A Self-Correcting Sequential Recommender

Yujie Lin
Shandong University
Qingdao, China
yu.jie.lin@outlook.com

Chenyang Wang
Shandong University
Qingdao, China
201900122032@mail.sdu.edu.cn

Zhumin Chen
Shandong University
Qingdao, China
chenzhumin@sdu.edu.cn

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
m.derijke@uva.nl

Xiuzhen Cheng
Shandong University
Qingdao, China
xzcheng@sdu.edu.cn

Pengjie Ren∗
Shandong University
Qingdao, China
renpengjie@sdu.edu.cn

ABSTRACT

Sequential recommendations aim to capture users’ preferences from their historical interactions so as to predict the next item that they will interact with. Sequential recommendation methods usually assume that all items in a user’s historical interactions reflect her/his preferences and transition patterns between items. However, real-world interaction data is imperfect in that (i) users might erroneously click on items, i.e., so-called misclicks on irrelevant items, and (ii) users might miss items, i.e., unexposed relevant items due to inaccurate recommendations.

To tackle the two issues listed above, we propose STEAM, a Self-correcting Sequential Recommender. STEAM first corrects an input item sequence by adjusting the misclicked and/or missed items. It then uses the corrected item sequence to train a recommender and make the next item prediction. We design an item-wise corrector that can adaptively select one type of operation for each item in the sequence. The operation types are ‘keep’, ‘delete’ and ‘insert’. In order to train the item-wise corrector without requiring additional labeling, we design two self-supervised learning mechanisms: (i) deletion correction (i.e., deleting randomly inserted items), and (ii) insertion correction (i.e., predicting randomly deleted items). We integrate the corrector with the recommender by sharing the encoder and by training them jointly. We conduct extensive experiments on three real-world datasets and the experimental results demonstrate that STEAM outperforms state-of-the-art sequential recommendation baselines. Our in-depth analyses confirm that STEAM benefits from learning to correct the raw item sequences.

CCS CONCEPTS
• Information systems → Recommender systems.

KEYWORDS
Sequential recommendation, Sequence correction, Self-supervised learning


1 INTRODUCTION

An important challenge of sequential recommendation is how to capture a user’s preferences as accurately as possible by modeling the sequential dependencies of a user’s historical interaction sequences [6, 40, 41]. There is a considerable body of prior work towards this goal. Early models are based on the Markov chain (MC) assumption that the next item only depends on its adjacent items to learn transition relationships [9, 11, 31]. Later, deep learning-based methods have been applied to sequential recommendation tasks for modeling more complex relations, such as recurrent neural networks (RNNs) [12, 24, 30, 47], convolutional neural networks (CNNs) [35, 37, 53], memory networks [4, 14, 39], transformers [15, 34, 48], and graph neural networks (GNNs) [2, 49, 54]. More recently, self-supervised learning (SSL) has been introduced to sequential recommendation for extracting robust item correlations by semi-automatically exploiting raw item sequences [23, 50–52].

Most prior work ignores the fact that user-item interaction sequences may be imperfect, which means that they do not always accurately reflect user preferences and the transition patterns between items. As illustrated in Fig. 1, there may be two kinds of imperfections in item sequences: (i) The user may erroneously click on irrelevant items, so the item sequence may contain misclicked items. For example, in Fig. 1(b), the user mistakenly clicks $i_0$, which is a book about computer basics. As a result, the recommendation model may recommend another book about computer science, i.e.,
item. For misclicked items, the corrector can delete them, while the latter randomly deletes items from the raw item sequence and recalls them afterwards. We integrate the item-wise corrector and a recommender in STEAM by sharing the encoder. For the recommender, we use the raw item sequence and the corrected item sequence to train it by the masked item prediction task [34]. We use the joint loss from the corrector and the recommender to train the whole STEAM. We conduct experiments on three real-world datasets. The results show that STEAM significantly outperforms state-of-the-art sequential recommendation baselines. We find that STEAM benefits from learning to correct input sequences for better recommendation performance. We also carry out experiments on simulated test sets by randomly inserting and deleting items, demonstrating that STEAM is more robust than most baselines on more noisy data.

The main contributions of this work are as follows:

• We propose a self-correcting sequential recommender (STEAM) that is able to correct the raw item sequence before conducting recommendation.

• We design an item-wise corrector to correct the raw item sequence and two self-supervised learning mechanisms, deletion correction and insertion correction, to train the corrector.

• We conduct extensive experiments to demonstrate the state-of-the-art performance of STEAM. To facilitate reproducibility, we release the code and data at https://github.com/TempSDU/STEAM.

2 RELATED WORK

2.1 Sequential recommendation

Early work on sequential recommendation adopts MC to capture the dynamic transition of user interactions. Rendle et al. [31] combine first-order MCs and matrix factorization (MF) to predict the subsequent user action. He and McAuley [11] employ the high-order MCs to consider more preceding items and mine more complicated patterns. With the development of deep learning, neural networks have been introduced to address sequential recommendation. Hidasi et al. [13] adopt gated recurrent units (GRUs) [5] to build a sequential recommendation model. Li et al. [19] enhance the GRU-based sequential recommendation model with an attention mechanism [3] to more accurately capture the user’s current preference. Tang and Wang [35] propose a CNN-based model to model sequential patterns in neighbors. Later, more advanced neural networks have been applied. Chen et al. [4] introduce a memory mechanism [46] to design a memory-augmented neural network for leveraging users’ historical records more effectively. Kang and McAuley [15] employ the unidirectional transformer [38] to capture long-range correlations between items. Sun et al. [34] further use a bidirectional transformer and the masked item prediction task for sequential recommendation. Wu et al. [49] utilize the GNN [45] to model more complex item transition patterns in user sequences.

