RecAD: Towards A Unified Library for Recommender Attack and Defense

Changsheng Wang[*]
wcsa23187@gmail.com
University of Science and Technology of China
Hefei, Anhui, China

Jianbai Ye[*]
jianbaiye1999@gmail.com
University of Science and Technology of China
Hefei, Anhui, China

Wenjie Wang[†]
wenjiewang96@gmail.com
National University of Singapore
Singapore

Chongming Gao
chongming.gao@gmail.com
University of Science and Technology of China
Hefei, Anhui, China

Fuli Feng
fulifeng93@gmail.com
University of Science and Technology of China
Hefei, Anhui, China

Xiangnan He
xiangnanhe@gmail.com
University of Science and Technology of China
Hefei, Anhui, China

ABSTRACT

In recent years, recommender systems have become a ubiquitous part of our daily lives, while they suffer from a high risk of being attacked due to the growing commercial and social values. Despite significant research progress in recommender attack and defense, there is a lack of a widely-recognized benchmarking standard in the field, leading to unfair performance comparison and limited credibility of experiments. To address this, we propose RecAD, a unified library aiming at establishing an open benchmark for recommender attack and defense. RecAD takes an initial step to set up a unified benchmarking pipeline for reproducible research by integrating diverse datasets, standard source codes, hyper-parameter settings, running logs, attack knowledge, attack budget, and evaluation results. The benchmark is designed to be comprehensive and sustainable, covering both attack, defense, and evaluation tasks, enabling more researchers to easily follow and contribute to this promising field. RecAD will drive more solid and reproducible research on recommender systems attack and defense, reduce the redundant efforts of researchers, and ultimately increase the credibility and practical value of recommender attack and defense. The project is released at https://github.com/gusye1234/recad.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Recommender Systems; Shilling Attack and Defense; Benchmark

ACM Reference Format:
Changsheng Wang, Jianbai Ye, Wenjie Wang, Chongming Gao, Fuli Feng, and Xiangnan He. 2023. RecAD: Towards A Unified Library for Recommender Attack and Defense. In *Seventeenth ACM Conference on Recommender Systems (RecSys '23), September 18–22, 2023, Singapore, Singapore.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3604915.3609490

1 INTRODUCTION

In recent decades, recommender systems have become increasingly important in various areas, including E-commerce recommendation [20], short video entertainment [15], news headlines [28], and online education [42]. However, the widespread use of recommender systems has also led to concerns regarding their security. When an attacker successfully attack a recommender system, users may be offended by malicious recommendations, and the platform owner may lose the trust of users in the platform. Additionally, merchants who rely on recommender platforms may suffer from commercial losses. Malicious attacks may even cause recommender systems to engage in unethical or illegal behavior, resulting in adverse effects on society. Therefore, it is essential to address these security risks and ensure the integrity of recommender systems to maintain user trust and prevent any negative consequences.

Recently, industry and academia are trying to develop strategies for both attacking and defending recommender systems, especially pay attention to the techniques about shilling attacks and defense [43]. In shilling attacks, fake users are generated and assigned high ratings for a target item, while also rating other items to act like normal users for evading. There are three main kinds of shilling attack methods: heuristic methods, gradient methods, and neural methods. Heuristic methods [5, 21, 27] involve artificially or randomly selecting items based on user preferences, and then intuitively fabricate interaction information. Gradient methods [12, 23] estimate the gradients of maximizing attack objectives to directly optimize the interactions of fake users, while Neural methods [25, 26, 38] optimize the neural networks parameters to predict the optimal fake user behaviors for maximizing attack objectives.

To combat these attacks, many defense methods [7, 10, 30] are emerging to enhance the defense ability of existing recommendation models. Essentially, these defense models aim to distinguish between fake data generated by the attack model and real user data,

[*] The first two authors contributed equally to this research.
[†] Corresponding author.

RecSys '23, September 18–22, 2023, Singapore, Singapore

ACM ISBN 979-8-4007-0241-9/23/09…$15.00
https://doi.org/10.1145/3604915.3609490

ensuring that the recommendation model uses as much real data as possible [44]. Currently, the mainstream defense model can be divided into three types according to whether the label of fake users can be obtained [1, 10, 37]. As new attack methods emerge, defense models are constantly evolving to keep up with these threats. Therefore, staying up-to-date with the latest attack and defense methods is essential for maintaining the security of recommender systems.

With the continuous emergence of new attack and defense algorithms in the field of recommender system security, there are several research challenges that deserve attention. Firstly, while many articles provide details about their experiments, there is often a lack of standardization in dataset processing methods, which can lead to unfair comparison. Secondly, there is a lack of unified settings for attack experiments. Various works usually leverage different experimental setting, making it difficult to compare and evaluate different models. It is critical to establish a standardized approach for similar attack settings to facilitate the comparison and evaluation of different models. Additionally, many works lack public code, which can create repetition and difficulties for subsequent researchers trying to advance the field. To address these challenges, researchers should strive to provide clear and standardized descriptions of dataset processing methods, unified settings for attack experiments, and make their code publicly available to facilitate replication and extension of their work. These efforts can help promote the development of the recommender attack and defense and contribute to more robust and effective recommender system security solutions.

To address the aforementioned challenges, we have initiated a project to develop a unified framework for Recommender Attack and Defense, named RecAD. RecAD aims to improve the reproducibility of existing models and simplify the development process for new recommender attack and defense models. Our benchmarking library design is innovative and effective, revealing several advantages compared to earlier attempts.

- **Unified library framework.** RecAD is implemented using Pytorch[1], one of the most popular deep learning frameworks. The library is composed of three core modules, namely the data module, model module, and evaluation module. The library supports a vast array of options provided by each module, and a straightforward configuration ensures that users can promptly complete algorithm reproduction and comparison. The seamless interface integration of the three core modules also enables the minimal adjustment for incorporating new algorithms, allowing for continuous development and extension within our framework in the future.
- **Comprehensive benchmark models and datasets.** RecAD provides support not only for replacing individual models but also for integrating a wide range of research issues. From generating fake attack data to defending against existing data and injecting data into victim models, RecAD covers the entire spectrum of shilling attack and defense research. It provides an array of choices for all models and datasets, guaranteeing an ample assortment of combinations for researchers to utilize. This allows them to execute, compare, and assess the entire procedure, relying on lucid instructions and configurations. RecAD is highly adaptable and scalable, with original dataset copies that can be effortlessly transformed into a practical form using the provided preprocessing tools or scripts. Additionally, we are continuously expanding our library with additional datasets and methods to better serve the needs of the community and researchers.
- **Extensive and standard evaluation protocols.** RecAD offers evaluation methods from two perspectives: attack evaluation and defense evaluation. Researchers interested in continuing the offensive direction or those focusing on the defensive direction can use the corresponding evaluation methods. Additionally, it provides standard evaluation techniques for assessing the effectiveness of defense models, encapsulating the entire evaluation process within a singular module enables RecAD to more readily accommodate more evaluation techniques, thus enhancing its adaptability and versatility.
- **Openness and high integration of models.** Openness is crucial for promoting transparency, collaboration, and reproducibility in computer science research. RecAD adopts a highly integrated approach, simplifying the relationships between modules as much as possible and making the corresponding parameters publicly available at each module. This ensures that subsequent researchers who use our framework to add new models only need to make the corresponding modules public, allowing other researchers to quickly and efficiently reproduce the work and ensure the openness of the field in the future.
- **The generalization of attacker's knowledge.** The attacker's knowledge level directly impacts the effectiveness of the attack. A high degree of accessible knowledge about the recommender system allows an attacker to craft adversarial examples that can evade the model's defenses. RecAD can elevate white-box attacks to gray-box attacks and customize the proportion of data accessible by the attackers for gray-box attacks [13], promoting the fair comparison between a wide range of attackers.

