Rank List Sensitivity of Recommender Systems to Interaction Perturbations

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

Berk Ustun
berk@ucsd.edu
University of California San Diego
United States

Julian McAuley
jmcauley@eng.ucsd.edu
University of California San Diego
United States

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

ABSTRACT
Prediction models can exhibit sensitivity with respect to training data: small changes in the training data can produce models that assign conflicting predictions to individual data points during test time. In this work, we study this sensitivity in recommender systems, where users’ recommendations are drastically altered by minor perturbations in other unrelated users’ interactions. We introduce a measure of stability for recommender systems, called Rank List Sensitivity (RLS), which measures how rank lists generated by a given recommender system at test time change as a result of a perturbation in the training data. We develop a method, CASPER, which uses cascading effect to identify the minimal and systematic perturbation to induce higher instability in a recommender system. Experiments on four datasets show that recommender models are overly sensitive to minor perturbations introduced randomly or via CASPER — even perturbing one random interaction of one user drastically changes the recommendation lists of all users. Importantly, with CASPER perturbation, the models generate more unstable recommendations for low-accuracy users (i.e., those who receive low-quality recommendations) than high-accuracy ones.

CCS CONCEPTS
• Information systems → Recommender systems.

KEYWORDS
Recommender Systems, Model Stability, Input Data Perturbation

1 INTRODUCTION
Small changes in training data can produce large changes in outputs of machine learning models [7, 40, 47, 57, 59, 75]. Marx et al. [45] showed how classification tasks can often admit competing models that perform almost equally well in terms of an aggregate performance metric (e.g., error rate, AUC) but that assign conflicting predictions to individual data points. Likewise, Black and Fredrikson [6] showed how removing a point from a training dataset can produce models that assign drastically different predictions, and highlighted how this lack of stability disproportionately affects points with low confidence predictions.

The sensitivity with respect to minor data perturbations is especially meaningful and concerning in modern recommender systems – where data points pertain to user interactions. In this setting, sensitivity would imply that the recommendations for a user change due to small arbitrary changes in the training data from another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70]. In this setting, sensitivity would imply that the treatment recommendations for a patient by a given system could change due to noisy training data for another unrelated user. This effect can be disruptive or even dangerous when recommendation systems are used for applications in healthcare, finance, education, and housing [62, 64, 79, 85]. Consider a system that recommends a specific treatment to a patient based on data from their electronic health record [64, 70].
low-quality recommendations for all users or even disproportionate damage on specific user or item groups [16, 18, 19, 58, 81].

These effects broadly underscore the need to measure the sensitivity in the development of recommender systems – so that model developers and end users can decide whether to use a specific recommendation, or whether to use recommender systems at all.

In this work, we study the sensitivity of recommender systems, so that practitioners can measure the stability of their models and make informed decisions in model development and deployment. Our problem statement is: Can an arbitrary change in a single data point in the training data change the recommendations for other data points? If so, what is the maximum change in recommendations possible with that change?

We propose a novel framework to measure the stability by comparing two recommendation lists for each test interaction – the recommendation list from a recommender model trained on the original training data, and the recommendation list from a model trained on the perturbed training data. Then, the two recommendation lists are compared for each test interaction as shown in Figure 1. If the two lists are the same, then we say that the model is stable to the perturbation; otherwise, the model is unstable.

Our approach requires a metric to differentiate the order of items between two lists. Standard next-item prediction metrics such as MRR, Recall, NDCG, and AUC are applicable to one list (by measuring the rank of the ground-truth next item in the list). Extensions of these metrics, e.g., the difference in MRR, are not appropriate since those metrics can remain unchanged even if the rank list is drastically different, but the ground-truth item’s rank remains similar. This happens in practice (see Figure 3(b)). Thus, we introduce a formal metric to quantify the stability of recommender systems, namely Rank List Sensitivity (RLS), which measures the similarity in the rank lists generated in the presence versus absence of perturbations. We employ two metrics to measure RLS, namely Rank-Biased Overlap (RBO) [69] and Jaccard Similarity [30]. RBO measures similarity in the order of items in two lists, while the Jaccard score highlights the overlap in the top-K items without considering their order. Higher scores in both metrics are better.

We introduce two training data perturbation methods to measure the stability of recommender systems: random perturbations and CASPER perturbations. Random perturbations select one interaction out of all training data interactions randomly for perturbations.

Using random perturbations, we can measure the model sensitivity caused due to arbitrary errors and noise. On the other hand, CASPER is designed to identify an interaction whose perturbation can introduce higher instability in recommendations than random perturbations. Such interaction reveals model vulnerabilities that can potentially be exploited by adversaries to manipulate the recommender system. To find the deliberate perturbation, we hypothesize a cascading effect by creating an interaction-to-interaction dependency graph. Then, CASPER perturbs an interaction with the largest number of descendants in the graph, which leads to significant changes in the generated recommendations. CASPER is fast and scalable to the dataset size and does not require model parameters or gradients to identify the perturbation.