Recently, self-supervised learning has demonstrated its effectiveness in extracting contextual features by constructing training signals from the raw data with dedicated tasks [20]; it has been
We denote the item set as $\mathcal{I}$, and $|\mathcal{I}|$ is the number of items. We denote an item sequence as $S = [i_1, ..., i_\ell]$, where $i_j \in \mathcal{I}$ is the interacted item at the $t$-th position of $S$ and $|S|$ is the sequence length. We denote a subsequence $[i_j, ..., i_k]$ of $S$ as $S_{j:k}$. Especially, we denote a raw item sequence as $S'$. For training STEAM with deletion and insertion correction, we randomly modify $S'$ and then ask STEAM to recover it. For each item in $S'$, we keep it with probability $p_k$, insert one item before it with probability $p_i$, or delete it with probability $p_d$, where $p_k + p_i + p_d = 1$. Note that we can keep inserting more items with $p_i$, which are all sampled uniformly from $\mathcal{I} \setminus S$. For the last item in $S'$, we would not delete it to avoid confusion with the next item prediction. The randomly modified sequence is denoted as $S^m$. We have an operation sequence to mark the ground-truth correction operations on the items of $S^m$, denoted as $O = [o_1, ..., o_{|S^m|}]$, where $o \in \{'\text{keep}'$, ‘delete’, ‘insert’$\}$. Note that we do not consider ‘replace’ as it can be achieved by the combination of ‘delete’ and ‘insert.’ We denote all items in $S^m$ whose ground-truth correction operations are ‘insert’ as $I^{ins}$. For each item $i \in I^{ins}$, we denote the ground-truth inserted sequence as $S^{<i} = [i_1, ..., i_{|S'|+1}, \{\text{eos}\}]$, which should be inserted in reverse order before $i$, where $\{\text{eos}\} \in \mathcal{I}$ is a special token representing the ending. Note that the order is $[i_{|S'|+1}, ..., i_1]$ after insertion. We revise $\mathcal{S}'$ using the corrector to get the corrected sequence, which is denoted as $\mathcal{S}$. To train STEAM with the masked item prediction, we randomly mask some items in $S'$ or $\mathcal{S}'$ with a special token $\{\text{mask}\} \in \mathcal{I}$ with probability $p_m$, and then ask STEAM to predict masked items. The masked $\mathcal{S}'$ ($\mathcal{S}'$) and its masked items are denoted as $\mathcal{S}'(\mathcal{S}')$ and $\mathcal{P}(\mathcal{P})$.

The deletion and insertion correction tasks are to maximize $P(S' | S^m) = P(O | S^m) \times \prod_{i \in I^{ins}} P(S^{<i} | S^m)$. The masked item prediction task is to maximize $P(\mathcal{P} (\mathcal{S}') | S'$ and $P(\mathcal{P}(\mathcal{S}'))$. The sequential recommendation task aims to predict the next item based on $P(i_{\mathcal{S}'|+1} | S')$ or $P(\mathcal{P}(i_{\mathcal{S}'|+1} | S'))$, which is equivalent to predict a masked item appending to the last position of $S'$ or $\mathcal{S}'$.

An overview of STEAM is shown in Fig. 2. STEAM has three main modules: (i) a shared encoder, (ii) an item-wise corrector, and (iii) a recommender. The encoder is used to encode the input sequence. The item-wise corrector first predicts the correction operations on all items. Then, the items whose correction operations are ‘delete’ will be deleted. For the items whose correction operations are ‘insert’, the item-wise corrector uses the reverse generator to generate all inserted sequences. The recommender aims to predict the masked items in item sequences. To train the corrector, we take the randomly modified item sequence as input and ask the corrector to recover it. To train the recommender, we first use the corrector to get the corrected item sequence. Then, we randomly mask some items in the corrected item sequence or the raw item sequence, and use the recommender to predict them. Finally, we use the joint loss to optimize STEAM. During testing, we append a $\{\text{mask}\}$ to the last position of the raw item sequence or the corrected item sequence, and use STEAM to predict the next item.

Next, we provide the details of STEAM.

3 METHOD
3.1 Overview
We denote the item set as $\mathcal{I}$, and $|\mathcal{I}|$ is the number of items. We denote an item sequence as $S = [i_1, ..., i_{|S|}]$, where $i_j \in \mathcal{I}$ is the interacted item at the $t$-th position of $S$ and $|S|$ is the sequence length. We denote a subsequence $[i_j, ..., i_k]$ of $S$ as $S_{j:k}$. Especially, we denote a raw item sequence as $S'$. For training STEAM with...
3.3 Item-wise corrector

The item-wise corrector aims to execute correction operations at the item level. It selects one type of correction operation from ‘keep’, ‘delete’ and ‘insert’ for each item in the raw item sequence. If the selected correction operation is ‘insert’ for an item, the item-wise corrector further uses a generator to generate the inserted sequence.