2 RELATED WORK

2.1 Overview of Shilling Attack and Defense

In the past two decades, researchers have conducted experiments to demonstrate the feasibility of attacking real-world recommender systems, such as YouTube, Google Search, Amazon, and Yelp. These experiments have shown that it is possible to manipulate recommendation systems in practice, resulting in an increasing focus on this field from both the academic community and industry. To promote its development, researchers have typically focused on either shilling attacks or defense mechanisms. With the advancements in deep learning, the field has seen a notable increase in the effectiveness of these methods.

2.2 Shilling attack

The objective of an shilling attack is to interfere with the recommendation strategy of the victim recommender system through a series of measures [9, 31, 32]. The ultimate goal is to enhance the

[1] https://pytorch.org/.

exposure of a specific target item among all users after the recommender model is trained. To achieve this objective, attackers often inject fake users into the historical interactive data, or training matrix, of the recommender system. However, if these fake users are not adequately protected, they will be sent into the recommender system model during the training process, thus disrupting the recommendation strategy of the system. As a result, the key challenge of an shilling attack is to construct the interaction behaviors of the fake users. The interaction behaviors of the constructed users can generally be classified into three categories:

- **Heuristic attacks.** Heuristic attacks involve selecting items to create fake profiles based on subjective inference and existing knowledge. The goal is to strengthen the connection between fake users and other real users while evading defense methods and achieving exposure enhancement of the final target item [5, 27]. Currently, existing methods include the Random Attack [21], Average Attack [22], Bandwagon Attack, and Segment Attack [6]. The Random Attack is a low-knowledge method that selects filler items randomly, while the Average Attack selects filler items randomly and requires more knowledge. In the case of an Average Attack, the target item needs to be given the highest rating to implement a push attack. Segment Attack selects items of the same category as the target item and maximizes their rating, with the goal of creating a stronger correlation with the corresponding target user among real users so that it can attack more effectively.
- **Gradient attacks.** Gradient attacks involve relaxing the discrete space to a continuous space to ensure that the objective function can be optimized by the gradient to achieve the optimal attack effect. For instance, Li et al. [12, 23] developed poisoning attacks optimized for matrix factorization-based recommender systems, while Yang et al. [45] developed poisoning attacks optimized for co-visitation rule-based recommender systems. Additionally, there are gradient attack methods based on Next-item [49], and graph [14]. However, all Gradient Attacks require known types of recommender systems to carry out specific optimization, which does not have good generalization. Moreover, in order to achieve bilevel optimization [19], directly adjusting interactions according to gradients involved transforming the discrete interactions into continuous optimization. During the process of re-discretization, information loss occurred, leading to sub-optimal results and the lack of robustness in the model.
- **Neural attacks.** Neural Attacks, primarily inspired by deep learning [19], generate realistic profiles that have a significant impact on recommender systems by optimizing the parameters of neural networks to maximize the objective function. WGAN [2] draws on Wasserstein's distance, which has shown better empirical performance than the original GAN [17]. It can emulate real user behavior with fake user behavior to achieve the effect of fake user behavior. AIA [38] reviewed the bi-level optimization problems of the surrogate model and proposed time-efficient and resource-efficient solutions. AUSH [25] and Legup [26] solve the randomness caused by noise generation in common models, making the generated template artificially based on known knowledge, resulting in a more undetectable configuration file. When the attacker's knowledge is limited to a black box, researchers use RL attack [11, 36, 52] to complete the attack, with the attacker adjusting changes based on feedback given by the spy user in the victim model. The methods of Neural Attacks all show better performance on real datasets than Gradient and Heuristic attacks.

In addition to the challenges associated with constructing effective shilling attacks, another emerging issue is the **knowledge of the attacker** [5]. In today's world, data security and privacy are increasingly important to both users and companies. This makes it increasingly challenging for attackers to obtain the necessary user data to construct effective attack models. As a result, researchers have begun to consider the attacker's knowledge as a key constraint for the attack model. The attacker's knowledge can be classified into three categories: *white box*, *grey box*, and *black box*. In a white box attack, the attacker has complete knowledge of the target recommender model, which includes all the data of the victim model used for training and the network structure and parameters of the victim model. In a grey box attack, the attacker can only access part of the training set of the target model and has no knowledge of the victim model. In a black box attack, only some spy users are allowed as attack feedback.

In addition to shilling attacks, there are other types of attacks, such as attacks that involve modifying the real user interaction history [48] or attacks based on federated learning recommender models [33, 34, 46, 55]. The former is not very effective due to the adoption of multiple privacy protection mechanisms by real recommender platforms, such as email and mobile phone hardware binding. Therefore, this method is easily detected and defended by the platform and is insufficient for a large-scale attack. On the other hand, the latter is still in the theoretical research stage and the models proposed are too basic, at the same time this kind of method has not yet been implemented by companies. This means that the criteria for the above two methods still need to be explored by more researchers.

2.3 Defense

A defense model can be viewed as a checkpoint responsible for detecting possible fake users in the data before it is sent to the recommender model. The defense model eliminates fake users to ensure that the recommendation results are not interfered with by attackers to the greatest extent possible. Some defense models attempt to find the law of data distribution from all the data or obtain the probability of the corresponding label through probability methods to predict and classify. Currently, the defense direction can be classified into three categories:

- **Supervised defense models,** which need to be pre-labeled with true and false data. The goal of the model is to learn the relationship between the input and output variables, so that it can make predictions on new data. The learning process involves minimizing the difference between the predicted output and the true output for each example in the training data. In other words, the model is trained to approximate the mapping from inputs to outputs. In the direction of recommender defense, Supervised work emerges in the initial exploration of this field, such as CoDetector [10], DegreeSAD [24], BayesDetector [44].

