Robustness of Deep Recommendation Systems to Untargeted Interaction Perturbations

Sejoon Oh  
soh337@gatech.edu  
Georgia Institute of Technology  
United States

Srijan Kumar  
srijan@gatech.edu  
Georgia Institute of Technology  
United States

ABSTRACT
While deep learning-based sequential recommender systems are widely used in practice, their sensitivity to untargeted training data perturbations is unknown. Untargeted perturbations aim to modify ranked recommendation lists for all users at test time, by inserting imperceptible input perturbations during training time. Existing perturbation methods are mostly targeted attacks optimized to change ranks of target items, but not suitable for untargeted scenarios. In this paper, we develop a novel framework in which user-item training interactions are perturbed in unintentional and adversarial settings. First, through comprehensive experiments on four datasets, we show that four popular recommender models are unstable against even one random perturbation. Second, we establish a cascading effect in which minor manipulations of early training interactions can cause extensive changes to the model and the generated recommendations for all users. Leveraging this effect, we propose an adversarial perturbation method CASPER which identifies and perturbs an interaction that induces the maximal cascading effect. Experimentally, we demonstrate that CASPER reduces the stability of recommendation models the most, compared to several baselines and state-of-the-art methods. Finally, we show the runtime and success of CASPER scale near-linearly with the dataset size and the number of perturbations, respectively.

1 INTRODUCTION
Deep learning-based sequential recommender systems have shown remarkable success in generating accurate recommendations on web platforms [7, 31, 33, 37, 61, 70]. These methods take historical data of user-item interactions as ground truth for training, and use the learned model to generate future recommendations for all users [55]. Ensuring that recommendation models are reliable and have high accuracy is crucial for the success of web platforms.

It is well-known that deep learning models, such as those in computer vision and NLP, can be unreliable under adversarial input data perturbations, i.e., the outputs of the models can change significantly under minor and insignificant changes in the training data [8, 36, 42, 48, 50, 63]. However, the stability of deep sequential recommender systems with respect to untargeted training data perturbations has not been investigated. In untargeted perturbations, minor perturbations in training data change ranked recommendation lists of all users compared to the ranked lists in the absence of perturbations. For instance, as shown in Figure 1, deletion of one training interaction can change the recommendation lists of all users. Stable recommender systems should generate the same ranked lists for all users in the absence and presence of minor perturbations. The ranking of the items in the recommendation list has a huge impact on users’ satisfaction [45]; for example, users engage with the top recommended results several times more than lower-ranked results [1], and users drop out if top recommendations are irrelevant. Thus, if minor perturbations can drastically change the top-K recommendations or the order of items in the ranked list, the reliability of the model and consequently user satisfaction will be dramatically reduced. It is therefore imperative to quantify the stability of deep recommender systems against such perturbations.

Perturbations in training data are practically feasible, both unintentionally and adversarially. Unintentional perturbations are introduced because recommendation models are trained by taking historical user-item interactions as ground truth [55]. However, user actions can be fickle, unreliable, and noisy [5]. For instance, a user U can misclick on video A instead of video B she wants to watch; or, a user V could have clicked on a post P in a social network, but unintentionally did not. Such inadvertent presence of minimal unintentional perturbations should not impact the generated recommendations, but do they? Furthermore, in the worst-case, adversarial perturbations can be done by an attacker (e.g., hacker or company insider) who can access the training data [10, 30, 32]. What is the maximum damage that an attacker can do to the recommender system by introducing minimal adversarial perturbations?

While existing work focuses on targeted attacks [8, 12, 16, 18, 19, 39, 49, 59, 68, 69], and most work is on traditional non-deep recommendation systems [18, 36, 59, 60] with detectable user or item perturbations [4, 15], new methods are needed to study the impact of untargeted perturbations on deep sequential recommenders using minor interaction-level perturbations.

Present work. We investigate the sensitivity of deep sequential recommendation models under minor interaction-level perturbations in the training data. The outcome is the manipulation of all users’ ranked recommendation lists. We define the stability of a
We show that cascading effect holds in deep sequential recommendation systems. Leveraging the cascading effect, we create an adversarial perturbation method called CASPER. We investigate the stability of four popular deep recommendation models, namely, LSTM [26], TSASRec [37], JODIE [33], and LatentCross [7]. We use four standard recommendation datasets to ensure generalizability, namely, LastFM [25], Foursquare [65], Wikipedia [3], and Reddit [2]. We quantify the model stability using three ranklist metrics which compare the changes to all or top-K items in the output ranked lists with and without perturbations. The ranklist metrics are Rank-biased Overlap (or RBO) [57], top-K Jaccard similarity, and average rank difference of items.

To answer the first research question, we investigate the stability of four recommendation models on four datasets. We show that the state-of-the-art models are unstable against random interaction perturbations. Even one random interaction perturbation drastically changes the entire ranked lists of items for all users. This is shown as reduced RBO scores (lower than 1.0) of four recommendation models on the Foursquare dataset in Figure 2(a), on all four datasets (shown later in Figure 3), and as low top-K Jaccard scores (Figure 5).

To answer the second research question, we create an adversarial perturbation method. We first hypothesize a cascading effect between interactions. It states that perturbations made to the training data at earlier timestamps lead to higher changes to the ranked lists during testing, compared to perturbations made at later timestamps. We show that cascading effect holds in deep sequential recommendation systems. Leveraging the cascading effect, we create an interaction-level adversarial perturbation method called CASPER. CASPER is a gray-box method that does not require model parameters or gradients. We compare CASPER with three baselines and two state-of-the-art adversarial attacks. We show experimentally that CASPER reduces the stability of recommendation models the most compared to existing methods across datasets. For instance, results on the LatentCross model (most stable against random perturbations on average) across the datasets are shown in Figure 2(b). CASPER leads to the most drop in performance across datasets.