Given an item $i_t$ with its hidden representation $h_t \in \mathbb{R}^e$ indexed from the input $H_e$, we follow Eq. 4 to obtain the probability distribution $P(\hat{\delta}_t | S)$ for the corresponding correction operation $\hat{o}_t$:

$$P(\hat{\delta}_t | S) = \text{softmax}(Wh_t),$$

where $\hat{o}_t$ is the predicted version of $o_t$, $W \in \mathbb{R}^{3 \times e}$ is the projection matrix. When testing, the operation with the maximum probability in $P(\hat{\delta}_t | S)$ will be applied to the item $i_t$.

Assuming that we have to insert items before the item $i_t$ of $S$ and its currently generated inserted sequence is $S^{<c_i}_{1:n-1}$, the reverse generator first follows the same way in the encoder to get the item embeddings of all items in $S^{<c_i}_{1:n-1}$. Note that the item embedding matrix is shared between the encoder and the corrector.

Then, we stack the hidden representation $h_t$ of $i_t$ obtained from $H_e$ with all item embeddings $\{e_1, \ldots, e_{n-1}\}$ of $S^{<c_i}_{1:n-1}$:

$$H^e_{c} = \begin{bmatrix} h_t + p_t \vline e_1 + p_1 \vline \ldots \vline e_{n-1} + p_{n-1} \end{bmatrix},$$

where $H^e_{c} \in \mathbb{R}^{n \times e}$ is the initial hidden representation matrix for the reverse generator. We also add the position embeddings here, which are shared between the encoder and the corrector too. We apply dropout to $H^e_{c}$ as we do in the encoder.

Next, the reverse generator uses a unidirectional transformer with $L_c$ layers to update $H^e_{c}$ as Eq. 6:

$$H^e_{c} = \text{Trm}_{uni}(H^e_{c-1}),$$

where $\text{Trm}_{uni}$ represents a unidirectional transformer block; again, please see [15, 38] for details. $H^e_{c} \in \mathbb{R}^{n \times e}$ is the hidden representation matrix at the $l$-th layer. $H^e_{c}$ is the last hidden representation matrix, which will be the input of the item-wise corrector and the recommender. For convenience, we ignore the superscript of $H^e_{c}$ (i.e., $H_c$) in the following modules.

Finally, we calculate the probability distribution $P(\hat{i}_n | S^{<c_i}_{1:n-1}, S)$ for the next inserted item $i_n$ by Eq. 7:

$$P(\hat{i}_n | S^{<c_i}_{1:n-1}, S) = \text{softmax}(E^\top h_n),$$
where $E$ is the item embedding matrix, $h_{\mathbf{e}} \in \mathbb{R}^e$ is the hidden representation at the last position of $H_c$. In particular, the first inserted item $i_t$ is generated based on $H^0_c = [h_t + p_1]$, so we define $P(i_t \mid S_{<t}^{\text{c}}, S) = P(\hat{i}_t \mid S)$. 

Since we have the complete ground-truth inserted sequence $S^{<t}$, when training, we can use the hidden representations of all positions of $H_c$ to calculate $P(\hat{i}_{n+1} \mid S_{<t}^{\text{c}}, S)$ for all $n$ at one time. When testing, we will generate inserted items one by one with greedy search until generating $[\text{eos}]$ or achieving a maximum length.

### 3.4 Recommender

The recommender is to predict the masked items in $\tilde{S}$ or $\tilde{S}^c$, and to recommend the next item for $S'$ or $S^c$. 

Given the hidden representation matrix $H_c$ of the input sequence $S$ from the encoder, we let $H^0_c = H_c$, where $H^0_c \in \mathbb{R}^{|S| \times e}$ is the initial hidden representation matrix for the recommender. First, the recommender utilizes a bidirectional transformer with $L_r$ layers to update $H^r_c$, as shown in Eq. 8:

$$H^r_c = \text{Trm}_b(H^{r-1}_c),$$

where $H^r_c \in \mathbb{R}^{|S| \times e}$ is the hidden representation matrix at the $l$-th layer. $H^{L_r}_c$ is the last hidden representation matrix, and we also denote it as $H_c$ for short.

Then, assuming that we mask item $i_t$ in $S$ and obtain its hidden representation $h_t \in \mathbb{R}^e$ from $H_c$, the recommender follows Eq. 9 to calculate the probability distribution $P(\hat{i}_t \mid S)$ for $i_t$:

$$P(\hat{i}_t \mid S) = \text{softmax}(E^T h_t),$$

where $E$ is the item embedding matrix, which is also used in the encoder and the corrector. When training, the item $i_t \in \tilde{P}$ ($\tilde{F}$) and the sequence $S$ is $\tilde{S}$ ($\tilde{S}^c$). When testing, the item $i_t$ is $i_{|S^c|+1} (i_{|S'|+1})$ and the sequence $S$ is $S'$ ($S^c$), so we can get $P(\hat{i}_{|S^c|+1} \mid S^c) (P(\hat{i}_{|S'|+1} \mid S'))$ for recommending the next item.

### 3.5 Joint learning

We train STEAM with the deletion correction task, the insertion correction task, and the masked item prediction task.