- **Semi-supervised defense models,** as explored in [7, 42], aim to use a minimal amount of false data while still maintaining the purpose and accuracy of the supervised method. This is because attackers typically use a small amount of data to launch attacks, leading to an inherent imbalance between true and false training samples, highlighting the crucial importance of maintaining the supervised aspect of the method.
- **Unsupervised defense models,** which have been intensively investigated in recent years, including traditional machine learning models such as probabilistic models [1], statistical anomaly detection [4], PCA [30], SVM [54], and K-means [8]. More recently, network models have been used for detection, such as Graph Embedding [51], Sequential GANs [35], Recurrent Neural Network [16], and Dual-input CNN [47].

In addition to the model-based prediction introduced above to realize the defense of the recommender platform, some scholars also have trained the recommender model by using adversarial data training [18, 29, 37, 41] so that the recommender model can have better generalization in the face of fake data.

2.4 Benchmarking for Recommender Attack and Defense

Despite the recent growth in the field of RS security, different studies have employed different data sets, evaluation methods, and knowledge constraints, resulting in significant fairness issues when comparing different models. This has had a negative impact on the steady development of the field. Although some works have attempted to address these issues in the past, there is still a need for a comprehensive and unified library to solve the current dilemma.

For instance, in AUSH [25], the author provided a code that integrated multiple attack models, but the workflow was inefficient and required a significant amount of time for subsequent researchers to study the code structure. Additionally, the code was not friendly for adding new models under the same framework and focused more on the study of attack models. Moreover, the code only provided a limited data set and did not include the data processing method, making it difficult to test the model on a working public data set. In SDLib[2], some defense models and attack models were provided, but the attack model was outdated and did not complete the entire process from attack generation to defense detection and injection into the recommender model. Furthermore, the code language used in this work was obsolete. Our framework overcomes the limitations of previous methods by abstracting each component into relatively independent modules, ensuring the unity and extensibility of the model. This allows for better maintenance and development of the framework in the future.

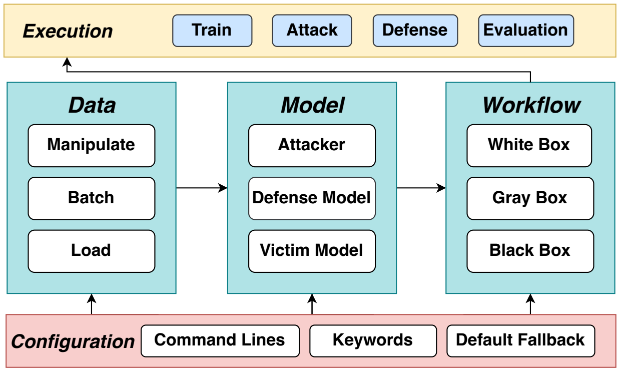


Figure 1: The overall framework of RecAD.

Table 1: Collected data in our library.

Dataset	#Users	#Items	#Interations	#Density
MovieLens-1m*	5,950	3,702	567,533	0.257%
Yelp*	54,632	34,474	1,334,942	0.070%
Amazon-Game*	3,179	5,600	38, 596	0.216%
Book-Crossing	105,284	340,557	1,149,780	0.003%
Last.FM	1,892	17,632	92,834	0.278%
Epinions	116,260	41,269	188,478	0.004%
Gowalla	107,092	1,280,969	6,442,892	0.005%

*means the dataset is used in the experiments and only kept high-frequency users and items (at least 10 interactions).

3 THE LIBRARY-RECAD

The overall framework of RecAD is illustrated in Figure 1. At the bottom, our library maintains a flat structure for the default hyperparameters globally, and the core components are built upon it with automated parameter loading (see Section 4). Our library abstracts the core modules at three levels: data, model, and workflow. In the following, we briefly present the designs of these three core modules.

3.1 Data Module

The data module serves as the fundamental part of the entire library, as it provides essential runtime information such as batches and indicators of scale. It takes charge of dataset loading, batch generation, and fake data manipulation.

3.1.1 Dataset Loading. To create an actively-contributed benchmark, it is important to make the addition of new datasets as easy as possible. Therefore, we have designed the data module to keep the required dataset formats simple and flexible. At present, our library only requires the human-readable CSV format with specific column names to load datasets into explicit or implicit interactions. This design decision allows users to easily add their own datasets to the library without having to modify the codebase. Our library already supports multiple datasets (as shown in Table 1), and we also provide auxiliary functions to convert datasets from other well-known recommender frameworks, such as RecBole [53]. This

[2] https://github.com/Coder-Yu/SDLib.
[2] https://grouplens.org/datasets/movielens/1m/.
[3] https://www.yelp.com/dataset.

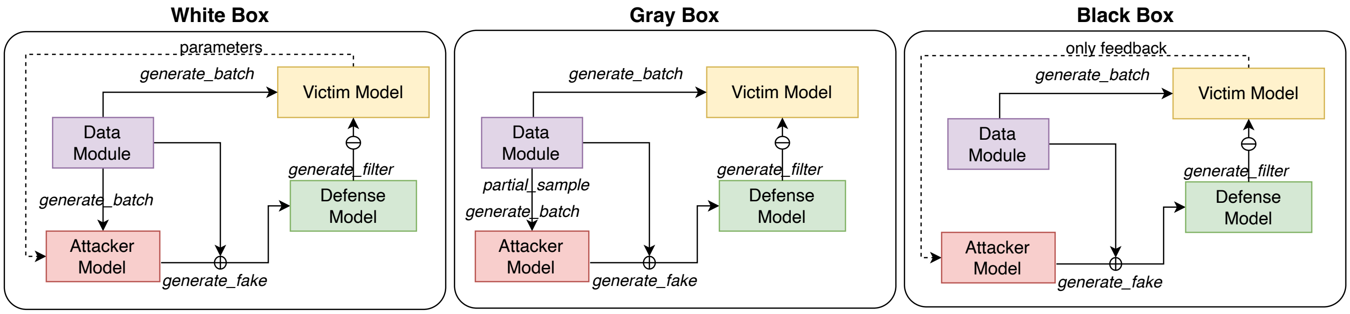
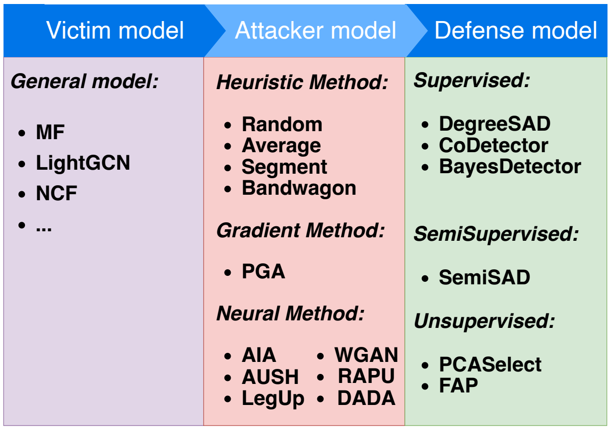


Figure 2: Component workflow under different attack knowledge.