Experimentally, we first investigate the sensitivity of models to random perturbations. We show that the recommender models are sensitive to random interaction perturbations. Even one random interaction perturbation drastically changes the entire rank lists of items for all users. This is shown as low RBO scores (lower than 1.0 score means the rank list has changed) of four recommendation models on Foursquare (Figure 2(a)) and all four datasets (shown later in Figure 3), and as low top-10 Jaccard scores (Figure 7). We underline that the instability of the models occurs due to the data change, not the training randomness (e.g., different random seed, initialization, etc.), since we remove all the randomness during the training to focus solely on the effect of training data perturbation.

Next, we compare CASPER with five training data perturbation algorithms. We show that CASPER identifies a perturbation to be made that reveals higher sensitivity in recommendation models compared to existing methods across datasets. Importantly, we find that CASPER identifies an interaction whose perturbation results in low-accuracy user groups being more impacted as per model sensitivity – the top-10 Jaccard scores are lower for low-MRR users than for high-MRR users (see Figure 2(b)). Since the item ranking in the recommendation list has a significant impact on user satisfaction [53], if the recommendation is low-quality and unstable, the user satisfaction and engagement can be dramatically reduced, and it may result in user dropout. We provide the repository of our dataset and code used in the paper for reproducibility.

### 2 RELATED WORK

**Data Perturbation in Recommender Systems.** Our work is broadly related to a stream of research on perturbations in deep recommender systems [4, 10, 11, 16, 18, 19, 43, 44, 58, 72, 73, 80, 82, 83]. Much of this work has generated perturbations that alter the
rank of target item(s) (see Table 1). These methods highlight the vulnerability of specific recommendations. However, they provide incomplete stability since they focus on specific target items, rather than the entire or top-K rank lists for all users. Furthermore, they are not appropriate as baselines because they are not applicable for interaction-level perturbations or work only on multimodal recommenders [4, 11, 44] and matrix factorization-based models [18, 72, 73]. Some CF-based [40] and RL-based [7] recommender systems, provide untargeted perturbations for recommender systems that reduce the model’s prediction accuracy considerably. However, those methods do not work on our perturbation setting since they provide user- or item-level perturbations instead of interaction-level or focus on degrading the model’s prediction accuracy without altering the rank lists of all users.

Data Perturbation in Other Domains. Many input perturbation methods [20, 21, 47, 48, 57, 60, 84] have been developed for image classification. These methods cannot be directly applied to recommender systems due to complexities of sequential data (e.g., discrete input data and temporal dependency between interactions). Many data perturbation algorithms [8, 24, 37, 49, 66] for natural language processing (NLP) have been proposed. 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 long sequential dependencies.

Stability & Multiplicity in Machine Learning. Our work is also related to a stream of work on stability and multiplicity in machine learning [3, 6, 12, 15, 27, 38, 45, 52, 56, 68]. Recent work in this area has shown that datasets can admit multiple nearly-optimal solutions that exhibit considerable differences in other desirable characteristics (e.g., predictions on specific data points, behavior to model shifts, counterfactual explanations). For instance, Black and Fredrikson [6] study data multiplicity caused by inserting or removing a single user (“leave-one-out”) on several ML models. Marx et al. [45] demonstrate the potential fairness issue in recidivism prediction problems. While the majority of papers focus on the model multiplicity in classification models, they do not study the stability in recommender systems caused by data perturbations.

3 PRELIMINARIES

We consider a sequential recommendation task, where a recommender model \( M: X \rightarrow R^X_M \) is trained to learn users’ behavioral patterns from a sequence of their actions. A trained model \( M \) generates a rank list of all items \( R^X_M \) that a user may interact with given a test interaction \( X_k \in X_{test} \). Items are ordered in terms of the likelihood of user interaction, and the system shows the top-K items from the rank list \( R^X_M \) to each user. We denote the set of users and items as \( U \) and \( I \), respectively. We study the sensitivity of four methods to train a sequential recommendation model:

- **LSTM** [26]: given a sequence of items, it predicts the next item via Long Short-Term Memory (LSTM).
- **TiSASRec** [41]: a recent self-attention based model that predicts the next item using the relative time intervals and absolute positions among previous items.
- **JODIE** [36]: a coupled RNN-based recommendation model which predicts the next item via RNNs to learn user and item embeddings.

---

### Table 2: Recommendation datasets used in Sections 5, 6, 7.

<table>
<thead>
<tr>
<th>Name</th>
<th>Users</th>
<th>Items</th>
<th>Interactions</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LastFM</td>
<td>980</td>
<td>1,000</td>
<td>1,293,103</td>
<td>Music playing history</td>
</tr>
<tr>
<td>Foursquare</td>
<td>2,106</td>
<td>5,597</td>
<td>192,602</td>
<td>Point-of-Interest check-in</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>1,914</td>
<td>1,000</td>
<td>142,143</td>
<td>Wikipedia page edit history</td>
</tr>
<tr>
<td>Reddit</td>
<td>4,675</td>
<td>953</td>
<td>134,489</td>
<td>Subreddit posting history</td>
</tr>
</tbody>
</table>

- **LATENTCROSS** [5]: a gated recurrent unit (GRU) [9] based model which uses contextual features, like time difference between interactions. This model is used in YouTube [5].