For the correction tasks, we first randomly insert or delete items in a raw sequence $S'$ to get a modified sequence $S^m$, then we ask STEAM to delete the inserted items and insert the deleted items for $S^m$. Through the deletion correction task and the insertion correction task, we can obtain self-supervised signals for correcting raw input sequences without additional manual labeling. Specifically, our goal is to minimize the negative log-likelihood of $P(S' \mid S^m)$, as shown in Eq. 10:

$$L_1 = -\log P(S' \mid S^m)$$

$$= -\left( \log P(O \mid S^m) + \sum_{i \in T^{m}} \log P(S_{<i}^c \mid S^m) \right)$$

$$= -\sum_{i=1}^{S_{<i}^c} \log P(\hat{S}_{i} = o_i \mid S^m) + \sum_{i \in T^{m}} \sum_{n=1}^{S_{<i}} \log P(\hat{i}_n = i_n \mid S_{<i-1}^m),$$

where $L_1$ is the loss for the corrector.

In the masked item prediction task, we first employ STEAM to correct $S'$ to get the corrected item sequence $S^c$, then randomly mask items in $S'$ and $S^c$, and use STEAM to predict the masked items. We also minimize the negative log-likelihood of $P(\hat{S}^c \mid S')$ and $P(\hat{S}^c \mid \tilde{S})$, i.e., Eq. 11:

$$L_2 = -\left( \log P(\hat{S}^c \mid S') + \log P(\hat{S}^c \mid \tilde{S}) \right)$$

$$= -\left( \sum_{i \in P} \log P(\hat{i} = i \mid S') + \sum_{i \in P} \log P(\hat{i} = i \mid \tilde{S}) \right),$$

where $L_2$ is the loss for the recommender.

Finally, we use the joint loss $L$ shown in Eq. 12 to optimize the parameters of STEAM:

$$L = L_1 + L_2.$$  

Here, $L$ is minimized by the standard backpropagation algorithm.

### 4 EXPERIMENTAL SETUP

#### 4.1 Research questions

We seek to answer the following research questions: **(RQ1)** How does STEAM perform compared with the state-of-the-art sequential recommendation methods? **(RQ2)** What benefits can the recommender in STEAM obtain from the corrector in STEAM? **(RQ3)** How does STEAM perform with different levels of noise?

#### 4.2 Datasets

We evaluate STEAM on three datasets with varying domains. **Beauty** and **Sports** belong to a series of product review datasets crawled from Amazon.com by McAuley et al. [25]. We select Beauty and Sports subcategories in this work following Zhou et al. [57]. **Yelp** is a dataset for business recommendation released by Yelp.com. As it is very large, we only use the transaction records between "2019-01-01" and "2019-12-31."

We follow common practices [15, 57] to preprocess all datasets. We remove users and items whose interactions are less than 5. We sort each user’s interactions by time to construct an item sequence. For each item sequence, we use the last item for testing, the second last item for validation, and the remaining items for training. We pair the ground-truth item for testing or validation with. For each dataset, we also construct a simulated test set, which is different from the real test set by randomly modifying test sequences to introduce more imperfections. For each item in a test sequence excluding the ground-truth item, we will keep it with probability 0.8, insert one item before it (simulating misclicks on irrelevant items) or delete it (simulating unexposed relevant items due to inaccurate recommendations) with probability 0.1. We limit the count of continuous insertion operations less than 5. The inserted item is sampled uniformly from the item set. The statistics of the processed datasets are summarized in Table 1.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Users</th>
<th>#Items</th>
<th>#Actions</th>
<th>Avg. length</th>
<th>Sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beauty</td>
<td>22,362</td>
<td>12,101</td>
<td>194,682</td>
<td>8.7</td>
<td>99.93%</td>
</tr>
<tr>
<td>Sports</td>
<td>35,597</td>
<td>18,357</td>
<td>294,483</td>
<td>8.3</td>
<td>99.95%</td>
</tr>
<tr>
<td>Yelp</td>
<td>22,844</td>
<td>16,552</td>
<td>236,999</td>
<td>10.4</td>
<td>99.94%</td>
</tr>
</tbody>
</table>

Table 1: Statistics of the datasets after preprocessing.
4.3 Baselines
We compare STEAM with the following representative baselines, which can be grouped into (i) vanilla sequential recommendation models, (ii) SSL-based sequential recommendation models, and (iii) denoising sequential recommendation models. For each group, we only consider the recent state-of-the-art methods.

- Vanilla sequential recommendation models:
  - GRU4Rec [13] employs a GRU to model sequential patterns between items for sequential recommendation.
  - SASRec [15] uses a unidirectional transformer to model item sequences for predicting next items.
  - BERT4Rec [34] adopts a bidirectional transformer trained on the masked item prediction task.
  - SRGNN [49] models item sequences using a GNN with an attention network.

- SSL-based sequential recommendation models:
  - CL4SRec [51] uses three self-supervised tasks based on item crop, item mask, and item reorder respectively to train a transformer-based sequential recommendation model.
  - DuoRec [29] is the state-of-the-art SSL-based sequential method that employs a model-level augmentation approach based on dropout and a novel sampling strategy to construct contrastive self-supervised signals.

- Denoising sequential recommendation models:
  - FMLP-Rec [57] integrates FFT with an all-MLP architecture for denoising in sequential recommendation. There are few research on denoising for sequential recommendation work. We select FMLP-Rec as it is the latest state-of-the-art one.
  - Self-supervised learning improves the robustness of DuoRec and CL4SRec to resist the influence of imperfections including noise in item sequences by FFT, while the vanilla models may overfit on the noisy data due to their over-parameterized architectures [18, 26]. FMLP-Rec performs worse than DuoRec and is comparable to CL4SRec. Self-supervised learning improves the robustness of DuoRec and CL4SRec to resist the influence of imperfections including noise in item sequences themselves.

