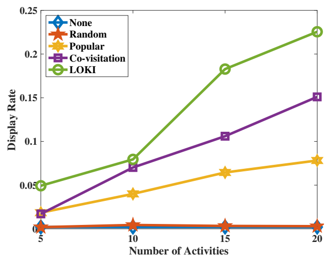

(e) VS BPRMF on Steam

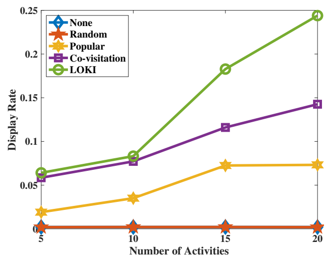

(f) VS FPMC on Steam


(g) VS GRU4REC on Steam


(h) VS TransRec on Steam


(i) VS BPRMF on Gowalla


(j) VS FPMC on Gowalla


(k) VS GRU4REC on Gowalla

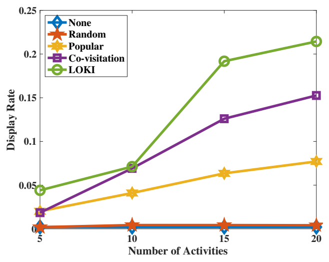

(l) VS TransRec on Gowalla

Fig. 6: Impact of the number of activities per controlled user.

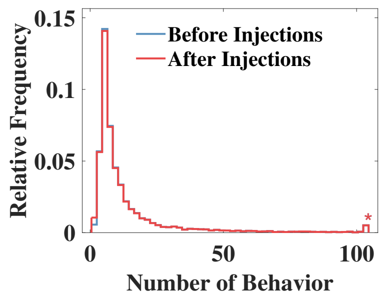


Fig. 7: Distributions of the length of sequential user behavior samples before and after the injection on Amazon dataset. (Better view with color).

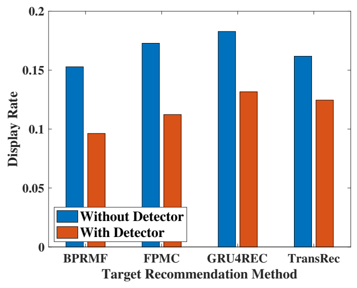


Fig. 8: Effectiveness of the proposed framework against recommendation systems with anomaly detector.

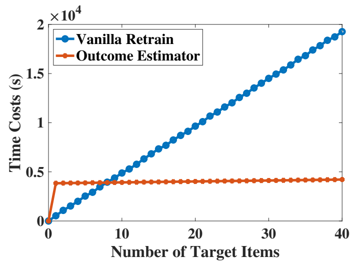


Fig. 9: The Time Costs of The Outcome Estimator v.s. the Vanilla Retraining w.r.t. The Number of Target Items on the Amazon Dataset.

the time costs of the outcome estimator versus the vanilla retraining w.r.t. the number of target items on the Amazon dataset in Figure 9, and the experiments are conducted on a GPU server with Intel Xeon E5 CPU, 256GB memory, and Nvidia Titan XP 12GB video memory. The method is implemented via Pytorch [4] and Facebook Higher [5].

From Figure 9, we can observe that when the number of target items is small (less than 10), using an outcome estimator is slower than vanilla model retraining. This is because the outcome estimator has to pre-compute the in-

4. https://pytorch.org/
5. https://github.com/facebookresearch/higher

termediate Hessian vector products for target users. Such an operation may involve unrolled first-order optimization loops to approximate higher-order gradients and not surprisingly cause some overhead. However, these results can be cached so that further attacks against more target items can be very efficient.

4.7 Effectiveness against Recommendation Systems with Anomaly Detector

In real-world online platforms, anomaly detectors are deployed to detect potential malicious users. In this section, we evaluate the effectiveness of the poisoning attacks when a recommendation system has a deployed anomaly detector. In our experiments, we deployed an unsupervised detector introduced in [30].

The intuition behind this detector is that the behavior of malicious users (i.e., choosing items) tends to form dense blocks in the user-item bi-partial graph. The detector locates the malicious users based on dense block discovery. As the attacker cannot know what kind of anomaly detectors are deployed in the target recommendation system, we do not include the unsupervised detector in the recommender simulator for the sake of fairness. The users that are detected as malicious users by the detector are removed during the experiments.