## 2 RELATED WORK

### Poisoning Attacks and Perturbations for Recommender Systems

As shown in Table 1, most of the recent literature [6, 12, 13, 16, 18, 19, 39, 40, 49, 59, 60, 67–69] have focused on targeted perturbations and poisoning attacks on deep recommender systems, where they aim to increase (or decrease) the rank of target item(s) via perturbing training data, but do not manipulate the entire or top-K ranked lists for all users. Therefore, perturbations generated from existing approaches are not suitable to test the overall stability of deep recommender systems. Furthermore, most of them are not appropriate as baselines because they are not applicable for interaction-level perturbations or work only on multimodal recommenders [6, 13, 40] and matrix factorization-based models [18, 59, 60]. Few existing algorithms, such as CF-attack [36] and RL-attack [8], provide untargeted perturbations for recommender systems that reduce the model’s prediction accuracy significantly. However, those methods do not work on deep sequential recommendation models, focus on attacking the model’s prediction accuracy but not the ranked lists of all users, or are easily detectable as they make user or item perturbations [4, 15].

### Poisoning Attacks and Perturbations in Other Domains

After a pioneering work of Szegedy et al. [52] about the weakness of neural networks against small changes on images, many adversarial perturbation attacks [20, 21, 42, 43, 48, 51] have been introduced for image classification tasks. However, these cannot be directly applied to recommender systems due to sequential data dependency. Recently, data perturbation attacks [9, 34, 54] for natural language processing (NLP) have been proposed. However, we cannot employ them directly for our setting since they either are targeted perturbations, have different perturbation levels (e.g., word or embedding modifications), or cannot model the long sequential dependency.

## 3 MODELS, METRICS, AND DATASETS

In this section, we introduce deep sequential recommendation models, evaluation metrics, and datasets used in the paper.
3.1 Deep Sequential Recommendation Models
We consider a sequential recommendation problem, where recommendation models are trained to learn users’ behavioral patterns from the sequence of their actions. A trained model generates a ranked list of all items in decreasing order of the user’s likelihood to interact with the item next. The top-K items from the ranked list become the recommendation list displayed to the user. We refer to a set of users and items as $U$ and $I$, respectively.

Deep sequential recommendation models use different prediction modules such as recurrent neural networks (RNN) [7, 26, 33, 61, 70] or attention mechanisms [31, 37] to create efficient models. Among them, we investigate the sensitivity of four popular methods that have shown high performance:

• LSTM [26]: given a fixed-length (L) sequence of items, it utilizes Long Short-Term Memory (LSTM) to predict the next item.

• TiSARec [37]: a self-attention based model that takes relative time intervals and absolute positions among items for next-item prediction. This is a recent state-of-the-art model.

• JODIE [33]: a coupled RNN-based graph recommendation model which trains two RNNs to learn user and item embeddings as well as an embedding projection operator to predict the next item.

• LatentCross [7]: a gated recurrent unit (GRU) [11] based model which uses context features, like time difference between interactions. This model is used on YouTube [7].

3.2 Next-item Prediction Metrics
The performance of sequential recommendation models is evaluated using the rank of the ground-truth item among all items in the next-item prediction task. Two metrics are popularly used, namely, Mean Reciprocal Rank (MRR) and Recall@K (typically K=10) [24, 33]. Both metrics lie between 0 and 1, and higher values are better. We refer to these two metrics as next-item metrics as they provide average statistics of the ranks of ground-truth next items only. However, they cannot quantify changes in the entire ranked lists.

3.3 Ranklist Metrics
We define three ranklist metrics that can accurately quantify changes in users’ ranked lists of items. The metrics are computed based on the order and rank of all $|I|$ items (not just the top-K ranked items) to quantify the changes in the ranked lists. The metrics are:

1. Rank-biased Overlap (RBO): RBO [57] is a similarity metric of item ordering between two ranked lists. RBO lies between 0 and 1. Higher RBO means the ordering of items in the two lists is similar. RBO gives more importance to similarities in the top part of two ranked lists, while dissimilarities in the bottom of two ranked lists have less impact on RBO. Since the order of items is highly important in recommendation systems [45], RBO will be the primary metric we will use to quantify changes in the ranked lists.

2. Top-K Jaccard similarity: We use Jaccard similarity (i.e., $Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$) to compute the ratio of common items in the top-K recommendation lists generated with and without perturbations. This metric does not consider the item ordering. The Jaccard score ranges from 0 to 1. A model is stable if its Jaccard score is close to 1. We set $K = 10$ as it is common practice [24, 33].

3. Average rank difference: For a stable model, the ranks of all items in the recommendation list should not change significantly after minor perturbations. Hence, we measure the average absolute difference of ranks of a randomly sampled set of items with and without perturbations. A model is stable if the difference is small.

3.4 Datasets
We use four standard recommendation datasets for this study. These datasets are selected because they come from diverse domains and have been used in numerous previous studies. Table 2 lists the statistics. In each dataset, we filter out users with fewer than 10 interactions.

| Name        | Users | Items | Interactions |
|-------------|-------|-------|--------------|
| LastFM [25] | 980   | 1,000 | 1,293,103    |
| Foursquare [65] | 2,106 | 5,397 | 192,602      |
| Wikipedia [3] | 1,914 | 1,000 | 142,143      |
| Reddit [2]  | 4,675 | 953   | 134,489      |

4 SENSITIVITY OF RECOMMENDATION MODELS TO RANDOM PERTURBATIONS
In this section, we answer the first research question by evaluating the sensitivity (or stability) of deep sequential recommendation models against random perturbations.