5 EXPERIMENTAL RESULTS
5.1 Overall performance
To answer RQ1, we compare STEAM against the baselines listed in Section 4.3 on the real test sets specified in Section 4.2. Table 2 lists the evaluation results of all methods. Based on these results, we have the following observations.

First, STEAM consistently outperforms all baselines by a large margin on most evaluation metrics of all datasets. Although STEAM only achieves the second best performance in terms of MRR@5 and MRR@10 on the Beauty dataset, it is almost comparable with the best result. Compared with the baselines using single recommendation models, STEAM jointly trains the recommender and the item-wise corrector, which shares the encoder and item embeddings. On the one hand, the deletion and insertion correction tasks for the corrector can provide the recommender with powerful self-supervised signals to obtain better item representations and robust item correlations. On the other hand, the corrector can correct the input sequence to reduce imperfections so that the recommender in STEAM predicts the next item more accurately. A detailed analysis can be found in the following sections.

Second, the SSL-based models CL4SRec and DuoRec perform better than the vanilla models on all metrics and datasets. Especially, DuoRec achieves the second best performance on most metrics, and achieves the best performance in terms of MRR@5 and MRR@10 on the Beauty dataset. This demonstrates that self-supervised learning can improve the performance of sequential recommendation by deriving extra supervision signals from item sequences themselves. The performance of DuoRec is obviously better than CL4SRec, which confirms the effectiveness of the model-level augmentation method and the sampling strategy proposed in DuoRec.

Third, although the denoising model FMLP-Rec is an all-MLP model, it shows superior performance compared to the vanilla models that adopt more complex architectures like the transformer. This is because FMLP-Rec can filter out noisy information in item sequences by FFT, while the vanilla models may overfit on the noisy data due to their over-parameterized architectures [18, 26]. FMLP-Rec performs worse than DuoRec and is comparable to CL4SRec. Self-supervised learning improves the robustness of DuoRec and CL4SRec to resist the influence of imperfections including noise in item sequences [21, 52], so they can fully exploit the power of the transformer to model sequences.

5.2 Benefits of the corrector
To answer RQ2, we analyze the effect of the item-wise corrector and the self-supervised correction tasks.

We first compare the performance of STEAM and its recommender in Table 2. We observe that STEAM significantly outperforms its recommender in terms of all evaluation metrics on all datasets. Therefore, we can attribute the improvement of STEAM on sequential recommendation to the integration of the recommender with the item-wise corrector. Moreover, by comparing the recommender of STEAM and BERT4Rec, we find that they have almost similar performance. Because we train the recommender of STEAM by the masked item prediction task separately, the recommender of STEAM is equivalent to BERT4Rec in this case.

As shown in Table 3, we use Overall-R and Overall-C to denote the performances of STEAM on the raw item sequences and the corrected item sequences of the real test sets, respectively. We see that Overall-C is slightly better than Overall-R on most evaluation
We can verify that most test item sequences do not change after correction. Therefore, we can confirm that the corrector helps the recommender achieve better performance by correcting input sequences.

Table 2: Performance comparison of different methods on the real test sets. The best performance and the second best performance are denoted in bold and underlined fonts respectively. * indicates that the performance gain of STEAM against the best baseline is statistically significant based on a two-sided paired t-test with \( p < 0.05 \).

<table>
<thead>
<tr>
<th>Model</th>
<th>Real Beauty</th>
<th>Real Sports</th>
<th>Real Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>GRU4Rec</td>
<td>32.95</td>
<td>42.59</td>
<td>21.63</td>
</tr>
<tr>
<td>SASRec</td>
<td>36.58</td>
<td>45.57</td>
<td>25.43</td>
</tr>
<tr>
<td>BERT4Rec</td>
<td>36.67</td>
<td>47.28</td>
<td>23.38</td>
</tr>
<tr>
<td>SRGNN</td>
<td>37.33</td>
<td>47.65</td>
<td>25.15</td>
</tr>
<tr>
<td>CL4SRec</td>
<td>39.29</td>
<td>48.75</td>
<td>27.59</td>
</tr>
<tr>
<td>DuoRec</td>
<td>40.95</td>
<td>50.78</td>
<td>28.84</td>
</tr>
<tr>
<td>FMLP-Rec</td>
<td>39.69</td>
<td>48.72</td>
<td>28.01</td>
</tr>
<tr>
<td>Recommender</td>
<td>35.73</td>
<td>46.47</td>
<td>22.84</td>
</tr>
<tr>
<td>STEAM</td>
<td>42.57*</td>
<td>52.89*</td>
<td>28.75</td>
</tr>
</tbody>
</table>

Table 3: Performance analysis of STEAM on different groups of the real test sets. Overall-R (Overall-C) is the performance on all raw (corrected) test item sequences. Changed-R (Changed-C) is the performance on the raw (corrected) test item sequences of the changed sequence group. Unchanged is the performance on the test item sequences of the unchanged sequence group.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Changed</th>
<th>#Keep</th>
<th>#Delete</th>
<th>#Insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Beauty</td>
<td>29.91</td>
<td>88.60</td>
<td>4.03</td>
<td>7.37</td>
</tr>
<tr>
<td>Real Sports</td>
<td>23.82</td>
<td>95.72</td>
<td>4.21</td>
<td>0.07</td>
</tr>
<tr>
<td>Real Yelp</td>
<td>2.17</td>
<td>99.63</td>
<td>0.15</td>
<td>0.22</td>
</tr>
</tbody>
</table>