Victim model	Attacker model	Defense model
General model: • MF • LightGCN • NCF • ...	*Heuristic Method:* • Random • Average • Segment • Bandwagon *Gradient Method:* • PGA *Neural Method:* • AIA • WGAN • AUSH • RAPU • LegUp • DADA	*Supervised:* • DegreeSAD • CoDetector • BayesDetector *SemiSupervised:* • SemiSAD *Unsupervised:* • PCASelect • FAP

Figure 3: The models that are supported by RecAD.

provides further flexibility for users to utilize the datasets that they are familiar with.

3.1.2 Batch Generation. Our library prioritizes seamless integration between datasets, models, and workflows, which presents challenges for batch generation. To address this, we design a flexible and generic interface (*generate_batch*). The interface allows the caller to provide runtime configuration parameters (*e.g.*, pairwise sampling; binarizing the ratings) and dispatches itself to the corresponding behavior. This design reduces the workloads on developers who are attempting to adapt their data and allows them to focus on providing as much runtime information as possible.

3.1.3 Fake Data Manipulation. In our library, we recognize the importance of addressing the manipulation of fake data during runtime. Specifically, we must account for both the injection of fake data from attacker models and the filtering of fake data by defense models. We address this challenge with unified interfaces named *inject_data* and *filter_data*, respectively. These interfaces are called by the attacker and defense models to manipulate the dataset.

3.2 Model Module

The model implementation is the most versatile part of the library, and we offer maximum flexibility to accommodate different approaches. To account for the similarities and differences between models, we introduce a general base model and its successors: the victim, attacker, and defense models. Figure 3 presents the models that have been implemented.

3.2.1 Base Model. We don't provide framework-level abstractions for model optimization. Instead, the models are responsible for their own single-epoch training and evaluation, which can be implemented through a set of auxiliary functions provided by the library. This design choice is aimed at reducing the complexity of the framework and enabling the integration of a wide range of models, without requiring modification of the framework-level abstractions for each individual model. To facilitate this, we use unified interfaces (*train_step*, *test_step*) that enable the callers to initiate the training or evaluation process of the models.

3.2.2 Victim Model. Victim models are recommender models, and the library provides a unified interface for training and testing them. This makes it easy to integrate any victim model into the library without the need for modification of the core framework.

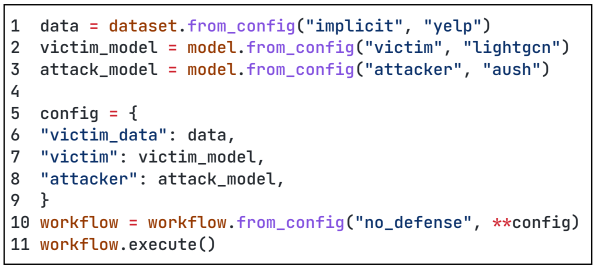
3.2.3 Attacker Model. In our library, the training of the attacker model shares the same interface with victim models (*i.e. train_step*). After training, the attacker model generates the fake data through a unified interface (*generate_fake*) and then forwards the contaminated data to the next module. Since the full set of the dataset is not necessarily exposed (*e.g.* Gray box attacking in Figure 2), *generate_fake* should explicitly receive the target dataset as a parameter.

3.2.4 Defense Model. The defense model is trained on the attacked data through the same training interface. The objective of the model is to output a filtered dataset with fake data removed. Our library summarizes a unified interface *generate_filter* to wrap the implementation details of each defense model.

3.3 Workflow Module

This module is the corresponding abstraction of different attack knowledge (Figure 2). The workflow module holds the instantiations of the data module and model module, controlling the exposure of data and the interaction of modules. It also contains the boilerplate code for the training loop and evaluation callbacks (*e.g.*, early stop; report after training).

3.3.1 Data Exposure. The data exposure level for different models varies depending on the attack knowledge settings and the running stages (as shown in Figure 2). For instance, the attacker model may



```
data = dataset.from_config("implicit", "yelp")
victim_model = model.from_config("victim", "lightgcn")
attack_model = model.from_config("attacker", "aush")

config = {
"victim_data": data,
"victim": victim_model,
"attacker": attack_model,
}
workflow = workflow.from_config("no_defense", **config)
workflow.execute()
```

Figure 4: A code snippet of module instantiations of RecAD.

be exposed separately to full, partial, or zero training data. Similarly, the victim model may be trained on clean data initially and later re-trained on the contaminated data during the attack process. The workflow module in our library is responsible for constructing the appropriate data flow according to the attack knowledge and ensuring that no accidental data leakage occurs. This way, our library provides a flexible and secure environment for implementing and testing various attack and defense models under different settings.

3.3.2 Module Interaction. The interactions between modules vary between attacks. In a white-box attack, the model has direct access to all the training data, whereas, in a black-box attack, the model receives feedback from the victim without any access to the training data. For workflows [26, 38] where no defense model is involved, the fake data generated by the attacker model flows directly into the victim's training without filtering. The workflow module arranges the dependencies of modules and prevents any inappropriate interactions between them.

3.3.3 Training & Evaluation. In order to better control the data exposure and module interaction, we give the workflow module the responsibility for launching the training and evaluation of the contained models. The workflow module contains the boilerplate codes for wrapping the training loop outside the models' *train_step*. Also, we design a hooking mechanism to provide flexibility for models to set up their evaluation callbacks. This allows models to define their own evaluation metrics to evaluate the model's performance at different stages of the training process.

4 USAGE GUIDELINE OF THE LIBRARY

In the following two parts, we first show the typical usage to instantiate the existing modules of our library, then detail the steps to extend our library with a new implementation.

4.1 Module Instantiations

Attacking a recommender system often involves using multiple datasets and machine learning models, which makes the training and testing process more complex than for regular recommender systems. Our library simplifies this process by exposing the necessary modules to users and providing a unified interface called *from_config* for instantiating them (Figure 4). Two kinds of parameters may be needed from *from_config*: hyper-parameters and runtime parameters.

4.1.1 Hyper-parameters. Our library employs the hashing table to store all default hyper-parameters of modules together and offer global access across programs. While instantiating, our library automatically loads default parameters on-fly from the hashing table and updates them from the keyword arguments passed by the user. The decoupling of default hyper-parameters and the actual module implementation facilitates a quick overview of configurable parameters for the user.

