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

Zhaochun Ren
Shandong University
Qingdao, China
zhaochun.ren@sdu.edu.cn

Xin Xin
Shandong University
Qingdao, China
xinxin@sdu.edu.cn

Qiang Yan
WeChat, Tencent
Guangzhou, China
rolanyan@tencent.com

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 **S**elf-correc**T**ing s**E**quenti**A**l reco**M**mender. 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.

*Corresponding author.

WWW '23, May 1–5, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583479

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

Sequential recommendation, Sequence correction, Self-supervised learning

ACM Reference Format:
Yujie Lin, Chenyang Wang, Zhumin Chen, Zhaochun Ren, Xin Xin, Qiang Yan, Maarten de Rijke, Xiuzhen Cheng, and Pengjie Ren. 2023. A Self-Correcting Sequential Recommender. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3543507.3583479

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_6, which is a book about computer basics. As a result, the recommendation model may recommend another book about computer science, i.e.,

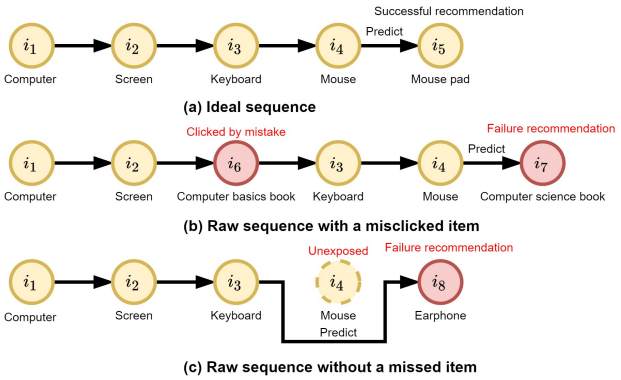

Figure 1: Examples for two kinds of imperfect item sequences. Sub-figure (a) is an ideal item sequence without any imperfection. Sub-figure (b) is an imperfect item sequence that contains a misclicked item (i.e., i_6). Sub-figure (c) is an imperfect sequence that lacks a missed item (i.e., i_4).

i_7, which is actually not interesting for the user. (ii) Some relevant items may not be exposed to the user, so the user may not be aware of them and thus will not click them. As a result, the item sequence may lack some *missed* items. For instance, in Fig. 1(c), the user cannot interact with i_4, because i_4 is not exposed to the user. Accordingly, the recommendation model may not further recommend i_5, a mouse pad, but may recommend i_8 which is an earphone. For the first kind of imperfection, misclicked items can be considered as noise in item sequences. Some studies have addressed capturing and eliminating noise from user-item interaction sequences [28, 57]. These denoising sequential methods take the raw sequence as the input to train the model and predict the next item without explicitly modifying the given sequence. Besides, they neglect the second kind of imperfection, which means they cannot recall missed and/or unexposed items.

Blindly learning a model on the raw data without considering its inherent imperfections may fail to capture a user's true preferences, and harm the user experience and downgrade recommendation performance. To this end, we enable a recommendation model to learn to correct the raw item sequence before making recommendations. There are two main challenges to realize this correction. First, the raw item sequence may mix different kinds of imperfections and contain multiple imperfections in multiple positions. Hence, the first challenge is how to simultaneously apply different and multiple correction operations to the item sequence. Second, it is difficult to identify the misclicked items and complement the missed items manually. Therefore, the second challenge is how to train the corrector model without additional labeling.

We propose a novel sequential recommendation model, called *self-correcting sequential recommender* (STEAM), which first corrects the input item sequence using a corrector, and then uses the corrected item sequence to train a recommender and make the next item prediction. Specifically, we propose an item-wise corrector that can adaptively apply 'keep', 'delete' and 'insert' operations to the items in an input item sequence. If the selected operation is 'insert', the corrector will further use a reverse generator to generate the inserted sequence which will be inserted reversely before the item. For misclicked items, the corrector can delete them, while for missed items, the corrector can insert them. We design two self-supervised tasks to generate imperfect sequences and supervised signals automatically: (i) deletion correction, and (ii) insertion correction. The former randomly inserts items to the raw item sequence and makes the model learn to 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

introduced to sequential recommendation as well. Zhou et al. [56] propose four auxiliary self-supervised tasks to maximize the mutual information among attributes, items, and sequences. Xia et al. [50] propose to maximize the mutual information between sequence representations learned via hypergraph-based GNNs. Xie et al. [51] use item crop, item mask, and item reorder as data augmentation approaches to construct self-supervision signals. Liu et al. [21] propose data augmentation methods to construct self-supervised signals for better exploiting item correlations. Qiu et al. [29] perform contrastive self-supervised learning based on dropout [33].