Table 4: Statistics of correction operations by STEAM on the real test sets. #Changed is the percentage of the changed test item sequences after correction. #Keep, #Delete and #Insert are the percentages of different types of correction operations during correction. Next, we compare Overall-R in Table 2 with all baselines in Table 2. The comparison shows that STEAM can significantly outperform all baselines even based on the raw sequences. We also conclude that the joint learning with the corrector accounts for most of the improvement of STEAM over existing methods. This is because the recommender and the corrector share the encoder and item embeddings. The item representations and the item correlations learned by the self-supervised deletion correction and insertion correction tasks can be transferred to the recommender, which can enable the recommender to obtain better recommendation results and a robust performance on imperfect sequences.

Finally, we count the percentages of different correction operations performed by STEAM during correction. See Table 4. We observe that STEAM chooses to keep items in most cases. This is reasonable because most item sequences reflect normal user behavior data without noise. Therefore, STEAM tends to keep most sequences and items unmodified. Based on the statistics, we see that STEAM is not only able to delete items but also insert items, which is a key difference from denoising methods like FMLP-Rec.

We also conduct ablation studies to analyze the effectiveness of each self-supervised task and each correction operation, please see Appendix A and B for details.

5.3 Robustness analysis

To answer RQ3, we conduct experiments and analyses on the simulated test sets defined in Section 4.2 by randomly inserting and/or deleting items. The experimental results are shown in Table 5. The main observations from Table 5 are similar to those from Table 2. The performance for most baselines decreases as most of
than STEAM on the Yelp dataset, it may sacrifice recommendation performance for robustness. Similarly, SASRec obtains better values on some metrics, but its recommendation performance is relatively inferior. In contrast, DuoRec is the best baseline in recommendation performance, but its robustness is worse than FMLP-Rec and STEAM. We conclude that STEAM strikes a better balance between recommendation performance and robustness than other methods.

6 CONCLUSION AND FUTURE WORK

We have presented STEAM, a self-correcting sequential recommender that can learn to correct the raw item sequence before making recommendations by identifying users’ misclicks on irrelevant items and/or recalling unexposed relevant items due to inaccurate recommendations. In order to train the corrector without manual labeling work, we have proposed two self-supervised tasks, the deletion correction and the insertion correction, which randomly insert or delete items and ask the corrector to recover them. We have conducted extensive experiments on three real-world datasets to show STEAM consistently outperforms state-of-the-art sequential recommendation baselines and achieves robust performance on simulated test sets with more imperfect cases.

STEAM has the following limitations: (i) it cannot execute ‘delete’ and ‘insert’ on the same item simultaneously; and (ii) it can only insert items before the chosen item. As to future work, we plan to design a more flexible corrector by repeating the correction process with multiple iterations. We would also like to combine the corrector with other recommendation models and tasks besides BERT4Rec and the masked item prediction task.

Table 5: Performance comparison of different methods on the simulated test sets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Simulated Beauty</th>
<th>Simulated Sports</th>
<th>Simulated Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>GRU4Rec</td>
<td>32.22</td>
<td>42.13</td>
<td>21.28</td>
</tr>
<tr>
<td>SASRec</td>
<td>35.97</td>
<td>45.26</td>
<td>24.97</td>
</tr>
<tr>
<td>BERT4Rec</td>
<td>35.83</td>
<td>46.79</td>
<td>22.79</td>
</tr>
<tr>
<td>SRGNN</td>
<td>36.64</td>
<td>46.81</td>
<td>24.50</td>
</tr>
<tr>
<td>CL4SRec</td>
<td>38.66</td>
<td>48.22</td>
<td>26.96</td>
</tr>
<tr>
<td>DuoRec</td>
<td>40.26</td>
<td>50.13</td>
<td>28.39</td>
</tr>
<tr>
<td>FMLP-Rec</td>
<td>39.38</td>
<td>48.47</td>
<td>27.85</td>
</tr>
<tr>
<td>Recommender</td>
<td>35.14</td>
<td>45.96</td>
<td>22.22</td>
</tr>
<tr>
<td>STEAM</td>
<td>42.09∗</td>
<td>52.21∗</td>
<td>28.45</td>
</tr>
</tbody>
</table>

Table 6: Robustness analysis of different models. Each value is a performance disturbance.

<table>
<thead>
<tr>
<th>Model</th>
<th>Beauty</th>
<th>Sports</th>
<th>Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>GRU4Rec</td>
<td>-2.21%</td>
<td>-1.08%</td>
<td>-1.62%</td>
</tr>
<tr>
<td>SASRec</td>
<td>-1.67%</td>
<td>-0.68%</td>
<td>-1.81%</td>
</tr>
<tr>
<td>BERT4Rec</td>
<td>-2.29%</td>
<td>-1.04%</td>
<td>-2.52%</td>
</tr>
<tr>
<td>SRGNN</td>
<td>-1.85%</td>
<td>-1.76%</td>
<td>-2.58%</td>
</tr>
<tr>
<td>CL4SRec</td>
<td>-1.60%</td>
<td>-1.09%</td>
<td>-2.28%</td>
</tr>
<tr>
<td>DuoRec</td>
<td>-1.68%</td>
<td>-1.28%</td>
<td>-1.56%</td>
</tr>
<tr>
<td>FMLP-Rec</td>
<td>-0.78%</td>
<td>-0.51%</td>
<td>-0.57%</td>
</tr>
<tr>
<td>Recommender</td>
<td>-1.65%</td>
<td>-1.10%</td>
<td>-2.71%</td>
</tr>
<tr>
<td>STEAM</td>
<td>-0.28%</td>
<td>-1.02%</td>
<td>+0.64%</td>
</tr>
</tbody>
</table>

them cannot handle noisy and missed items. It is worth noting that STEAM achieves better MRR@5 and MRR@10 than DuoRec on the simulated Beauty test set, which indicates that the superiority of STEAM becomes more obvious with more imperfect cases.