3.1 Datasets

We use four recommendation datasets from diverse domains summarized in Table 2. In each dataset, we filter out users with fewer than 10 interactions.

- LastFM [22, 25, 31, 39, 55] includes the music playing history of users represented as \( \text{user, music, timestamp} \).
- Foursquare [74, 76–78] is a point-of-interest dataset represented as \( \text{user, location, timestamp} \).
- Wikipedia [2, 14, 35, 36, 42, 50, 51] contains the edit records of Wikipedia pages represented as \( \text{user, page, timestamp} \).
- Reddit [1, 14, 34, 36, 42, 51] includes the posting history of users on subreddits represented as \( \text{user, subreddit, timestamp} \).

3.2 Next-Item Prediction Metrics

The dataset-level performance of a sequential recommendation model is evaluated in a next-item prediction task by calculating the rank of the ground-truth item among all items, averaged over all test interactions. Two metrics are widely used: (i) Mean Reciprocal Rank (MRR) [65]; (ii) Recall@K (typically K=10) [23, 36]. 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.

4 MEASURING RANK LIST SENSITIVITY

We create a framework to measure the stability of recommendation systems against perturbations.

**Procedure.** First, we train a recommendation model \( M \) with the original data without perturbations, and it generates one ranked recommendation list \( R^X_M \) for each test interaction \( X_k \) in test data \( X_{test} \), where \( k \) indicates an index of an interaction. Second, we train another recommendation model \( M' \) with perturbed training data, and it generates ranked recommendation lists \( R^X_{M'} \) \( \forall X_k \in X_{test} \).

We measure the similarity of recommendations for each test example \( X_k \) by comparing the two recommendation lists \( R^X_M \) and \( R^X_{M'} \). We devise Rank List Sensitivity (RLS) metrics to measure the similarity (described in the next paragraph). Then, we average the individual RLS score across all \( X_k \in X_{test} \). If the model is perfectly stable, then \( R^X_M \) and \( R^X_{M'} \) should be identical \( \forall X_k \in X_{test} \), and the average RLS value should be maximized.

We repeat the above process multiple times with different random seeds (to average the impact of individual experiments), and report average values of RLS metrics across different runs. The average RLS value quantifies the model stability.

**Rank List Sensitivity Metrics.** To measure the stability of a recommendation model \( M \), we need metrics that can compare the
similarity between recommendation lists generated with versus without perturbations, i.e., \( R_{X_M}^k \) versus \( R_{X_{M'}}^k \). Standard next-item prediction metrics (described in Section 3.2) only measure the rank of the ground-truth next item in one recommendation list. Extensions of these metrics, e.g., the difference in MRR or Recall, to measure similarity are not appropriate since these metrics can remain unchanged if the ground-truth item’s rank is the same in \( R_{X_M}^k \) and \( R_{X_{M'}}^k \), even though the positions of the other items in the two rank lists are drastically different. This happens in practice – see Figure 3(b), where the difference between MRR and Recall values of recommendation models \( M \) and \( M' \) are almost identical. To compute the list similarity accurately, we need to measure how a perturbation impacts the order of all items across two recommendation lists. Thus, we need metrics that are sensitive to differences in the positions of all items, not only the ground-truth item.

We introduce a formal metric called Rank List Sensitivity (RLS) to quantify the stability of recommender systems by comparing the items and their ranking in two lists (or two top-K lists). Mathematically, RLS metrics of a model \( M \) against input perturbation are defined by the following:

\[
RLS = \frac{1}{|X_{test}|} \sum_{X_k \in X_{test}} \text{sim}(R_{X_M}^k, R_{X_{M'}}^k)
\]

where \( \text{sim}(A, B) \) is a similarity function between two rank lists \( A \) and \( B \). We use the following two similarity functions in this paper. (1) **RBO (Rank-biased Overlap):** RBO [69] measures the similarity of orderings between two rank lists \( R_{X_M}^k \) and \( R_{X_{M'}}^k \). RBO lies between 0 and 1. Higher RBO means the ordering of items in the two lists is similar. For reference, the RBO between two randomly-shuffled rank lists is approximately 0.5. RBO is more responsive to similarities in the top part of two rank lists, meaning that it imposes higher weights on the top-K items. This property distinguishes RBO from other measures like Kendall’s Tau [32]. RBO of two rank lists \( A \) and \( B \) with \(|I|\) items is defined as follows.

\[
RBO(A, B) = (1 - p) \sum_{d=1}^{|I|} p^{d-1} \frac{|A[1:d] \cap B[1:d]|}{d}
\]

where \( p \) is a tunable parameter (recommended value: 0.9).