The studies listed above train models on the raw item sequences neglecting the fact that the raw sequences might be noisy due to users' casual click or inaccurate recommendations due to system exposure bias.

2.2 Denoising recommendation

Denoising recommendation aims to improve recommendation performance by alleviating the noisy data issue. Prior work exploits additional user behavior and auxiliary item features to identify the noisy data, such as 'skip' [7], 'dwell time' [16], 'gaze' [55], 'like' [1] and item side information [22]. These methods need extra feedback and manual labeling, which hinders their practical application. Recently, another line of work has been dedicated to eliminating the effect of noisy data without introducing external signals. Wang et al. [42] observe that noisy data usually leads to large loss values in the early stage of training, and design two adaptive loss functions to down-weight noisy samples. Wang et al. [44] propose an iterative relabeling framework to identify the noise by exploiting the self-training principle. Wang et al. [43] assume that predictions on noisy items vary across different recommendation models and propose an ensemble method to minimize the KL-divergence between the two models' predictions. Gao et al. [8] argue that the models are prone to memorize easy and clean patterns at the early stage of training, so they collect memorized interactions at the early stage as guidance for the following training.

Most denoising recommendation methods focus on non-sequential recommendation. There are a few studies targeting denoising for sequential recommendation. Qin et al. [28] design a denoising generator for next basket recommendation that is based on contrastive learning to determine whether an item in a historical basket is related to the target item. Tong et al. [36] mine sequential patterns as the prior knowledge to guide a contrastive policy learning model for denoising and recommendation. Inspired by fast Fourier transforms (FFTs) [32], Zhou et al. [57] propose an all-MLP model with learnable filters for denoising sequential recommendation.

However, existing denoising recommendation methods do not correct the raw data explicitly. Besides, they do not recall the missed items due to system exposure bias.

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 $S = [i_1, \ldots, i_{|S|}]$, where $i_t \in \mathcal{I}$ is the interacted item at the t-th position of S and $|S|$ is the sequence length. We denote a subsequence $[i_j, \ldots, i_k]$ of S as $S_{j:k}$. Especially, we denote a raw item sequence as S^r. For training STEAM with deletion and insertion correction, we randomly modify S^r and then ask STEAM to recover it. For each item in S^r, 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^r, 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 $O = [o_1, \ldots, o_{|S^m|}]$, where $o \in$ {'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 $S^{<i} = [i_1, \ldots, i_{|S^{<i}|-1}, [eos]]$, which should be inserted in reverse order before i, where $[eos] \in \mathcal{I}$ is a special token representing the ending. Note that the order is $[i_{|S^{<i}|-1}, \ldots, i_1, i]$ after insertion. We revise S^r using the corrector to get the corrected sequence, which is denoted as S^c. To train STEAM with the masked item prediction, we randomly mask some items in S^r or S^c with a special token $[mask] \in \mathcal{I}$ with probability p_m, and then ask STEAM to predict masked items. The masked S^r (S^c) and its masked items are denoted as $\widetilde{S}^r$ ($\widetilde{S}^c$) and $\widetilde{I}^r$ ($\widetilde{I}^c$).

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

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 $[mask]$ 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.

3.2 Encoder

The target of the encoder is to encode the input item sequence to get the hidden representations of all positions of the item sequence. The encoder is shared by the corrector and recommender, whose output will be the input of the two other modules.

Specifically, the encoder first maintains an item embedding matrix $\mathbf{E} \in \mathbb{R}^{e \times |\mathcal{I}|}$ to project the high-dimensional one-hot vector of an item to a low-dimensional dense vector. For each item i_t in a