Interaction-level Perturbations. We introduce the concept of minimal random perturbations, where exactly one perturbation is applied to training data. Minimal perturbations are equivalent to adding minimal noise to training data. The minimal perturbation setting helps to infer the model stability in the ‘best-case’ scenario—more perturbations will obviously lead to more changes in the output recommendations. To make the perturbations truly minimal, we perturb only one interaction (instead of all interactions of a user or an item) in the entire training data. One random interaction is either deleted, inserted, or the interaction’s item is replaced with another random item.

Experimental setup. We use the first 90% interactions of each user for training the recommendation model, and the rest of the interactions are used for the test, which is a popular setting used in countless existing papers [17, 27, 41, 56]. For each recommendation model, we use the hyperparameters mentioned in their original publications. Other hyperparameters are set as follows: the maximum training epoch is set to 50, a learning rate is set to 0.001, and the size of the embedding dimension is set to 128. For LSTM and TiSARec, the maximum sequence length per user is set to 50. We execute all our experiments on 4 NVIDIA Tesla K80 GPUs. All code was written in Python using the PyTorch library.

Procedure. We train two recommendation models: one with the original data without perturbations and a second one with the perturbed data. Note that both the recommendation models are trained...
with the exact same settings (i.e., same initialization, parameters and hyperparameters, hardware). This is done to specifically tease out the effect of the perturbation (i.e., if there is no perturbation, the trained models and outputs will be identical in every way).

For every test user-item interaction, two lists are generated: a ranked list of items from the original recommendation model (without perturbations) and a ranked list of items from the second model (with perturbations). The two lists of each test interaction are compared using the ranklist metrics. The ranklist metrics across all test interactions are then aggregated. We repeat this process multiple times with different random perturbations (to average the randomness effect), and report average values of ranklist metrics across the different runs.

**Findings.** We present the RBO scores of four recommendation models against random deletion and random replacement perturbations in Figure 3. The ideal score for a perfectly stable model is 1.0, which is when the ranked lists with and without perturbations are identical. However, we observe that all four recommendation models exhibit low RBO scores on all datasets, ranging from 0.75 to 0.95 in most cases, while sometimes dropping below 0.6. Similar drops are observed for the other ranklist metrics, including top-10 Jaccard similarity (see Figure 5) and in the case of random insertion perturbation. Thus, the order of items in the ranked lists and the items in the top-10 recommendation lists change even if only one training data point is perturbed. Please recall that ranklist metrics are measured across all users, illustrating that even though only one perturbation of a user was made, the ranked lists and top-10 recommendations of all users change drastically.

Comparing the four models, the LatentCross model has the highest RBO in most cases against random perturbations, and its RBO score is consistently high. We see similar trends in other ranklist metrics too. This indicates that LatentCross is the most stable model against random perturbations.

**Why are models unstable against minimal perturbations?** Only one interaction over one million interactions (size of the datasets used) is perturbed. Yet, it changes the ranked lists and top-10 recommendation lists of all users. Why is there such a profound effect? This is due to two reasons. First, the slight change in training data leads to changes in the parameters of a trained recommendation model. Since interactions are time-stamped, mini-batches of training data are created and processed in temporally increasing order. Say an interaction in a mini-batch was perturbed. When processing the mini-batch, model parameters will be updated differently during training (compared to when there is no perturbation). The change in model parameters will affect the updates in later mini-batches. The differences will further multiply over multiple epochs. Thus, with perturbations, the final trained model weights will be different from the ones obtained without perturbations, which can result in different ranked lists. Second, recommendation models are trained to accurately predict only the ground-truth next item as high in the ranked list as possible (ideally, rank 1). The models are not trained to optimize the positions of the other items in the ranked list. Together, the two reasons mean that the ordering of all but the ground-truth next item is highly likely to change due to the training data perturbation.

5 MODEL STABILITY AGAINST ADVERSARIAL PERTURBATIONS

In this section, we aim to answer the second research question of quantifying the maximal damage to the model stability that an adversary can do with minimal perturbations. We saw in the previous section that even random perturbations can change output recommendations. So, what is the ‘worst-case’ damage that can be introduced with carefully-selected perturbations?

5.1 Goal and Capabilities of Perturbations

**Goal:** We aim to find minimal perturbations that maximally change the ranked lists of items generated when there are perturbations, compared to when there are no perturbations. As described earlier, the change is defined in terms of the order of items in the ranked lists of all items (and naturally, changes to the top-K recommended items as well). Thus, which perturbations should be made to ensure the maximal damage to the model stability?

**Capabilities:** We assume adversaries have the following capabilities:

1. **Interaction-level perturbation.** Interactions in the training data can be modified, but all users or items cannot be. Interaction perturbation is the smallest perturbation and has lower detectability than user and item perturbations.

2. **Minimal perturbation.** Only one interaction can be perturbed in the entire training data. This is to measure the stability of recommendation models under the lowest possible perturbation. Naturally, more perturbations will result in a greater decrease in stability.

3. **Possible perturbations: deletion, replacement, and insertion.** Three types of fundamental perturbations are allowed: a user-item interaction can be deleted from the training data (deletion perturbation), an item of a user-item interaction can be replaced with another item (replacement perturbation), or a new user-item interaction can be added to the training data (insertion perturbation). Due to space constraints, we highlight results for deletion and replacement perturbations (replacement is a combination of one deletion and one insertion). We focus on these perturbations in this work to gauge the model stability against fundamental perturbations. More complex perturbations, such as a combination of all three types of perturbations, will of course lead to lower stability and will be explored in future work.