(2) **Top-K Jaccard similarity:** The Jaccard similarity [30]

\[
\text{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}
\]

is a normalized measure of similarity of the contents of two sets \( A \) and \( B \). We use it to measure the similarity of items in the top-K recommendation lists generated with and without perturbations, i.e., \( R_{X_M}^k \{1:k\} \) and \( R_{X_{M'}}^k \{1:k\} \). The Jaccard score ranges from 0 to 1, and is agnostic to the ordering of items. A model is stable if its Jaccard score is close to 1. In all experiments, we set \( K = 10 \) to compare the top-10 recommendations [23, 36].

Top-K Jaccard metric can be useful for the industry due to its fast computation compared to RBO; RBO can be used for detailed analyses of the model stability since it focuses on full ranked lists.

---

**Figure 3:** (a) Stability of four recommendation models against random LOO perturbation. Existing models exhibit unstable predictions since RBO scores after the perturbation are low. (b) Impact of random LOO perturbation on next-item predictions of recommendation models. The differences in metrics with and without perturbations are marginal.

### 5 STABILITY AGAINST RANDOM PERTURBATIONS

In this section, we investigate the stability of recommendation models against random perturbations.

**Interaction-level Perturbations.** We measure the stability of a model with respect to arbitrary errors and noise through minimal random perturbations. These perturbations change one randomly-chosen sample in the training data – i.e., an interaction of a single user rather than all interactions of a user or an item. In particular, an interaction is either deleted (leave-one-out), inserted, or the interaction’s item is replaced with another random item.

**Experimental Setup.** Our goal is to test the stability of diverse recommendation models against a random interaction perturbation. We use the first 90% of interactions of each user for training the recommendation model, and the rest are used for testing, which is a common setting used in several papers [17, 28, 46, 67]. For each 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 TiSASRec, the maximum sequence length per user is set to 50.

**Procedure to Measure Stability.** We follow the procedure described in Section 4 and use the two RLS Metrics to measure the stability of recommendation models against random perturbations.

**Findings.** We present the RBO scores of four recommendation models on Foursquare against random leave-one-out perturbation in Figure 3(a). 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. Recall that since the RBO score between two randomly-shuffled rank lists is approximately 0.5, it shows that the drop of RBO caused by perturbations is meaningful, but the rank list does not change.
randomly, which is expected. Similar drops are observed for top-10 Jaccard similarity and in the case of insertion and replacement perturbations. Insertion and replacement perturbation results are excluded due to space limitation. Thus, we observe the instability of existing models against even minor random perturbation. Notably, perturbation of a user’s interaction leads to drastic changes in the recommendations of unrelated users.

Comparing the four models, LatentCross has the highest RBO in most cases against random perturbations. This indicates that LatentCross is the most stable model against random perturbations.

Controlling for Training Randomness while Measuring Model Stability. Other research has found that randomness during the training (e.g., random initialization, mini-batch shuffling, etc.) can generate different models and predictions in machine learning [3, 45]. Thus, in all our experiments (including the ones above), we specifically test the effect of the input data perturbation on model stability by controlling all other randomness (e.g., fixing the random seed and initialization). During a single run, we train two recommendation models $M$ and $M'$ (before and after perturbations) using the exact same settings without any training randomness. In other words, if there is no perturbation, the trained models $M$ and $M'$ and their outputs will be identical in every way.

Impact of Perturbations on Next-Item Prediction Metrics: We find that the trained recommender models with and without perturbations have similar dataset-level performance metrics (both have almost identical MRR and Recall scores), as shown in Table 3(b). However, the generated recommendation lists are drastically different, as indicated by the low RBO and Jaccard scores. This shows that multiple equivalent models can be trained that have similar dataset-level metrics, but provide conflicting recommendations. Similar findings have been made in other prediction settings [45].

One may wonder that if the dataset-level metrics are the same, is there any concern if the rank lists vary? We argue that this is indeed a matter of concern due to the following three reasons:

(a) Since several equivalent models generate different predictions, the specific recommendations, e.g., which drug to administer or which treatment procedure to follow, can vary depending on which model is used. It is important for the algorithm designer and the end-user to know that if the recommendation for a certain user can be easily changed by unrelated minor perturbations, then perhaps none of the recommendations should be followed for that user.

(b) Since multiple recommender models exist with equivalent next-item prediction performance, then how can the algorithm designer decide which model to deploy? We argue that given comparable models, stabler recommender models should be used.

(c) Our work highlights the importance of “beyond-accuracy” metrics (e.g., RLS metrics) given that different recommender models vary in their stability with respect to the RLS metrics.