Due to space limitation, we merely report the results on the Steam dataset. Figure 8 reports the display rate of different attack methods against different fraud-aware recommender systems. We observe that the proposed *LOKI* is still effective even a fraction of controlled users are detected and removed. This phenomenon indicates that the remained adversarial samples can still accomplish the attack task. Thus, we can conclude that the proposed framework *LOKI* is effective even if the target recommendation system is equipped with an anomaly detector.

Discussions: From the results above, we can conclude that the poisoning samples generated by LOKI can bypass the unsupervised anomaly detectors that utilize benign user ratings as knowledge. However, the recommendation platform can still rely on some meta-behaviors of users to identity potential controlled users. Since the goal of the controlled users is to deliver the poisoning samples to the training set of the recommendation system, their behavior may be abnormal temporally. For instance, they may register at almost the same time, login from the same IP address, rate or click a large batch of items within a very small time window, or have regular behavior temporal gaps. Detailed camouflage strategies of the controlled users from the temporal angle are outside the scope of this paper.

4.8 Influence of the Number of Queries Needed to Adjust the Simulator

As mentioned in Sec. 3.2, the recommender simulator utilizes the feedback from the target recommender to derive the weights of individual models within the simulator. However, it is still quite meaningful to understand how many queries an attacker needs to fit a "good" simulator that captures the characteristics of the target recommender. Here, we study the number of queries w.r.t the attack performance. Due to space limitations, we only report the results on the Amazon dataset against BPRMF, with the percentage of controlled users to 3% and the number of actions per user to 15. The results are shown in Figure 10.

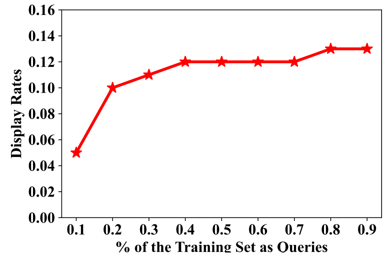


Fig. 10: The Number of Queries w.r.t the Attack Outcome

From Figure 10, we observe that when we use less than 0.2% of the training data to query the target recommender and improve the simulator, the attack performance increase significantly with the number of queries increases. However, when the number of queries keeps growing, the performance becomes steady. Such phenomena empirically show that the simulator merely needs a tiny portion of the training set to adjust the weights of individual recommendation models. Such a property makes the attack practical.

5 Related Work

In this section, we survey the related work from two aspects: general data poisoning attacks and poisoning attacks against recommendation systems.

Data Poisoning Attacks: Data poisoning attacks against machine learning algorithms have become an important research topic in the field of adversarial machine learning. This type of attack takes place during the training stage of machine learning models. The attacker tries to contaminate the training data by injecting well-designed samples to force a nefarious model on the learner. Data poisoning attacks have been studied against a wide range of learning systems. The first work in this field [31] investigated the vulnerability of support vector machines (SVM), where an attacker progressively injects malicious data points to the training set in order to maximize the classification error. Later, Mei et al. [32] generalized these attacks against SVM into a bilevel optimization framework against general offline learners with a convex objective function. Recently, poisoning attacks have been analyzed on many important machine learning algorithms, including neural networks [21], [33], [34], latent Dirichlet allocation [35], matrix factorization-based collaborative filtering [1], autoregressive models [36], [37], graph embeddings [38], and knowledge graph embeddings [39]. Existing work has almost exclusively focused on (1) whitebox settings, where the attacker observes the model architecture; (2) continuous data like image or acoustic data. In this paper, we focus on a more challenging blackbox setting, where the attacker does not know the architecture or the parameter of the target model. Instead, the attacker is merely given scant implicit feedback from the target model.