4. **Gray-box perturbations.** By definition of training data perturbations, an adversary has access to the training data (which is available in many cases, such as when training data is from social media [2, 3]). The adversary also knows the maximum sequence length of past user actions that the recommendation model takes to make predictions. The adversary does not have access to any...
Table 3: Validating the Cascading Effect Hypothesis: the table shows the average RBO scores of different perturbation strategies. Earliest-Random perturbation produces lower RBO than Random and Latest-Random perturbations (all p-values <0.05).

5.2 Cascading Effect Hypothesis

Here we propose a novel hypothesis that will be the building block of our perturbation method later.

Cascading Effect Hypothesis: Perturbations to interactions with earlier timestamps result in more changes in users’ ranked lists at test time than perturbations to interactions with later timestamps.

Two alternate hypotheses can be developed. First, perturbations on later timestamp interactions can cause more changes in ranked lists during test time. The rationale behind this alternate hypothesis is that interactions with later-timestamp are recent and closer to the test interactions, and thus perturbing them can have a more direct impact on test ranked lists. The second alternate hypothesis states the temporal position of an interaction is irrelevant. This means perturbing early-time-stamp interactions will give results similar to perturbing any random-time-stamp interactions.

Hypothesis Testing. We devise and compare three following perturbations: an Earliest-Random perturbation, where the first interaction of a randomly selected user is perturbed, a Latest-Random perturbation, where the last interaction of a randomly selected user is perturbed, and a Random perturbation, where a random interaction in the training data is perturbed.

We compare the RBO scores of these perturbations on the LastNetCross model (the most stable one as shown in the previous section) and Foursquare dataset (the hardest-to-predict as per next-item metrics). A perturbation method with lower RBO is more successful in changing ranked lists. We perform each perturbation 10 times (randomly perturbing one interaction each time). The resulting RBO score distributions are compared using the Wilcoxon signed-rank test [58].

The results are shown in Table 3. We see that the Earliest-Random perturbation leads to the highest ranklist changes of all users compared to the other two perturbations, as it has the lowest RBO scores in all three types of perturbations (all p-values <0.05). Next, we observe that between Random and Latest-Random, the former has lower RBO.

Why does Earliest-Random perturbation outperform Random and Latest-Random? One may think that Latest-Random perturbation may lead to the most changes in ranked lists of all users as it perturbs a more recent interaction. However, this is not true.

The reason why the cascading effect hypothesis holds is because of how models are trained using time-stamped sequential data. Interactions with low t will be included in earlier mini-batches, which will allow more changes to model parameters. Moreover, if an interaction (u, i, t) from a user u to an item i at timestamp t is perturbed, user u’s and item i’s latent representations (i.e., embeddings) will be updated differently compared to the case when there is no perturbation. After this update, whenever a user u interacts with another item j, the item j’s embedding will be updated differently as well. Similarly, when another user v interacts with an item i (or j), the user v’s embedding will be updated differently too. As a result, embeddings of all items and users, which interact with previously-perturbed users or items, will change. Together, these factors can lead to more changes in ranked lists of test interactions when perturbations are made to low timestamp interactions.

In Latest-Random perturbation, since t is high, there are fewer changes to model parameters and embeddings, and thus, there is less impact on all users’ ranked lists. Meanwhile, since perturbations on early-timestamp interactions are done in Earliest-Random, it changes the model parameters and embeddings more, which changes ranked lists of many users, thus lowering the average RBO.
with the same timestamp. Thus, each node has at most two outgoing edges (one each to common user and item). If the recommendation from early to later timestamp. No edges are present between nodes indicating to Figure 1, to find the best interaction-level perturbation. Blue and red edges indicate user- and item-sharing consecutive interactions, respectively. Green-colored nodes (interactions) show all descendants (including itself) of an interaction $X_i$. CASPER perturbs interaction $X_3$ since it has the largest cascading score $= 8$.

### 5.3 CASPER: Interaction-level Perturbation based on Cascading Effect

Here we answer the question: which interaction should be perturbed to produce the maximal changes on ranked lists of test interactions? A brute-force technique that tests the impact of every interaction is computationally prohibitive. This is because the model needs to be retrained after perturbing every interaction to calculate its impact on RBO. Thus, we leverage the cascading effect hypothesis to propose a new perturbation, named CASPER (Cascade-based Perturbation), to identify the ideal interaction for perturbations.

From the cascading effect hypothesis, we saw that perturbing earlier interactions results in a larger drop in RBO of test interactions. CASPER is based on the fact that perturbing an interaction $(u, i, t)$ will change the embeddings of $u$ and $i$ at time $t$, and the embeddings of other users and items that interact (after time $t$) with previously-perturbed items or users. We define a cascading score of an interaction as the number of training interactions that will be affected by perturbing that interaction. CASPER aims to identify the training interaction which has the highest cascading score.

As shown in Algorithm 1, we create a graph-based technique to approximate an interaction’s cascading score without retraining the model. Specifically, we first construct an interaction directed acyclic graph (IDAG; lines 2-23 in Algorithm 1), where nodes are training interactions and directed edges represent the influence of one interaction on another. The IDAG corresponding to the user sequences shown in Figure 1 is presented in Figure 4. Two nodes in the IDAG connected by a directed edge should either be consecutive interactions of the same user (e.g., $X_1$ and $X_4$) or of the same item (e.g., $X_3$ and $X_2$). A directed edge must follow the temporal order from early to later timestamp. No edges are present between nodes with the same timestamp. Thus, each node has at most two outgoing edges (one each to common user and item). If the recommendation model (such as LSTM or TSASRtc) has a maximum sequence length $L$, the IDAG is constructed only with the latest $L$ interactions of each user.