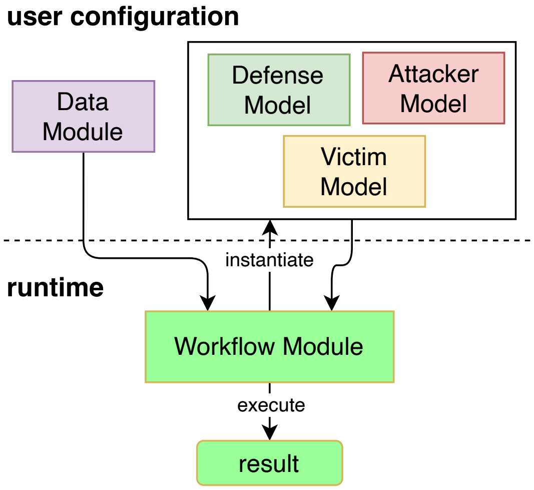


Figure 5: An illustration of lazy instantiation.

4.1.2 Runtime Parameters. Runtime parameters are the parameters that won't be settled before the runtime. For example, the model module in Figure 4 normally needs the numbers of the user and item to create the embeddings when instantiating. Due to the data injection or data filtering from the attacker model, the actual numbers of the user and item are not known before the runtime. But the dependency between the model and data module is clear, and it is burdensome to ask the user to manually pass in the required instances in the program. Hence, we implement lazy instantiation (Figure 5) to make runtime parameters transparent at the user level. The module won't actually instantiate at the time the user call *from_config* if the needed runtime parameters are not passed. Instead, the workflow will sort out the dependencies between modules and automatically fill in the required runtime parameters to complete the instantiation. This decouples the instantiation of modules from the availability of runtime parameters, making the library more flexible and adaptable to different scenarios.

4.2 Module Extension

In our library, we provide the base class for all the core modules: *BaseData*, *BaseModel*, and *BaseWorkflow*. We require the extended module must be the corresponding base class's subclass so that the necessary abstract interfaces can be called properly.

4.2.1 General Module. Two abstract methods must be implemented for all the modules:

- ***from_config***: users pass arguments to this method to instantiate a new module. Our library has already implemented the argument sanity checking and overwriting the default hyper-parameters in the father class. A new module should assign the default hyper-parameters in this method.

Table 2: Overall attack performance on three recommendation datasets.

		ML-1M				Yelp				Amazon			
Attack Method	Attack Knowledge	HR@10	HR@20	HR@50	HR@100	HR@10	HR@20	HR@50	HR@100	HR@10	HR@20	HR@50	HR@100
No Attacker	None	0.0050	0.0109	0.0297	0.0656	0.0114	0.0190	0.0375	0.0630	0.0000	0.0000	0.0003	0.0016
RandomAttacker	White Box	0.0050	0.0082	0.0228	0.0457	0.0078	0.0112	0.0214	0.0362	0.0000	0.0000	0.0000	0.0000
SegmentAttack	White Box	0.0069	0.0123	0.0288	0.0630	0.0057	0.0083	0.0153	0.0258	0.0397	0.0520	0.0675	0.0832
BandwagonAttack	White Box	0.0059	0.0119	0.0267	0.0592	0.0066	0.0114	0.0257	0.0431	0.0050	0.0205	0.0523	0.0854
AverageAttack	White Box	0.0016	0.0044	0.0167	0.0400	0.0053	0.0090	0.0169	0.0284	0.0085	0.0170	0.0463	0.0914
WGAN	White Box	0.0023	0.0060	0.0149	0.0340	0.0143	0.0177	0.0254	0.0344	0.1646	0.1788	0.2043	0.2226
AIA	Gray Box 20% data	0.0078	0.0180	0.0459	0.1007	0.0187	0.0273	0.0465	0.0686	0.0441	0.0873	0.4278	0.4839
AUSH	Gray Box 20% data	0.0071	0.0151	0.0434	0.0945	0.0135	0.0217	0.0393	0.0617	0.0583	0.1170	0.4392	0.4805
Legup	Gray Box 20% data	0.0094	0.0130	0.0283	0.0471	0.0068	0.0099	0.0162	0.0242	0.1847	0.2015	0.2286	0.2566

- ***info_describe***: modules interact through this method. The method should return a hash table with the named variable that this module can expose publicly.

4.2.2 Core Modules. The core modules have specialized interfaces that need to be implemented in addition. We have discussed most of the below interfaces in Section 3.

- **Data Module**: The most important interface for this module is *generate_batch*. The interface should take the caller's keyword arguments as the input, and return the correct batches of the dataset for later training or testing.
- **Model Module**: Right now, three kinds of models are considered: victim model, attacker model, and defense model. They are all required to be implemented with two interfaces: *train_step* and *test_step* to perform one-epoch training or testing. Besides, for the attacker model and defense model, *generate_fake* and *generate_filter* need to be implemented, respectively.
- **Workflow Module**: An interface named *execute* should be implemented for users to explicitly launch the whole workflow. Inside the interface, the implementor should correctly instantiate and arrange the modules.

5 EXPERIMENTS

This section showcases the application of RecAD by implementing various representative attackers and detection models. Through a comparison of the outcomes produced by these models, valuable insights can be derived.

5.1 Comparison of Attackers

We illustrate the performance of all attackers in three recommendation datasets in Table 2. The goal of all attackers is to make the target items get higher rankings, *i.e.*, larger HR@k.

The two gray box methods, AIA and AUSH, exhibit the best performances across all metrics and datasets, which attests to the efficacy of neural network-based approaches. In contrast, the performance of Legup is less consistent. For instance, Legup displays optimal performance with respect to HR@10 in Amazon, whereas it experiences a decrease in rank, falling to the middle-lower range, with respect to HR@20, HR@50, and HR@100. Additionally, in Yelp, Legup performs inadequately across all metrics, and its weak robustness is further illustrated in Figure 6. The Legup model has been observed to exhibit unstable performance, which can be attributed to its training methodology that involves the simultaneous use of three distinct models. This approach has resulted in the same training instability issues that are commonly associated with GANs. Specifically, the use of multiple models in training can lead to a lack of consistency in the learned representations across the different models. This, in turn, can create conflicts in the optimization process and cause the model's performance to become highly dependent on the initialization and training procedures.

The heuristic method RandomAttacker exhibits the poorest performance across all metrics and datasets, even when compared to a situation in which no attacker is utilized. In other words, RandomAttacker not only fails to enhance the ranking of target items but also results in a lowered ranking for those items. Due to the highly randomized nature of heuristic attacks, the resulting attack target can be skewed by the random effects, resulting in a greater impact of inserting fake users on vulnerable users.

In addition, other methods, including SegmentAttack, BandwagonAttack, AverageAttack, and WGAN, also occasionally result in a poorer ranking for the targeted items. Consequently, there remains substantial room for the development of effective attacker methods in recommender systems. Currently, the existing methods of attack are characterized by significant limitations, such as their capacity to target only specific structures of recommendation algorithms or their limited ability to transfer attacks to models in other domains.

5.2 Comparison of Defenders

We choose three supervised methods (DegreeSAD, CoDetector, and BayesDetector), one semi-supervised method (SemiSAD), and two unsupervised methods (PCASelectUser and FAP) to act as defenders, tasked with protecting the victim model from five attacker models. The goal of these experiments is to evaluate several defense methods using our framework process.