Poisoning Recommendation Systems: Similar to general data poisoning attacks, poisoning recommendation systems [1], [2], [3], [5], [40] aims to spoof a recommendation system via injecting adversarial samples, such that the system recommends as the attacker desires. The first study on poisoning recommendation systems [2] was carried out more than a decade ago. In early work, the proposed attacks are usually heuristics-driven. For instance, in random attacks [5], the attacker randomly selects some items for each injected controlled user and then generates a rating score for each selected item from a normal distribution, whose mean and variance are the same as those of the uncontaminated dataset. These methods rely on user-item ratings which do not exist in the next-item recommendation setting. Poisoning attacks [1], [3] that were proposed recently generate fake behavior that is optimized according to a particular type of recommendation system. Specifically, Li et al. [1] proposed poisoning attacks for matrix-factorization-based recommendation systems. Yang et al. [3] proposed poisoning attacks for association-rule-based recommendation systems, in which each user injects fake co-visitations between items instead of fake rating scores of items. Unlike these methods, the framework proposed in this paper does not require the details of the target system as prior knowledge. Hence, the proposed framework can be used in a broader spectrum of contexts.

6 Conclusions

In this work, we propose a sophisticated data poisoning attack against blackbox next-item recommendation system. The poisoning attack problem is formulated as a multi-step decision problem and is solved via deep reinforcement learning method. In practice, this task could be further complicated by the huge scale of recommendation dataset, the costly training time, and the access restrictions of real recommendation systems. The proposed framework leverages the influence approximation technique and the recommender simulator. To verify the effectiveness of the proposed model, we conduct an experiment on multiple real-world datasets against four representative next-item recommendation methods. Experimental results indicate that the proposed framework consistently outperforms all the baselines in terms of promoting the target items to the target users. We also study the impact of different factors on the poisoning results and what kind of users are vulnerable. Moreover, we find that the attacks are still effective when the anomaly detector, which filters malicious users from the recommendation system, is deployed. In the future, we will propose defend strategies to detect the poisoning samples generated by the attack method proposed in this paper.

Acknowledgments

The work was supported in part by the National Science Foundation under Grant NSF CNS-1742845. This work was also supported by Alibaba Group through Alibaba Research Intern Program.

References

[1] B. Li, Y. Wang, A. Singh, and Y. Vorobeychik, "Data poisoning attacks on factorization-based collaborative filtering," in *Advances in neural information processing systems*, 2016, pp. 1885–1893.

[2] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre, "Collaborative recommendation: A robustness analysis," *ACM Transactions on Internet Technology (TOIT)*, vol. 4, no. 4, pp. 344–377, 2004.

[3] G. Yang, N. Z. Gong, and Y. Cai, "Fake co-visitation injection attacks to recommender systems." in *Proceedings of Network and Distributed System Security Symposium, 2017*, 2017.

[4] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, "Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness," *ACM Transactions on Internet Technology (TOIT)*, vol. 7, no. 4, p. 23, 2007.

[5] S. K. Lam and J. Riedl, "Shilling recommender systems for fun and profit," in *Proceedings of the 13th international conference on World Wide Web*, 2004, pp. 393–402.

[6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013.

[7] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008, pp. 426–434.

[8] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, 2009, pp. 452–461.

[9] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on World Wide Web*, 2017, pp. 173–182.

[10] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 111–112.

[11] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World Wide Web*, 2010, pp. 811–820.

[12] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[13] Y. K. Tan, X. Xu, and Y. Liu, "Improved recurrent neural networks for session-based recommendations," in *Proc. of the 1st Workshop on Deep Learning for Recommender Systems*, 2016, pp. 17–22.

[14] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi, "Personalizing session-based recommendations with hierarchical recurrent neural networks," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 2017, pp. 130–137.

[15] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017, pp. 1419–1428.

[16] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *Proceedings of the 2018 IEEE International Conference on Data Mining*, 2018, pp. 197–206.

[17] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[18] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille, "Adversarial examples for semantic segmentation and object detection," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1369–1378.

[19] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[20] R. D. Cook and S. Weisberg, *Residuals and influence in regression*. New York: Chapman and Hall, 1982.

[21] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017, pp. 1885–1894.

[22] N. Agarwal, B. Bullins, and E. Hazan, "Second-order stochastic optimization for machine learning in linear time," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 4148–4187, 2017.

[23] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems*, 2001, pp. 556–562.

[24] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," 1967.