To further analyze the robustness of different models, we compare their performance on the real test set (see Table 2) and the simulated test set (see Table 5) and calculate the performance disturbance with \( \text{dist} = (v_{\text{sim}} - v_{\text{real}}) / v_{\text{real}} \), where \( \text{dist} \) represents the disturbance, \( v_{\text{real}} \) is the metric value on the real test set, and \( v_{\text{sim}} \) is the metric value on the simulated test set. Especially, for STEAM, \( v_{\text{real}} \) is the metric value on the raw item sequences of the real test set (see Overall-R in Table 2), while \( v_{\text{sim}} \) is the metric value on the corrected sequences of the simulated test set by default. As we hope to evaluate how STEAM handles the simulated imperfections added into the real test set, we consider the performance of STEAM on the raw item sequences without correcting the inherent imperfections in the real test set.

The performance disturbance of all models is listed in Table 6. First, we can find that most performance disturbance values are minus, which illustrates that sequence imperfections will degrade model performance. Second, FMLP-Rec shows competitive robustness and performs better than STEAM on the Yelp dataset, which confirms its effectiveness at denoising. Finally, STEAM achieves most of the best results on the Beauty and Sports datasets and most of the second best results on the Yelp dataset, proving its robustness. The disturbance values of STEAM on MRR@5 and MRR@10 on the Beauty dataset are even positive, which illustrates that STEAM can not only correct the simulated imperfections but also correct some inherent imperfections. Although FMLP-Rec is more robust than STEAM on the Yelp dataset, it may sacrifice recommendation performance for robustness. Similarly, SASRec obtains better values on some metrics, but its recommendation performance is relatively inferior. In contrast, DuoRec is the best baseline in recommendation performance, but its robustness is worse than FMLP-Rec and STEAM. We conclude that STEAM strikes a better balance between recommendation performance and robustness than other methods.

### 5.3 Simulation Analysis

Table 5: Performance comparison of different methods on the simulated test sets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Simulated Beauty</th>
<th>Simulated Sports</th>
<th>Simulated Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>GRU4Rec</td>
<td>32.22</td>
<td>42.13</td>
<td>21.28</td>
</tr>
<tr>
<td>SASRec</td>
<td>35.97</td>
<td>45.26</td>
<td>24.97</td>
</tr>
<tr>
<td>BERT4Rec</td>
<td>35.83</td>
<td>46.79</td>
<td>22.79</td>
</tr>
<tr>
<td>SRGNN</td>
<td>36.64</td>
<td>46.81</td>
<td>24.50</td>
</tr>
<tr>
<td>CL4SRec</td>
<td>38.66</td>
<td>48.22</td>
<td>26.96</td>
</tr>
<tr>
<td>DuoRec</td>
<td>40.26</td>
<td>50.13</td>
<td>28.39</td>
</tr>
<tr>
<td>FMLP-Rec</td>
<td>39.38</td>
<td>48.47</td>
<td>27.85</td>
</tr>
<tr>
<td>Recommender</td>
<td>35.14</td>
<td>45.96</td>
<td>22.22</td>
</tr>
<tr>
<td>STEAM</td>
<td>42.09∗</td>
<td>52.21∗</td>
<td>28.45</td>
</tr>
</tbody>
</table>

Table 6: Robustness analysis of different models. Each value is a performance disturbance.

<table>
<thead>
<tr>
<th>Model</th>
<th>Beauty</th>
<th>Sports</th>
<th>Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>GRU4Rec</td>
<td>-2.21%</td>
<td>-1.08%</td>
<td>-1.62%</td>
</tr>
<tr>
<td>SASRec</td>
<td>-1.67%</td>
<td>-0.68%</td>
<td>-1.81%</td>
</tr>
<tr>
<td>BERT4Rec</td>
<td>-2.29%</td>
<td>-1.04%</td>
<td>-2.52%</td>
</tr>
<tr>
<td>SRGNN</td>
<td>-1.85%</td>
<td>-1.76%</td>
<td>-2.58%</td>
</tr>
<tr>
<td>CL4SRec</td>
<td>-1.60%</td>
<td>-1.09%</td>
<td>-2.28%</td>
</tr>
<tr>
<td>DuoRec</td>
<td>-1.68%</td>
<td>-1.28%</td>
<td>-1.56%</td>
</tr>
<tr>
<td>FMLP-Rec</td>
<td>-0.78%</td>
<td>-0.51%</td>
<td>-0.57%</td>
</tr>
<tr>
<td>Recommender</td>
<td>-1.65%</td>
<td>-1.10%</td>
<td>-2.71%</td>
</tr>
<tr>
<td>STEAM</td>
<td>-0.28%</td>
<td>-1.02%</td>
<td>+0.64%</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