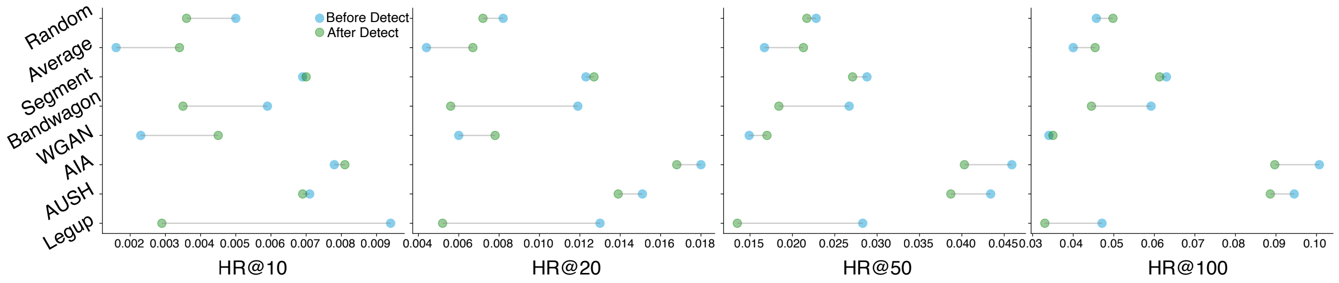


Figure 6: Performances of attackers before and after detection by PCASelectUser in the ML-1M dataset.

Table 3: Defense performance against five representative shilling attackers.

		AIA			Legup			WGAN			RandomAttacker			SegmentAttacker		
		Gray Box 20% data			Gray Box 20% data			Gray Box 20% data			White Box			White Box		
Detect Method	Data Lable	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
DegreeSAD	True Data	0.782	0.845	0.812	0.782	0.841	0.810	0.782	0.840	0.810	0.780	0.843	0.810	0.781	0.840	0.810
	Fake Data	0.720	0.630	0.672	0.717	0.632	0.672	0.716	0.632	0.671	0.718	0.627	0.669	0.715	0.631	0.670
CoDetector	True Data	0.898	0.861	0.879	0.887	0.885	0.886	0.897	0.873	0.885	0.908	0.877	0.892	0.904	0.880	0.892
	Fake Data	0.796	0.846	0.820	0.840	0.843	0.841	0.811	0.844	0.827	0.809	0.854	0.831	0.823	0.857	0.840
BayesDetector	True Data	0.943	0.946	0.945	0.945	0.945	0.945	0.936	0.943	0.940	0.944	0.936	0.940	0.938	0.943	0.940
	Fake Data	0.915	0.910	0.912	0.914	0.913	0.913	0.909	0.899	0.904	0.896	0.908	0.902	0.909	0.902	0.905
SemiSAD	True Data	0.895	1.000	0.945	0.911	1.000	0.954	0.921	1.000	0.959	0.903	1.000	0.949	0.892	1.000	0.943
	Fake Data	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
PCASelectUser	True Data	0.953	0.985	0.969	0.954	0.986	0.970	0.954	0.986	0.970	0.952	0.983	0.967	0.952	0.983	0.967
	Fake Data	0.100	0.034	0.050	0.170	0.057	0.086	0.170	0.057	0.086	0.000	0.000	0.000	0.000	0.000	0.000
FAP	True Data	0.963	0.992	0.977	0.970	1.000	0.985	0.970	1.000	0.985	0.872	0.296	0.442	0.953	0.658	0.728
	Fake Data	0.526	0.184	0.272	1.000	0.325	0.491	1.000	0.343	0.511	0.920	0.647	0.967	0.968	0.969	0.961

We present three evaluation metrics for the predicted label results, namely F1-score, Recall, and Precision. To evaluate the performance of our model, we split our data into two categories: True Data and Fake Data. True Data refers to the original real data used to train the recommender system, while Fake Data represents the fake data generated by attackers. We have provided three evaluation metrics for each category instead of treating them as a whole, as we believe that an effective detector should be able to not only successfully predict fake data but also avoid misclassifying real data. Hence, we hope that the values for the three metrics corresponding to both types of data are as high as possible, indicating that the detector models have better defensive performances from two dimensions.

Based on the data presented in Table 3, it can be observed that although the three supervised methods may not exhibit the highest performance, they demonstrate consistent performance against various attacks. Conversely, the semi-supervised method is not effective in defending against attacks due to the requirement of more data for training and evaluation, which is restricted by the attack budget in our approach. Consequently, the semi-supervised method misclassifies both real and fake data. Among the unsupervised methods, FAP shows promising results for certain attacks and outperforms other defense methods, but still displays certain limitations in some metrics.

5.3 Robustness of Attackers Encountering Detection

For illustration, we visualize the performance comparison before and after detection by PCASelectUser. Due to space limitations, we only present the results in the ML-1M dataset.

From Figure 6, we can observe that the performance of all attack methods will vary after detection. In heuristic methods, Bandwagon exhibits a notable difference in performance before and after detection. After detection, there is a marked decrease in all four HR metrics. The potential reason is that Bandwagon selects the popular items as users' fakes preferences, where this pattern is relatively easier to identify, making the generated data easier to be detected.

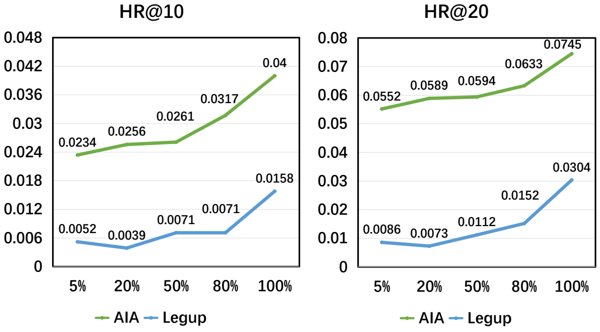


Figure 7: Attack performance of AIA and Legup with different proportions of data.

In the neural methods, Legup demonstrates a similar phenomenon with a more significant performance difference before and after detection. In the HR@10 metric, Legup outperforms all other attacker methods before the detection, however, it has the worst performance after the detection. On HR@20, HR@50, and HR@100, it remains to be the worst one after detection. One possible reason for this is that Legup's optimization objective is more complex and it includes a greater number of modules compared to other attacker methods. Both the two attackers show poor robustness before and after detection, while other methods exhibited relatively high robustness after detection.

Counterintuitively, the results of AverageAttack and WGAN demonstrate an inverse effect: the targeted items rank higher after detection, *i.e.*, the detection process helps the attackers achieve their purposes. There are two potential explanations. The first possibility is that this method generates users that are virtually indistinguishable from real ones, rendering detection modules theoretically unable to identify them. The second explanation is that the mechanism by which the method generates fake users was not taken into account by the detection module, allowing it to evade detection by this detection method.