Why are models unstable against minimal random perturbations? Only one interaction over one million interactions (size of the datasets used) is perturbed. Yet, it changes the rank lists and top-10 recommendation lists of all users. Why is there such a profound effect? This is due to two reasons:

(1) The slight change in training data leads to changes in the parameters of a trained recommendation model $M$. Say an interaction in a mini-batch $m$ was perturbed. When processing $m$, model parameters $\Theta(M)$ will be updated differently during training (compared to when there is no perturbation). The changes in $\Theta(M)$ will affect the updates in later mini-batches. The differences will further cascade and multiply over multiple epochs. Thus, with perturbations, the final $\Theta(M)$ will be different from the ones obtained without perturbations, which can result in different rank lists.

(2) The model $M$ is trained to accurately predict only the ground-truth next item as high in the rank list as possible (ideally, rank 1). However, $M$ is not trained to optimize the positions of the other items in the rank list. Thus, the ordering of all except the ground-truth next item is highly likely to change due to input perturbation.

6 Stability against Casper Perturbation

While random perturbations show the model instability introduced due to arbitrary errors and noise, it is essential to find perturbations that can lead to even higher instability, which helps understand the lowest stability exhibited by a model. Adversaries can potentially exploit such perturbations to conduct untargeted attacks and make the recommendations unstable for all users. Thus, in this section, we ask: which interaction should be perturbed to yield maximum instability in a recommendation model? We aim to find perturbations that maximally change the rank lists $R^X_M$ compared to $R^X_M$ $\forall X_k \in X_{test}$. As before, we will consider minimal interaction-level perturbations, allowing one interaction to be perturbed. Three types of perturbations can be made: leave-one-out (LOO), insertion, and replacement. Due to space constraints, we will highlight LOO perturbation results as other perturbations yield similar model instability. Finally, we will consider gray-box perturbations — we assume access to training data and some model information such as the maximum sequence length of past user actions that the recommendation model uses to make predictions. Note that we do not require any details of the recommendation model such as the model’s architecture, parameters, or gradients [19, 29, 80].

6.1 Perturbing Interactions from Different Timestamps

A brute-force technique that tests the impact of every interaction perturbation on model stability is computationally prohibitive due to the need to retrain the model after each perturbation. To find an effective perturbation in a scalable manner, we first investigate the impact of perturbations in different positions in the training data.

We take inspiration from an idea from temporal recommendation models [5, 26, 36], where mini-batches $B = \{B_1, \ldots, B_T\}$, $B_1 \cup \cdots \cup B_T = X_{train}$ are created in temporal order (i.e., first $P$ interactions in the first batch $B_1$, and so on). In such models, earlier batches contain training interactions with early timestamps, and perturbing an interaction in the earlier batches is equivalent to perturbing an interaction with early timestamps. Since we saw in the case of random perturbations that the impact of perturbations on model parameters can cascade, we ask: how does perturbing interactions from different timestamps impact model stability?

We devise and compare three following heuristic 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,
We perform evaluations on the LatentCross model and Foursquare dataset (the hardest-to-predict dataset as per next-item metrics). We perturb each interaction 10 times (randomly perturbing one interaction only each time). The resulting RBO score distributions are compared using the Wilcoxon signed-rank test [71].

The RBO scores are shown in Table 4(a). Earliest-Random perturbation leads to the lowest RBO score in all three types of perturbations, i.e., LOO, replacement, and insertion (all p-values <0.05). We also observe that between Random and Latest-Random, the former has lower RBOs. These findings show that perturbing earlier timestamps leads to higher instability in recommendations. Since this happens due to the cascading impact of model parameter changes over mini-batch updates, we call this a “cascading effect”.

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

Now, we leverage the cascading effect to propose a new perturbation, named CASPER (Cascade-based Perturbation).

To approximate the impact of perturbing an interaction $X_k$, we define a cascading score of $X_k$ as the number of training interactions that will be affected if $X_k$ is perturbed. Inspired by temporal recommendation models [5, 26, 36], we create an interaction-to-interaction dependency graph, which encodes the influence of one interaction on another. Then, we approximate the cascading score of interaction $X_k$ as the number of descendants of $X_k$ in this graph. CASPER aims to identify the training interaction which has the highest cascading score, since its perturbation would maximize the cascading effect. Algorithm 1 shows the key steps of the method.

### Algorithm 1: CASPER: interaction-level perturbation based on cascading effect

```
Input : Training interaction data $X_{\text{train}}$, users and items $U$ and $I$, training interaction sequences $X^u$ and $X^i$ (sorted by timestamp) for each user $u$ and item $i$
Output: Perturbed training data $X_{\text{perturbed}}$

1. for each user $u \in U$ do
   2. for $k \in [1, 2, \ldots, |X^u| - 1]$ do
   3. Create an edge from $X^u_k$ to $X^u_{k+1}$ in $G$
   4. for each item $i \in I$ do
   5. for $k \in [1, 2, \ldots, |X^i| - 1]$ do
   6. Create an edge from $X^i_k$ to $X^i_{k+1}$ in $G$
   7. for each interaction $X_k = (u, i, t) \in X_{\text{train}}$ do
   8. if $\text{Indegree}(X_k) == 0$ then
   9. Perform breadth-first search (BFS) starting from $X_k$ to find all descendants of $X_k$ in $G$
   10. $\text{score}(X_k) \leftarrow$ total number of descendants of $X_k$ in $G$

11. Perturb the interaction with the highest cascading score

To obtain new perturbed training data $X_{\text{perturbed}}$, perturb the interaction $X_{\text{opt}} = \arg \max \text{score}(X_k)$
```