The cascading score of a node $X_k$ is approximated as the total number of descendants of $X_k$ in the IDAG. Descendants of a node $X_k$ in the IDAG are defined as all the nodes reachable from $X_k$ by following the outgoing edges in the IDAG. For example, in Figure 4, an interaction $X_3$ has 8 descendants (including itself), the highest among all nodes. By definition, a node’s parent will have a higher cascading score than the node itself. Hence, we accelerate the score computation process by calculating the cascading scores of zero in-degree nodes only (lines 24-27 in Algorithm 1). Finally, CASPER perturbs the node with the highest cascading score since it would maximize the cascading effect (line 28 in Algorithm 1).

The IDAG-based approximation approach makes CASPER a gray-box perturbation method, since it needs training data access, but does not need any parameters or gradients of the model [19, 28, 67]; it only needs to know whether a model has a maximum sequence length $L$, so that CASPER constructs the IDAG only with the recent $L$ interactions per user.

We have theoretically and experimentally shown that CASPER scales near-linearly to the dataset size (Section 5.4 and Figure 7(b)). For very large interaction graphs, we can utilize heuristic-based approximations of cascading scores using randomly-sampled interaction graphs. This is expected to make our method more scalable while trading off the perturbation performance.

### 5.4 Complexity Analyses of CASPER

In this subsection, we analyze the time and space complexities of CASPER. Our proposed method is fast, and it scales near-linearly with the number of interactions in the dataset (see Section 6.4).

#### Time complexity

The first step of CASPER is training and testing a deep sequential recommendation model $\Theta$ with original input data, which takes $O(T_\Theta + N_{test} \cdot infer)$, where $N_{test}$ is the number of test interactions, and $O(T_\Theta)$ and $O(infer)$ are training and single inference time complexities of the model $\Theta$. After that, CASPER constructs the IDAG which takes $O(N)$ (a model without a maximum sequence length) or $O(|U||L|)$ (a model with a maximum sequence length $L$), where $N$ and $|U|$ are the number of training interactions and the number of training users, respectively. The next step is computing cascading scores of zero in-degree nodes in the IDAG, which takes $O(ZN)$ (a model without a maximum sequence length) or $O(|Z||U||L|)$ (a model with a maximum sequence length $L$), where $Z$ is the number of zero in-degree nodes in the IDAG. Perturbations of top-$K$ interactions with the highest cascading scores take $O(K)$, which is negligible compared to other steps. Finally, CASPER retrained the model $\Theta$ with perturbed data and computes ranklist metrics, which takes $O(T_\Theta + N_{test}(\text{infer} + |I|))$ since RBO computation can take up to $O(N_{test}|I|)$. The final time complexity of CASPER is $O(T_\Theta + N_{test}(\text{infer} + |I|) + ZN)$ (a model without a maximum sequence length) or $O(T_\Theta + N_{test}(\text{infer} + |I|) + Z|U||L|)$ (a model with a maximum sequence length $L$).

#### Space complexity

The first step of CASPER is training and testing a deep sequential recommendation model $\Theta$ with original input data, which takes $O(M_\Theta + N_{test}|I|)$ space since we need to store original ranked lists for all test interactions, where $O(M_\Theta)$ is the space complexity of training the model $\Theta$, $N_{test}$ is the number of test interaction, and $|I|$ is the number of items. After that, CASPER constructs the IDAG which takes $O(N)$ space (a model without a maximum sequence length) or $O(|U||L|)$ space (a model with a maximum sequence length $L$), where $N$ and $|U|$ are the number of training interactions and the number of training users, respectively. The next step is computing cascading scores of zero in-degree nodes in the IDAG, which takes $O(N)$ space (a model...
without a maximum sequence length) or $O(|U|L)$ space (a model with a maximum sequence length $L$) since we need to perform a breadth-first search in the IDAG. Finally, CASPER retrained the model $\Theta$ with perturbed data and computes ranklist metrics, which takes $O(M_{\theta} + N_{test}|l|)$ space since we also need to store and use new ranked lists for ranklist metrics calculations. The final space complexity of CASPER is $O(M_{\theta} + N_{test}|l| + N)$ (a model without a maximum sequence length) or $O(M_{\theta} + N_{test}|l| + |U|L)$ (a model with a maximum sequence length $L$).

6 EXPERIMENTAL EVALUATION OF CASPER

In this section, we evaluate our proposed perturbation method, CASPER, with respect to the following aspects.

1. Effectiveness of Perturbation Methods (Section 6.2). How sensitive are the popular deep sequential recommender systems against our proposed perturbation method and baselines? Which perturbation is the best with respect to the ranklist metrics?

2. Impact of the Number of Perturbations (Section 6.3). Is the performance of CASPER proportional to the number of adversarial perturbations allowed on the dataset?

3. Running Time Analysis (Section 6.4). How scalable is CASPER with respect to the number of interactions in a dataset?

4. Impact on Next-Item Performance (Section 6.5). What is the impact of CASPER on the next-item prediction performance of the recommendation models?