5.4 Comparison of Defense Evaluation.

In light of the results presented in Figure 6 and Table 3, we have observed that relying solely on either injection-based or label prediction evaluation for assessing the performance of defense models may not be adequate. For instance, in the case of the WGAN method, the injection-based evaluation indicates that the exposure rate of the target item is even higher after defense than the direct attack, while the label prediction evaluation suggests that the current defense approach is more effective in true and false prediction. Thus, we urge future researchers in this field to use both evaluation methods to ensure the practical effectiveness of defense models. Our framework supports both evaluation processes, eliminating the need for researchers to repeat work.

5.5 Effect of knowledge of the attackers.

To investigate the impact of the scale of models' knowledge, *i.e.*, the quantity of training data for attacker models, on the results, we visualize the performance of two neural models (AIA and Legup) as the amount of training data varies. The results are shown in Figure 7. From the results, we observe that the performance of both models increases as the amount of data increases, albeit with a slight fluctuation for AIA at x50%. This inspires us to provide the attack model with more knowledge. However, as we venture into the exploration of novel attack algorithms, we must also take into consideration the importance of placing constraints on the known knowledge of these algorithms. This is particularly crucial, as it creates a trade-off between the scale of a model's knowledge and the effectiveness of the attack. Striking the right balance between these two factors is key to maximizing the potential impact of new attack algorithms while minimizing their potential negative consequences. This trade-off raises the critical question of how we can best manage the scale of models' knowledge while still maintaining the efficacy of the attack. This requires a comprehensive understanding of the intricate interplay between the scale of knowledge and the effectiveness of the attack, and a willingness to explore new frontiers of research and development in order to push the boundaries of what is currently possible.

6 CONCLUSION AND FUTURE WORK

Recommender systems have gained significant attention in recent years. However, the effectiveness and security of these systems have also become major concerns, as attackers may attempt to manipulate the recommendations for their own benefit. To promote research in this important field, we introduce RecAD, a new recommender library that provides a variety of benchmark datasets, evaluation settings, attackers, and defense models. By using RecAD, researchers can simulate a range of real-world scenarios and evaluate the robustness of different recommender systems against a variety of potential attacks.

In addition to advancing attacks and defenses on traditional models, we also acknowledge the transformative impact of large language models in the field of recommender systems [3, 40, 50]. Despite their powerful generative capabilities, these models are also susceptible to various attacks [39]. Therefore, our future research will also focus on the development of attack and defense mechanisms specifically tailored to large language model-based recommendations. In order to address this aim, We call upon researchers to collaborate and establish recommender system attack and defense methods that better align with the evolving needs of the field, enhancing the security and robustness of these models.

REFERENCES

[1] 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*. ACM, 348–352.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *Proceedings of the 34th International Conference on Machine Learning*, Doina Precup and Yee Whye Teh (Eds.), Vol. 70. PMLR, 214–223.

[3] Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023. TALLRec: An Effective and Efficient Tuning Framework to Align Large Language Model with Recommendation.

[4] Runa Bhaumik, Chad Williams, Bamshad Mobasher, and Robin Burke. 2006. Securing collaborative filtering against malicious attacks through anomaly detection. In *Proceedings of the 4th workshop on intelligent techniques for web personalization (ITWP'06), Boston*. AAAI, 10.

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

[6] R. Burke, B. Mobasher, R. Bhaumik, and C. Williams. 2005. Segment-based injection attacks against collaborative filtering recommender systems. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*. IEEE, 4 pp.–.

[7] Jie Cao, Zhiang Wu, Bo Mao, and Yanchun Zhang. 2013. Shilling attack detection utilizing semi-supervised learning method for collaborative recommender system. *World Wide Web* 16 (2013), 729–748.

[8] Anahita Davoudi and Mainak Chatterjee. 2017. Detection of profile injection attacks in social recommender systems using outlier analysis. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2714–2719.

[9] Yashar Deldjoo, Tommaso Di Noia, and Felice Antonio Merra. 2019. Assessing the impact of a user-item collaborative attack on class of users. *arXiv preprint arXiv:1908.07968* (2019).

[10] Tong Dou, Junliang Yu, Qingyu Xiong, Min Gao, Yuqi Song, and Qianqi Fang. 2018. Collaborative shilling detection bridging factorization and user embedding. In *Collaborative Computing: Networking, Applications and Worksharing*. Springer, 459–469.

[11] Wenqi Fan, Tyler Derr, Xiangyu Zhao, Yao Ma, Hui Liu, Jianping Wang, Jiliang Tang, and Qing Li. 2021. Attacking black-box recommendations via copying cross-domain user profiles. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1583–1594.

[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*. ACM, 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*. Association for Computing Machinery, New York, NY, USA, 381–392.

[14] 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*. ACM, 381–392.

[15] Chongming Gao, Wenqiang Lei, Jiawei Chen, Shiqi Wang, Xiangnan He, Shijun Li, Biao Li, Yuan Zhang, and Peng Jiang. 2022. Cirs: Bursting filter bubbles by counterfactual interactive recommender system. *arXiv preprint arXiv:2204.01266* (2022).

[16] Jianling Gao, Lingtao Qi, Haiping Huang, and Chao Sha. 2020. Shilling attack detection scheme in collaborative filtering recommendation system based on recurrent neural network. In *Advances in Information and Communication: Proceedings of the 2020 Future of Information and Communication Conference (FICC), Volume 1*. Springer, 634–644.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.

[18] Xiangnan He, Zhankui He, Xiaoyu Du, and Tat-Seng Chua. 2018. Adversarial personalized ranking for recommendation. In *The 41st International ACM SIGIR conference on research & development in information retrieval*. ACM, 355–364.

[19] Hai Huang, Jiaming Mu, Neil Zhenqiang Gong, Qi Li, Bin Liu, and Mingwei Xu. 2021. Data poisoning attacks to deep learning based recommender systems. *arXiv preprint arXiv:2101.02644* (2021).

[20] Zan Huang, Daniel Zeng, and Hsinchun Chen. 2007. A Comparison of Collaborative-Filtering Recommendation Algorithms for E-commerce. *IEEE Intelligent Systems* 22 (2007), 68–78.

[21] Parneet Kaur and Shivani Goel. 2016. Shilling attack models in recommender system. In *ICICT*, Vol. 2. IEEE, 1–5.

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

[23] Bo Li, Yining Wang, Aarti Singh, and Yevgeniy Vorobeychik. 2016. Data poisoning attacks on factorization-based collaborative filtering. *Advances in neural information processing systems* 29 (2016).

[24] Wentao Li, Min Gao, Hua Li, Jun Zeng, Qingyu Xiong, and Sachio Hirokawa. 2016. Shilling attack detection in recommender systems via selecting patterns analysis. *IEICE TRANSACTIONS on Information and Systems* 99, 10 (2016), 2600–2611.