Figure 4: (a) Average RBO scores of perturbing interactions from different positions in the training data. Earliest-Random perturbation produces lower RBO than Random and Latest-Random perturbations. (b) An IDAG corresponding to the interaction data in Figure 1. Blue and red edges indicate user- and item-sharing adjacent interactions, respectively. Green-colored nodes (interactions) show all descendants (including itself) of an interaction $X_k$. The cascading score of $X_3 = 8$, which is its number of descendants.

and a Random perturbation, where a random training interaction is perturbed. We test this cascading effect on LATENTCROSS model since it was the most stable against random perturbation.

We use the RBO metric to measure RLS caused by these perturbations on the LATENTCROSS model and Foursquare dataset (the hardest-to-predict dataset as per next-item metrics). We perform each perturbation 10 times (randomly perturbing one interaction only each time). The resulting RBO score distributions are compared using the Wilcoxon signed-rank test [71].

The RBO scores are shown in Table 4(a). Earliest-Random perturbation leads to the lowest RBO score in all three types of perturbations, i.e., LOO, replacement, and insertion (all p-values <0.05). We also observe that between Random and Latest-Random, the former has lower RBOs. These findings show that perturbing earlier timestamps leads to higher instability in recommendations. Since this happens due to the cascading impact of model parameter changes over mini-batch updates, we call this a “cascading effect”.

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

Now, we leverage the cascading effect to propose a new perturbation, named CASPER (Cascade-based Perturbation).

To approximate the impact of perturbing an interaction $X_k$, we define a cascading score of $X_k$ as the number of training interactions that will be affected if $X_k$ is perturbed. Inspired by temporal recommendation models [5, 26, 36], we create an interaction-to-interaction dependency graph, which encodes the influence of one interaction on another. Then, we approximate the cascading score of interaction $X_k$ as the number of descendants of $X_k$ in this graph. CASPER aims to identify the training interaction which has the highest cascading score, since its perturbation would maximize the cascading effect. Algorithm 1 shows the key steps of the method.

Creating the interaction-to-interaction dependency DAG: We create a graph-based technique to approximate an interaction’s cascading score without retraining the recommendation model. We first construct an interaction-to-interaction dependency directed acyclic graph (IDAG; lines 1-7 in Algorithm 1), where nodes are training interactions and directed edges represent which interaction influences another. The edge encodes the dependency that if the $k^{th}$ interaction of user $u$ (or item $i$) is perturbed, it will influence the $k + 1^{st}$ interaction of user $u$ (or item $i$). The IDAG corresponding to the training interactions from Figure 1 is presented in Figure 4(b).
Two nodes in the IDAG are connected by a directed edge if they are either consecutive interactions of the same user (e.g., X₁ and X₂) or of the same item (e.g., X₃ and X₄). 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 (first to the next interaction of the user and second to the next interaction of the item). If the recommendation model has a maximum sequence length (L), the IDAG is constructed only with the latest L interactions of each user.

**Calculating the cascading score in IDAG:** The cascading score of a node Xₖ is approximated as the total number of descendants of Xₖ in the IDAG. Descendants of a node Xₖ in the IDAG are defined as all the nodes reachable from Xₖ by following the outgoing edges in the IDAG. For example, in Figure 4(b), X₅ 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 computation by calculating the cascading scores of zero in-degree nodes only (lines 8-11 in Algorithm 1). Finally, CASPER perturbs the node with the highest cascading score since it would maximize the cascading effect (line 12 in Algorithm 1).

We have theoretically and experimentally shown that CASPER scales near-linearly to the dataset size (Section 6.3 and Figure 8(b)).

### 6.3 Complexity Analyses of CASPER

We analyze the time and space complexities of CASPER. We assume the maximum sequence length of a model is L.

**Time complexity.** CASPER first trains and tests a given recommendation model Θ with original input data, which takes \( O(T(Θ)) \), where \( T(Θ) \) is the time complexity of Θ. After that, CASPER constructs the IDAG which takes \( O(|U|L) \) where \(|U| \) is the number of users. Computing cascading scores of zero in-degree nodes in the IDAG, which takes \( O(Z|U|L) \) where \( Z \) is the number of zero in-degree nodes in the IDAG. Perturbing an interaction with the highest cascading scores takes \( O(Z) \). Finally, CASPER retrains the model Θ with perturbed data and computes RLS metrics, which takes \( O(T(Θ) + N_{test}|I|) \) since RBO should be calculated with all items \(|I| \), where \( N_{test} \) is the number of test interactions. The final time complexity of CASPER is \( O(T(Θ) + N_{test}|I| + Z|U|L) \).