6.1 Experimental Settings

6.1.1 Datasets. We use the four standard datasets introduced in Section 4, namely, LastFM, Foursquare, Wikipedia, and Reddit. We emphasize that the datasets are diverse, standard, and popular in the recommendation community. LastFM is a widely used recommendation benchmark dataset [22, 29, 35, 47], Foursquare is broadly utilized for point-of-interest recommendation tasks [62, 64–66], and Wikipedia and Reddit are popular for social network recommendation tasks [14, 33, 38, 44]. We select these datasets for experiments because (a) they come from diverse domains, thus ensuring generalizability; (b) the timestamps of interactions reflect when the corresponding activities happened (as opposed to Amazon review datasets where a review is posted much after a product is purchased, or MovieLens review dataset where a review is posted much after a movie is watched); and (c) these are popularly used datasets in numerous papers as listed above.

6.1.2 Baseline Methods. To the best of our knowledge, there are no interaction-level perturbation methods for deep sequential recommendation models. Therefore, we create strong baselines and two state-of-the-art methods based on the broader literature as follows:

- Random perturbation: This method randomly chooses an interaction for perturbation among all training interactions.
- Earliest-Random perturbation: This method randomly chooses an interaction for perturbation among the first interactions of all users in the training data.
- Latest-Random perturbation: This method randomly chooses an interaction for perturbation among the last interactions of all users in the training data.
- TracIn [46] perturbation: This method chooses the most important training interaction for perturbation, defined in terms of reducing the model’s loss during training. We use an influence estimator TracIn [46] that utilizes loss gradients from the model saved at every $T$ epoch to compute interaction importance. TracIn has been used to conduct adversarial attacks on models [23].

- Rev.Adv. [53] perturbation: This method is the state-of-the-art data poisoning attack that inserts a fake user with interactions crafted via a bi-level optimization problem. To adapt it for our deletion and replacement perturbation settings, we first find the most similar user in the training data to the fake user, and perform deletion or item replacement of the earliest or random interaction of that user, respectively. Therefore, we create two versions of Rev.Adv. – Rev.Adv. [53] (random) and Rev.Adv. [53] (earliest), which indicates the method chooses a random or earliest interaction of a user for perturbation, respectively.

Note that we do not include baselines that work only on multimodal recommenders [6, 13, 40] and matrix factorization-based models [18, 59, 60] as these are not applicable to our setting. We have also not included baselines that have shown similar or worse performance [49, 67, 68] compared to the above baselines, particularly compared to Rev.Adv. [53].

Once a method identifies an interaction to perturb, the perturbation is done as follows: the interaction can be deleted (deletion perturbation), its item can be replaced with another item (replacement perturbation), or another interaction of the same user can be injected before it (insertion perturbation). In replacement and insertion perturbations, the new item can be selected using three different strategies: selecting an item randomly, selecting the most popular item, or selecting the least popular (i.e., unpopular) item.

It is important to restate that the recommendation models are trained in the exact same setting during perturbations (i.e., same initializations, hyperparameters, etc.). Thus, if there is no perturbation, the model will produce the same ranked lists in every run. This setting is adopted to only measure the effect of different perturbations on the output ranked lists.

6.1.3 Deep Sequential Recommendation Models. We use the popular deep sequential recommendation algorithms: LSTM [26], TSiSASRec [37], JODIE [33], and LATENTCROSS [7] (their descriptions were given in Section 3.1) to test the effectiveness of our proposed perturbation method and baselines.

6.1.4 Experimental Setup. We follow the same experimental setup, as described previously in Section 4, which was used to test the model stability. Additionally, we use the following settings. We repeat all experiments multiple times and report average values of ranklist metrics. We use ground-truth test items as a random set of items while computing the average rank difference. To construct the IDAG for CASPER, we use all the interactions in JODIE and LATENTCROSS. For LSTM and TSiSASRec, we use the latest 50 interactions per user, as defined by the maximum sequence length in the original papers. To compute the influence of interactions in the TracIn perturbation, we take training loss gradients with respect to the last hidden layer. We save the loss gradients every 10 epochs and fix step sizes to the default learning rate of 0.001.
6.2 Effectiveness of Perturbations

Perturbations on the Foursquare dataset. Table 4 compares the performance of all perturbation methods on all four recommendation models and Foursquare dataset (the hardest-to-predict in terms of next-item metrics), averaged over 3 repetitions. Each column highlights the best and second-best perturbation model, in terms of the lowest RBO score.

First, we observe that the recommendation models are unstable against both deletion and replacement perturbations performed by all the methods. The RBO scores of all the recommendation models drop significantly below 1.0, indicating their low stability. Next, across all settings, our proposed method, CASPER, achieves the best performance across all but one setting, in which it performs the second best. It leads to the most reduction of RBO in most cases. We observe that CASPER is more effective on JODIE and LATENTCROSS models, since the other two models (LSTM and TiSASRec) have the maximum sequence lengths, which limit their interactions’ cascading effects. Similar observations hold with the average rank difference metric and top-K Jaccard score. It is also worth mentioning that Rev.Adv. (earliest) outperforms Rev.Adv. (random) in most cases, which substantiates the cascading hypothesis we introduced.

In item replacement perturbation (Table 4(b)), CASPER outperforms other methods in all cases. For CASPER, we find that replacing the item with the least popular item is the most effective strategy among all the others. One possible reason is that the change in user embeddings and model parameters by using an unpopular item will be the highest. Injection of the unpopular item diversifies the user’s interactions and embedding the most, and model parameters can be updated most differently with this heterogeneous item. This major update will cascade to later interactions and change all users’ recommendations drastically.

Deletion perturbation performance measured with top-10 Jaccard score. Figure 5 shows the top-10 Jaccard scores of CASPER and baselines on the LATENTCROSS model and Foursquare dataset for the deletion setting. All Jaccard scores are much lower than 1.0, which indicates all perturbations successfully change the top-10 recommendation lists of LATENTCROSS. CASPER shows the best perturbation performance (the lowest Jaccard score) among all methods. These indicate that one perturbation changes the top-10 recommendations for all users to a large extent. The changes in the top-10 lists can drastically reduce the model reliability and user trust. Similar trends are observed in other perturbations (the insertion case is in Table 5).