We thank our anonymous reviewers for their helpful comments. This research was supported by the National Key R&D Program of China with grant (No.2022YFC3303004, No.2020YFB1406704), the Natural Science Foundation of China (62102234, 62272274, 62022071, 61902219, 61972234, 62072279), the Key Scientific and Technological Innovation Program of Shandong Province (2019JZZY010129), the Tencent WeChat Rhino-Bird Focused Research Program (JR-WXG-2021411), the Fundamental Research Funds of Shandong University, and the Hybrid Intelligence Center, a 10-year program funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research, https://hybrid-intelligence-centre.nl. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES


Table 7: Ablation study for self-supervised tasks on the real test sets, where STEAM-DC is the variant of STEAM trained by the deletion correction task and the masked item prediction task, STEAM-IC is the variant of STEAM trained by the insertion correction task and the masked item prediction task.

<table>
<thead>
<tr>
<th>Model</th>
<th>Real Beauty</th>
<th>Real Sports</th>
<th>Real Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>Recommender</td>
<td>35.73</td>
<td>46.47</td>
<td>22.84</td>
</tr>
<tr>
<td>STEAM-DC</td>
<td>41.56</td>
<td>51.93</td>
<td>27.94</td>
</tr>
<tr>
<td>STEAM-IC</td>
<td>41.77</td>
<td>52.23</td>
<td>28.15</td>
</tr>
<tr>
<td>STEAM</td>
<td>42.57</td>
<td>52.89</td>
<td>28.75</td>
</tr>
</tbody>
</table>

Table 8: Ablation study for self-supervised tasks on the simulated test sets.

<table>
<thead>
<tr>
<th>Model</th>
<th>Simulated Beauty</th>
<th>Simulated Sports</th>
<th>Simulated Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>Recommender</td>
<td>35.14</td>
<td>45.96</td>
<td>22.22</td>
</tr>
<tr>
<td>STEAM-DC</td>
<td>40.94</td>
<td>50.93</td>
<td>27.54</td>
</tr>
<tr>
<td>STEAM-IC</td>
<td>40.87</td>
<td>51.47</td>
<td>27.43</td>
</tr>
<tr>
<td>STEAM</td>
<td>42.09</td>
<td>52.21</td>
<td>28.45</td>
</tr>
</tbody>
</table>

Table 9: Ablation study for the 'delete' operation on the real test sets, where STEAM-DK is the variant of STEAM that executes 'delete' and 'keep' operations only.

<table>
<thead>
<tr>
<th>Model</th>
<th>Real Beauty</th>
<th>Real Sports</th>
<th>Real Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEAM-DK</td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>Changed-R</td>
<td>32.08</td>
<td>43.56</td>
<td>20.01</td>
</tr>
<tr>
<td>Changed-C</td>
<td>32.92</td>
<td>43.86</td>
<td>20.56</td>
</tr>
</tbody>
</table>

Table 10: Ablation study for the 'insert' operation on the real test sets, where STEAM-IK is the variant of STEAM that executes 'insert' and 'keep' operations only.

<table>
<thead>
<tr>
<th>Model</th>
<th>Real Beauty</th>
<th>Real Sports</th>
<th>Real Yelp</th>
</tr>
</thead>
<tbody>
<tr>
<td>STEAM-IK</td>
<td>HR@5</td>
<td>HR@10</td>
<td>MRR@5</td>
</tr>
<tr>
<td>Changed-R</td>
<td>73.36</td>
<td>79.27</td>
<td>51.62</td>
</tr>
<tr>
<td>Changed-C</td>
<td>75.29</td>
<td>80.21</td>
<td>55.98</td>
</tr>
</tbody>
</table>

A ABLATION STUDY FOR SELF-SUPERVISED TASKS

To analyze the effectiveness of each self-supervised task, we carry out experiments with two variants of STEAM, i.e., STEAM-DC and STEAM-IC. STEAM-DC is trained by the deletion correction task and the masked item prediction task. STEAM-IC is trained by the insertion correction task and the masked item prediction task. The experimental results are shown in Table 7 and 8. We observe that both STEAM-DC and STEAM-IC perform better than Recommender, so we can confirm that using the single deletion correction mechanism or insertion correction mechanism improves the sequential recommendation performance. We also see that STEAM outperforms STEAM-DC and STEAM-IC, which proves it is necessary to combine these two self-supervised mechanisms to improve model performance.

B ABLATION STUDY FOR CORRECTION OPERATIONS

To evaluate the effectiveness of each correction operation, we carry out experiments with two variants of STEAM, i.e., STEAM-DK and STEAM-IK. STEAM-DK executes 'delete' and 'keep' operations, while STEAM-IK executes 'insert' and 'keep' operations. We focus on the changed sequence group, and report the Changed-R and Changed-C of STEAM-DK and STEAM-IK on the real test sets. The results are shown in Table 9 and 10. The result of Changed-C are better than those of Changed-R, in both tables. It illustrates that both deleting items and inserting items are useful, which can make the corrected sequence better for sequential recommendation. Moreover, the results of Changed-R in Table 9 and 10 are different, which suggests that the sequences corrected by the 'delete' operation are different from those corrected by the 'insert' operation. Therefore, STEAM should employ both 'delete' and 'insert' operations to correct item sequences.