**Space complexity.** The first step of CASPER is training and testing a deep sequential recommendation model Θ with original input data, which takes \( O(S(Θ) + N_{test}|I|) \) space since we need to store original rank lists for all test interactions, where \( S(Θ) \) is the space complexity of Θ. After that, CASPER constructs the IDAG which takes \( O(|U|L) \) space. The next step is computing cascading scores of zero in-degree nodes in the IDAG, which takes \( O(|U|L) \) space. Finally, CASPER retrains the model Θ with perturbed data and computes RLS metrics, which takes \( O(S(Θ) + N_{test}|I|) \) space. The final space complexity of CASPER is \( O(S(Θ) + N_{test}|I| + |U|L) \).

### 7 EXPERIMENTAL EVALUATION OF CASPER

In this section, we evaluate CASPER by the following aspects.

1. **Stability of Recommendation Models against Diverse Perturbations (Section 7.2).** How stable are existing recommender systems against CASPER and baseline perturbations?
2. **Impact of Perturbations on Different Users (Section 7.3).** Are there any user groups that are more susceptible and sensitive to input data perturbations?
3. **Impact of the Number of Perturbations (Section 7.4).** Is the performance of CASPER proportional to the number of perturbations allowed on the dataset?
4. **Running Time Analysis (Section 7.5).** Does the running time of CASPER scale with the dataset size?

#### 7.1 Experimental Settings

1. **Datasets.** We use the four standard datasets introduced in Section 5. LastFM is a widely used recommendation benchmark dataset [22, 31, 39, 55]. Foursquare is broadly utilized for point-of-interest recommendations [74, 76–78], and Wikipedia and Reddit are popular for social network recommendations [14, 36, 42, 51]. We select these datasets for experiments because (a) they come from diverse domains, thus ensuring generalizability, and (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).

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

   - **Random perturbation:** It randomly chooses an interaction for perturbation among all training interactions.
   - **Earliest-Random perturbation:** It randomly chooses an interaction for perturbation among the first interactions of all users in the training data.
   - **Latest-Random perturbation:** It randomly chooses an interaction for perturbation among the last interactions of all users in the training data.
   - **TracIn** [54] perturbation: It 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 [54] that utilizes loss gradients from the model saved at every \( T \) epoch to compute interaction importance.
   - **Rev.Adv.** [63] perturbation: It inserts a fake user with interactions crafted via a bi-level optimization problem for perturbations. To adapt it for our leave-one-out (LOO) and replacement perturbation settings, we first find the most similar user in the training data to the fake user, and perform LOO or item replacement of the earliest or random interaction of that user, respectively. Therefore, we create two versions of Rev.Adv. – Rev.Adv. [63] (random) and Rev.Adv. [63] (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 [4, 11, 44] and matrix factorization-based models [18, 72, 75] as these are not applicable to our setting. We have also not included baselines that have shown similar or worse performance [58, 80, 81] compared to the above baselines, particularly compared to Rev.Adv. [63]. 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.
Table 3: Effectiveness of perturbations on Foursquare dataset. We find instability of existing recommendation models measured by the RBO metric against LOO (left) and item replacement perturbations (right). All RBO scores are lower than 1.0. The best perturbation in each column is colored blue, and the second best is light blue, in terms of achieving the lowest RBO score.

(a) LOO perturbation comparison using RBO

<table>
<thead>
<tr>
<th>Model / Perturbations</th>
<th>LSTM</th>
<th>TiSASRec</th>
<th>JODIE</th>
<th>LATENTCross</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.779</td>
<td>0.9117</td>
<td>0.8316</td>
<td>0.9335</td>
</tr>
<tr>
<td>Earliest</td>
<td>0.786</td>
<td>0.8776</td>
<td>0.8211</td>
<td>0.8563</td>
</tr>
<tr>
<td>Random</td>
<td>0.776</td>
<td>0.8515</td>
<td>0.8420</td>
<td>0.9608</td>
</tr>
<tr>
<td>TracIn [54]</td>
<td>0.773</td>
<td>0.8545</td>
<td>0.8713</td>
<td>0.9625</td>
</tr>
</tbody>
</table>

(b) Item replacement perturbation comparison using RBO

<table>
<thead>
<tr>
<th>Model / Perturbations</th>
<th>LSTM</th>
<th>TiSASRec</th>
<th>JODIE</th>
<th>LATENTCross</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.779</td>
<td>0.9143</td>
<td>0.9414</td>
<td>0.9491</td>
</tr>
<tr>
<td>Earliest</td>
<td>0.774</td>
<td>0.8886</td>
<td>0.8871</td>
<td>0.8421</td>
</tr>
<tr>
<td>Random</td>
<td>0.781</td>
<td>0.9553</td>
<td>0.9898</td>
<td>0.9850</td>
</tr>
<tr>
<td>TracIn [54]</td>
<td>0.795</td>
<td>0.8320</td>
<td>0.9280</td>
<td>0.9696</td>
</tr>
</tbody>
</table>

Figure 5: Comparing perturbations on LATENTCross model across all datasets. CASPER shows the best perturbation performance.