[25] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[26] R. He and J. McAuley, "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering," in *proceedings of the 25th international conference on world wide web*, 2016, pp. 507–517.

[27] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2011, pp. 1082–1090.

[28] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018, pp. 565–573.

[29] R. He, W.-C. Kang, and J. McAuley, "Translation-based recommendation," in *Proceedings of the Eleventh ACM Conference on Recommender Systems*, 2017, pp. 161–169.

[30] M. Jiang, A. Beutel, P. Cui, B. Hooi, S. Yang, and C. Faloutsos, "A general suspiciousness metric for dense blocks in multimodal data," in *2015 IEEE International Conference on Data Mining*, 2015, pp. 781–786.

[31] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[32] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[33] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization," in *Proc. of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 27–38.

[34] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," in *Advances in Neural Information Processing Systems*, 2018, pp. 6103–6113.

[35] S. Mei and X. Zhu, "The security of latent dirichlet allocation," in *Artificial Intelligence and Statistics*, 2015, pp. 681–689.

[36] S. Alfeld, X. Zhu, and P. Barford, "Data poisoning attacks against autoregressive models," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[37] Y. Chen and X. Zhu, "Optimal adversarial attack on autoregressive models," *arXiv preprint arXiv:1902.00202*, 2019.

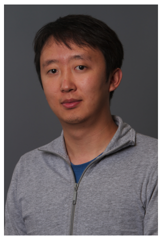
[38] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2847–2856.

[39] H. Zhang, T. Zheng, J. Gao, C. Miao, L. Su, Y. Li, and K. Ren, "Data poisoning attack against knowledge graph embedding," in *International Joint Conference on Artificial Intelligence*, 2019.

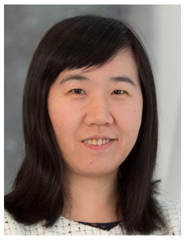
[40] H. Zhang, C. Tian, Y. Li, L. Su, N. Yang, W. X. Zhao, and J. Gao, "Data poisoning attack against recommender system using incomplete and perturbed data," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 2154–2164.

Hengtong Zhang received his Ph.D. degree from the Department of Computer Science and Engineering at SUNY Buffalo. He is a senior researcher at Tencent AI Lab. Before that he received B.E. degree from the School of Computer at Beijing University of Posts and Telecommunications in 2013. He is broadly interested in data mining and machine learning security with a focus on adversarial learning, data trustworthy analysis, text mining, and reinforcement learning.

Yaliang Li received the Ph.D. degree from the Department of Computer Science and Engineering at SUNY Buffalo in 2017. He is a research scientist at DAMO Academy, Alibaba Group. Before that he worked as a research scientist at Baidu Research, and a senior researcher at Tencent Medical AI Lab. He is broadly interested in machine learning and data mining with a focus on truth discovery, knowledge graph, question answering, differential privacy, recommendation, and more recently automated machine learning.



Bolin Ding is a research scientist in the Data Analytics and Intelligence Lab (DAIL) at Alibaba Group. He completed his Ph.D. in Computer Science at University of Illinois at Urbana-Champaign, M.Phil. in Systems Engineering and Engineering Management at The Chinese University of Hong Kong, and B.S. in Math and Applied Mathematics at Renmin University of China. Prior to joining Alibaba, he worked as a researcher in Microsoft Research. His research focuses on data privacy protection for the management, analytics, and learning of large-scale data. He has hold more than 10 US patents, and published more than 80 papers in top conferences and journals in related areas, including SIGMOD, VLDB, ICDE, KDD, ICML, NeurIPS, and ICLR.



Jing Gao is currently an associate professor in the School of Electrical and Computer Engineering at Prudue University. She received her Ph.D. in Computer Science from University of Illinois Urbana-Champaign. She is broadly interested in data and information analysis with a focus on truth discovery, crowdsourcing, multi-source data analysis, anomaly detection, information network analysis, transfer learning, data stream mining, and ensemble learning. She is also interested in various data mining applications in health care, bioinformatics, transportation, cyber security and education. She has published more than 150 papers in referred journals and conferences. She is the recipient of NSF CAREER award and IBM faculty award.