[25] Chen Lin, Si Chen, Hui Li, Yanghua Xiao, Lianyun Li, and Qian Yang. 2020. Attacking recommender systems with augmented user profiles. In *Proceedings of the 29th ACM international conference on information & knowledge management*. ACM, 855–864.

[26] Chen Lin, Si Chen, Meifang Zeng, Sheng Zhang, Min Gao, and Hui Li. 2022. Shilling Black-Box Recommender Systems by Learning to Generate Fake User Profiles. *IEEE Transactions on Neural Networks and Learning Systems* (2022), 1–15.

[27] G. Linden, B. Smith, and J. York. 2003. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing* 7, 1 (2003), 76–80.

[28] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. 2010. Personalized News Recommendation Based on Click Behavior. In *Proceedings of the 15th International Conference on Intelligent User Interfaces*. ACM, 31–40.

[29] Yang Liu, Xianzhuo Xia, Liang Chen, Xiangnan He, Carl Yang, and Zibin Zheng. 2020. Certifiable robustness to discrete adversarial perturbations for factorization machines. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 419–428.

[30] Bhaskar Mehta and Wolfgang Nejdl. 2009. Unsupervised strategies for shilling detection and robust collaborative filtering. *User Modeling and User-Adapted Interaction* 19 (2009), 65–97.

[31] Bamshad Mobasher, Robin Burke, Runa Bhaumik, and Chad Williams. 2007. Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology (TOIT)* 7, 4 (2007), 23–es.

[32] Michael P O'Mahony, Neil J Hurley, and Guénolé CM Silvestre. 2005. Recommender systems: Attack types and strategies. In *Association for the Advancement of Artificial Intelligence (AAAI)*. AAAI, 334–339.

[33] Dazhong Rong, Qinming He, and Jianhai Chen. 2022. Poisoning Deep Learning based Recommender Model in Federated Learning Scenarios. *arXiv preprint arXiv:2204.13594* (2022).

[34] Dazhong Rong, Shuai Ye, Ruoyan Zhao, Hon Ning Yuen, Jianhai Chen, and Qinming He. 2022. Fedrecattack: Model poisoning attack to federated recommendation. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2643–2655.

[35] Behzad Shahrasbi, Venugopal Mani, Apoorv Reddy Arrabothu, Deepthi Sharma, Kannan Achan, and Sushant Kumar. 2020. On Detecting Data Pollution Attacks On Recommender Systems Using Sequential GANs. *CoRR* abs/2012.02509 (2020).

[36] Junshuai Song, Zhao Li, Zehong Hu, Yucheng Wu, Zhenpeng Li, Jian Li, and Jun Gao. 2020. Poisonrec: an adaptive data poisoning framework for attacking black-box recommender systems. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 157–168.

[37] 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.

[38] Jiaxi Tang, Hongyi Wen, and Ke Wang. 2020. Revisiting adversarially learned injection attacks against recommender systems. In *Proceedings of the 14th ACM Conference on Recommender Systems*. ACM, 318–327.

[39] Jiongxiao Wang, Zichen Liu, Keun Hee Park, Muhao Chen, and Chaowei Xiao. 2023. Adversarial Demonstration Attacks on Large Language Models.

[40] Wenjie Wang, Xinyu Lin, Fuli Feng, Xiangnan He, and Tat-Seng Chua. 2023. Generative Recommendation: Towards Next-generation Recommender Paradigm.

[41] Chenwang Wu, Defu Lian, Yong Ge, Zhihao Zhu, Enhong Chen, and Senchao Yuan. 2021. Fight fire with fire: towards robust recommender systems via adversarial poisoning training. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1074–1083.

[42] Zhiang Wu, Jie Cao, Bo Mao, and Youquan Wang. 2011. Semi-SAD: applying semi-supervised learning to shilling attack detection. In *Proceedings of the fifth ACM conference on Recommender systems*. ACM, 289–292.

[43] Xingyu Xing, Wei Meng, Dan Doozan, Alex C. Snoeren, Nick Feamster, and Wenke Lee. 2013. Take This Personally: Pollution Attacks on Personalized Services. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, 671–686.

[44] Fan Yang, Min Gao, Junliang Yu, Yuqi Song, and Xinyi Wang. 2018. Detection of shilling attack based on bayesian model and user embedding. In *2018 IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 639–646.

[45] Guolei Yang, Neil Zhenqiang Gong, and Ying Cai. 2017. Fake Co-visitation Injection Attacks to Recommender Systems. In *Network and Distributed System Security Symposium*.

[46] Jingwei Yi, Fangzhao Wu, Bin Zhu, Yang Yu, Chao Zhang, Guangzhong Sun, and Xing Xie. 2022. UA-FedRec: Untargeted Attack on Federated News Recommendation. *CoRR* abs/2202.06701 (2022).

[47] Hongtao Yu, Haihong Zheng, Yishu Xu, Ru Ma, Dingli Gao, and Fuzhi Zhang. 2021. Detecting group shilling attacks in recommender systems based on maximum dense subtensor mining. In *2021 IEEE International Conference on Artificial Intelligence and Computer Applications*. IEEE, 644–648.

[48] William Zeller and Edward W Felten. 2008. Cross-site request forgeries: Exploitation and prevention. *The New York Times* (2008), 1–13.

[49] Hengtong Zhang, Yaliang Li, Bolin Ding, and Jing Gao. 2020. Practical data poisoning attack against next-item recommendation. In *Proceedings of The Web Conference*. ACM, 2458–2464.

[50] Jizhi Zhang, Keqin Bao, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023. Is ChatGPT Fair for Recommendation? Evaluating Fairness in Large Language Model Recommendation.

[51] Shijie Zhang, Hongzhi Yin, Tong Chen, Zi Huang, Lizhen Cui, and Xiangliang Zhang. 2021. Graph Embedding for Recommendation against Attribute Inference Attacks. In *Proceedings of the Web Conference 2021*. ACM, 3002–3014.

[52] Xudong Zhang, Zan Wang, Jingke Zhao, and Lanjun Wang. 2022. Targeted Data Poisoning Attack on News Recommendation System. *arXiv preprint arXiv:2203.03560* (2022).

[53] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, Yingqian Min, Zhichao Feng, Xinyan Fan, Xu Chen, Pengfei Wang, Wendi Ji, Yaliang Li, Xiaoling Wang, and Ji-Rong Wen. 2021. RecBole: Towards a Unified, Comprehensive and Efficient

Framework for Recommendation Algorithms. In *CIKM*. ACM, 4653–4664.

[54] 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.

[55] Xingchen Zhou, Ming Xu, Yiming Wu, and Ning Zheng. 2021. Deep model poisoning attack on federated learning. *Future Internet* 13, 3 (2021), 73.