Insertion perturbation. Table 5 presents the insertion perturbation performance of CASPER and baselines on LATENTCROSS model and Foursquare dataset. To introduce the perturbation, in all methods except Rev.Adv. [53], we insert a new interaction (with the same user and a new item) right before the interaction chosen by each perturbation method. In Rev.Adv. [53], we insert a fake user with one interaction chosen by the optimization. CASPER with the least popular item insertion shows the best perturbation performance as per all ranklist metrics. Insertion perturbation results on different recommendation models and datasets present similar trends to those of item replacement perturbations. Again, we validate the cascading effect as the insertion perturbation result of Rev.Adv. [53] (earliest) has superior performance than Rev.Adv. [53] (random).
Robustness of Deep Recommendation Systems to Untargeted Interaction Perturbations

Figure 6: Deletion and item replacement perturbations on the LatentCross model. CASPER shows the best performance on all four datasets.

Figure 7: (a) Perturbation scalability and (b) runtime of CASPER.

Figure 8: Impact of Perturbation on Next-Item Prediction. The differences in metrics with and without perturbations are marginal. Importantly, CASPER outperforms all baselines across all datasets on the LatentCross model. For example, on Foursquare, CASPER offers at least 10.0% performance improvements compared to baseline perturbations in terms of RBO and at least 43.9% in average rank difference.

We note that the relative order of the performance of perturbation methods is expected, as shown in Table 4 and Figure 6. Across all datasets, Latest-Random baseline performs worse than Random perturbation, which performs worse than the Earliest-Random, all due to the cascading effect. Similarly, all random perturbations are expected to have worse performance than advanced perturbation methods like Rev.Adv. [53] and CASPER.

6.3 Impact of the Number of Perturbations

Intuitively, more perturbations in training data will cause more damage. To test the effect of the number of perturbations on CASPER, we increase the number of perturbations from 1 to 8 and check its deletion perturbation performance on the LatentCross model and Foursquare dataset. CASPER selects the $k$ interactions with the highest cascading score when the number of perturbations is $k$. As shown in Figure 7(a), the perturbation performance of CASPER scales near-linearly with the number of perturbations. We omit results of replacement and insertion as they show similar trends.

6.4 Running Time Analysis

We vary the number of interactions in a dataset to test whether CASPER is scalable to the input data size. Specifically, we measure the running time of a deletion perturbation of CASPER on the LatentCross model and LastFM dataset (the largest one), while varying the number of interactions in the dataset from 10,000 to 1,000,000. Figure 7(b) shows CASPER scales near-linearly with the dataset size. This empirically validates the time complexity of CASPER (see Section 5.4 for details), which is linear in terms of the total number of interactions. Replacement and insertion perturbation versions of CASPER have similar trends.

6.5 Impact on Next-Item Prediction

Here we evaluate the impact of perturbations on the next-item prediction performance. Figure 8 shows next-item metrics of various recommendation models against the deletion perturbation performed by our proposed method CASPER on the Wikipedia dataset. All next-item metrics indeed change minimally without versus with the perturbation. For instance, the maximum change in MRR for the CASPER perturbation is only around 0.002. This is because, after perturbation, the models are retrained to predict the next item. Since the perturbations are minimal, the model can learn to predict the next item like when there is no perturbation. This is dangerous because deploying a model, solely based on its MRR and Recall performance, will lead to unstable recommendations. One may feel that if the MRR and Recall@10 scores do not change, then the recommendation models are stable. However, this is not the case since MRR and Recall@10 only narrowly view the ground-truth item. We have already seen that the top-recommended items and their ordering change dramatically (as seen using ranklist metrics). Since the items and their ordering in the top-K recommendations can impact user behavior [1, 45], it is of utmost importance to ensure that all the top-ranked items are high-quality and not manipulatable by adversarial perturbations. User satisfaction will suffer regardless of the rank of the ground-truth next item, if the top-10 recommendations are filled with irrelevant items and their ordering is spurious and manipulatable. The model will be unstable and untrustworthy.

7 CONCLUSION

In this paper, we established the vulnerability of recommendation models against minor input data perturbations. Adversarial
methods, such as CASPER, can further exacerbate the model shortcoming. This illustrates the need to create the next-generation of recommendation models that are reliable and robust.

This work has some shortcomings and future work opportunities. First, our work considered deletion, insertion, and replacement perturbations, with the goal of assessing the model sensitivity to minor perturbations. More complex manipulations, such as jointly using all three types of perturbations, can be explored in the future. Second, opportunities are ripe to develop defense mechanisms to improve the model stability against such scenarios. Finally, to make perturbation methods more scalable to very large interaction graphs, one can explore creating approximations of cascading scores using randomly-sampled interaction graphs, instead of using the entire graph. This is expected to make perturbations more scalable while trading off their performance.