7.1.3 Recommendation Models. We use popular recommender models: LSTM [26], TiSASRec [41], JODIE [36], and LATENTCross [5] described earlier to test the effectiveness of CASPER and baselines.

7.1.4 Experimental Setup. We follow the same experimental setup, as described previously in Section 5. Additionally, we use the following settings. We repeat all experiments multiple times and report average values of RLS metrics. To construct the IDAG for CASPER, we use all the interactions in JODIE and LATENTCross. For LSTM and TiSASRec, 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.

7.2 Stability of Recommendation Models against Diverse Perturbations

Perturbations of all models on Foursquare dataset. Table 3 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.

We observe the instability of all recommendation models against LOO and replacement perturbations. The RBO scores of all the recommendation models drop significantly below 1.0, indicating their low stability. CASPER achieves the best performance across all but one setting, where it performs the second best. It leads to the most reduction of RBO in most cases. CASPER shows lower variances of RLS values than those of baselines across different runs. We observe that CASPER is more effective on JODIE and LATENTCross models, since the other two models (LSTM and TiSASRec) use maximum sequence lengths, which limit their interactions’ cascading effects. In some cases, e.g., JODIE and LATENTCross in item replacement, their resulting RBO drops close to 0.5, which is similar to the case of random shuffling of ranked lists. Similar observations hold with top-K Jaccard score and for insertion perturbations. It is also worth mentioning that Rev.Adv. (earliest) outperforms Rev.Adv. (random) in most cases, which also substantiates the cascading effect.

In item replacement perturbation, CASPER outperforms other methods in all cases. For CASPER, 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. This major update will cascade to later interactions and change all users’ recommendations drastically.

Perturbations on LATENTCross model on all datasets. We further evaluate the effectiveness of CASPER versus baselines on the LATENTCross model (the most stable model against random perturbations) across four datasets. The results are shown in Figures 5(a) and 5(b). We confirm unstable predictions of LATENTCross against CASPER LOO and item replacement perturbations as well.
To investigate the differential impact of training data perturbations, Top-K Jaccard metric and insertion perturbations also show similar trends. Users with low accuracy suffer more from training data perturbations with any perturbation method. CASPER leads to the highest reduction of top-10 Jaccard similarity and outperforms all baselines across all user groups.

7.3 Impact of Perturbations on Different Users

To investigate the differential impact of training data perturbations on different users, we divide users into three groups: (1) High-MRR users, containing users who lie in the top 20% according to average MRR, (2) Low-MRR users, containing users with the lowest 20% average MRR, and (3) Mid-MRR users, which contains the remaining set of users. We contrast the average RLS of users across the three groups. Figures 6(a) and 6(b) compare the top-10 Jaccard scores across the three user groups on LatentCross model and two datasets (Foursquare and LastFM) against CASPER LOO perturbation. We discover that the trend of stability follows the accuracy trend – users with high accuracy receive relatively more stable predictions than the low-accuracy user group. This phenomenon highlights the relatively higher instability faced by users for which the model is already unable to make accurate predictions. This raises an aspect of unfairness across user groups. This highlights the need that practitioners should evaluate model stability across user groups before deploying models in practice.

Furthermore, we observe the same trend across different perturbation methods, as shown in Figure 7. Regardless of the perturbation method, low-MRR users experience lower stability compared to the other two groups. Notably, CASPER is able to generate the lowest stability across all user groups. Addressing the differential impact across user groups will be important to study in future work.

7.4 Impact of the Number of Perturbations

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

8 CONCLUDING REMARKS

Our work highlights that recommendation models can exhibit instability to minor changes in their training data. These effects underscore the need to measure this instability, and to develop methods that are robust to such changes. The measures and methods developed in this paper are an initial step in this direction. In particular, CASPER depends on cascading effect which is inspired by temporal recommendation models, meaning that it may return solutions that are sub-optimal for methods that are not trained with temporally-ordered mini-batches.

Future work topics include: expanding CASPER to handle more complex perturbations, to find more effective perturbations (e.g., interaction reordering) for other training regimes; and improving scalability of CASPER to handle very large interaction graphs (e.g., by creating approximations of cascading scores using a randomly-sampled interaction graphs, rather than the entire graph); developing methods that induce stability to data perturbations (e.g., via multi-objective learning aiming to accurately predict next items and preserve rank lists of a recommendation model simultaneously).

ACKNOWLEDGMENTS

This research is supported in part by Georgia Institute of Technology, IDEA$S$, and Microsoft Azure. S.O. was partly supported by ML@GT, Twitch, and Kwanjeong fellowships. We thank the reviewers for their feedback.