REFERENCES

[1] 2020. Over 25% of People Click the First Google Search Result. https://www.searchenginejournal.com/google-first-page-clicks/374516/.
[2] 2020. Reddit data dump. http://files.pushshift.io/reddit/.
[3] 2020. Wikipedia edit history dump. https://meta.wikimedia.org/wiki/Data_dumps.
[4] Mehrmet Aktukmak, Yasin Yilmaz, and Ismail Uysal. 2019. Quick and accurate attack detection in recommender systems through user attributes. In Proceedings of the 13th ACM Conference on Recommender Systems. 348–352.
[5] Xavier Amatriain, Josep M Pujol, and Nuria Oliver. 2009. I like it... i like it too. In Proceedings of the 25th ACM Conference on Hypertext and Hypermedia. 172–181.
[6] Vito Walter Anelli, Yashar Deldjoo, Tommaso Di Noia, Daniele Malitesta, and Felice Antonio Merra. 2021. A study of defensive methods to protect visual attack detection in recommender systems through user attributes. In Proceedings of the 15th ACM Conference on Recommender Systems. 348–352.
[7] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining. 46–54.
[8] Yuanjiang Cao, Xiaocong Chen, Lina Yao, Xianzhi Wang, and Wei Emma Zhang. 2020. Adversarial attacks and detection on reinforcement learning-based interactive recommender systems. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. 1669–1672.
[9] Alvin Chan, Yi Tay, Yew-Soon Ong, and Aston Zhang. 2020. Poison Attacks against Text Datasets with Conditional Adversarially Regularized Autoencoder. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing. 4175–4189.
[10] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplov Sivastava. 2018. Detecting backdoor attacks on deep neural networks by activation clustering. arXiv:1811.03728 (2018).
[11] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Embeddings for Statistical Machine Translation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 1724–1734.
[12] Konstantina Christakopoulou and Arindam Banerjee. 2019. Adversarial attacks on an oblivious recommender. In Proceedings of the 15th ACM Conference on Recommender Systems. 322–330.
[13] Rami Cohen, Oren Sar Shalom, Dietmar Jannach, and Amihood Amir. 2021. A Black-Box Attack Model for Visually-Aware Recommender Systems. In Proceedings of the 14th ACM International Conference on Web Search and Data Mining. 94–102.
[14] Hanjun Dai, Yichen Wang, Rafshat Trivedi, and Le Song. 2016. Deep coevolutionary network: Embedding user and item features for recommendation. arXiv preprint arXiv:1609.03675 (2016).
[15] Anahita Davoudi and Maimak 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.
[16] Tommaso Di Noia, Daniele Malitesta, and Felice Antonio Merra. 2020. Taamr: Targeted adversarial attack against multimedia recommender systems. In 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 1–8.
Robustness of Deep Recommendation Systems to Untargeted Interaction Perturbations

[41] Zaiaqo Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2020. Exploring data splitting strategies for the evaluation of recommendation models. In Fourteenth ACM Conference on Recommender Systems. 681–686.

[42] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. 2017. Universal adversarial perturbations. In Proceedings of the IEEE conference on computer vision and pattern recognition. 1765–1773.

[43] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 2016. Deepfool: a simple and accurate method to fool deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition. 2574–2582.

[44] Shalini Pandey, George Karpyis, and Jaideep Srivastava. 2021. IACN: Influence-Aware and Attention-Based Co-evolutionary Network for Recommendation. In Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 561–574.

[45] Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Sunfeng Ge, Wenwu Ou, et al. 2019. Personalized re-ranking for recommendation. In Proceedings of the 13th ACM Conference on Recommender Systems. 3–11.

[46] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. 2020. Estimating Training Data Influence by Tracing Gradient Descent. Advances in Neural Information Processing Systems 33 (2020).

[47] Pengjie Song, Zhao Li, Zehong Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Ozkasap. 2020. Certified defenses for data poisoning attacks. In Proceedings of the 31st International Conference on Neural Information Processing Systems. 3520–3532.

[48] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. IEEE Transactions on Evolutionary Computation 23, 5 (2019), 828–841.

[49] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. Computation 28, 4 (2010), 1245–1254. https://doi.org/10.1109/ICML.2010.19

[50] Eric Wallace, Tony Zhao, Shi Feng, and Sameer Singh. 2021. Conceived Data Poisoning Attacks on NLP Models. (June 2021), 139–150. https://doi.org/10.18653/v1/2021.naacl-main.13

[51] Shoujin Wang, Guo Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. 2014. Intriguing properties of neural networks. In International Conference on Learning Representations. http://arxiv.org/abs/1312.6199

[52] Jiuxiang Sun, Zhonghao Li, Zhehong Hu, Yucheng Wu, Zhengpeng Li, and Jian Li. 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.

[53] Pengjie Song, Zhao Li, Zehong Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Ozkasap. 2020. Certified defenses for data poisoning attacks. In Proceedings of the 31st International Conference on Neural Information Processing Systems. 3520–3532.

[54] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. IEEE Transactions on Evolutionary Computation 23, 5 (2019), 828–841.

[55] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. Computation 28, 4 (2010), 1245–1254. https://doi.org/10.1109/ICML.2010.19

[56] Eric Wallace, Tony Zhao, Shi Feng, and Sameer Singh. 2021. Conceived Data Poisoning Attacks on NLP Models. (June 2021), 139–150. https://doi.org/10.18653/v1/2021.naacl-main.13

[57] Shoujin Wang, Guo Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. 2014. Intriguing properties of neural networks. In International Conference on Learning Representations. http://arxiv.org/abs/1312.6199

[58] Jiuxiang Sun, Zhonghao Li, Zhehong Hu, Yucheng Wu, Zhengpeng Li, and Jian Li. 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.

[59] Pengjie Song, Zhao Li, Zehong Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Ozkasap. 2020. Certified defenses for data poisoning attacks. In Proceedings of the 31st International Conference on Neural Information Processing Systems. 3520–3532.

[60] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. 2019. One pixel attack for fooling deep neural networks. IEEE Transactions on Evolutionary Computation 23, 5 (2019), 828–841.