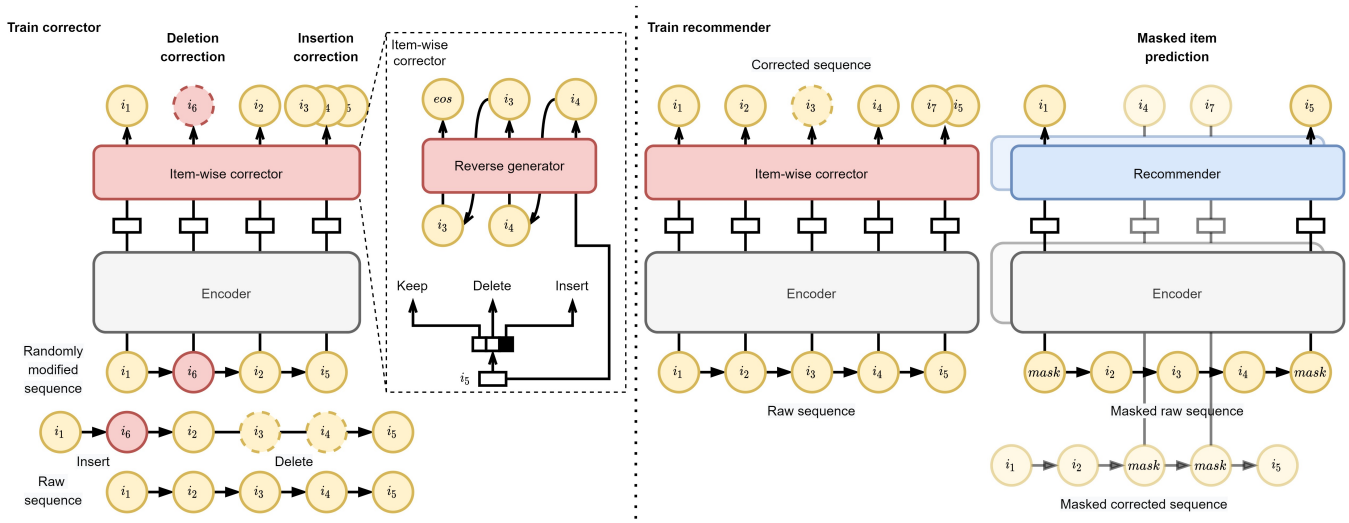

Figure 2: An overview of STEAM. For training the corrector, the item-wise corrector is asked to perform deletion correction and insertion correction on items to recover the raw item sequence that has been randomly modified. The raw item sequence with its corrected version are both used to train the recommender using the masked item prediction task. Finally, STEAM is optimized by the joint loss from the corrector and the recommender.

given sequence S, we follow Eq. 1 to project it:

$$\mathbf{e}_t = \mathbf{E}\mathbf{i}_t, \tag{1}$$

where $\mathbf{i}_t \in \mathbb{R}^{|\mathcal{I}|}$ is the one-hot vector of i_t, $\mathbf{e}_t \in \mathbb{R}^e$ is the item embedding of i_t, and e is the embedding size. We inject the position information into the model by adding the position embeddings:

$$\mathbf{h}_t^0 = \mathbf{e}_t + \mathbf{p}_t, \tag{2}$$

where $\mathbf{p}_t \in \mathbb{R}^e$ is the learnable position embedding of the t-th position and $\mathbf{h}_t^0 \in \mathbb{R}^e$ is the initial hidden representation of i_t. Moreover, we follow [15] to apply dropout to $\mathbf{h}_t^0$. We further stack the initial hidden representations of all items in S to get an initial hidden representation matrix $\mathbf{H}_e^0 \in \mathbb{R}^{|S|\times e}$.

Then, the encoder employs a bidirectional transformer with L_e layers to update $\mathbf{H}_e^0$, as shown in Eq. 3:

$$\mathbf{H}_e^l = \mathrm{Trm}_{\mathrm{bi}}(\mathbf{H}_e^{l-1}), \tag{3}$$

where $\mathrm{Trm}_{\mathrm{bi}}$ denotes a bidirectional transformer block; please refer to [34, 38] for details. $\mathbf{H}_e^l \in \mathbb{R}^{|S|\times e}$ is the hidden representation matrix at the l-th layer. $\mathbf{H}_e^{L_e}$ 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 $\mathbf{H}_e^{L_e}$ (i.e., $\mathbf{H}_e$) in the following modules.

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 $\mathbf{h}_t \in \mathbb{R}^e$ indexed from the input $\mathbf{H}_e$, we follow Eq. 4 to obtain the probability distribution $P(\hat{o}_t \mid S)$ for the corresponding correction operation o_t:

$$P(\hat{o}_t \mid S) = \mathrm{softmax}(\mathbf{W}\mathbf{h}_t), \tag{4}$$

where $\hat{o}_t$ is the predicted version of o_t, $\mathbf{W} \in \mathbb{R}^{3\times e}$ is the projection matrix. When testing, the operation with the maximum probability in $P(\hat{o}_t \mid S)$ 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 $S_{1:n-1}^{<i_t}$, the reverse generator first follows the same way in the encoder to get the item embeddings of all items in $S_{1:n-1}^{<i_t}$. Note that the item embedding matrix is shared between the encoder and the corrector.

Then, we stack the hidden representation $\mathbf{h}_t$ of i_t obtained from $\mathbf{H}_e$ with all item embeddings $\{\mathbf{e}_1, \ldots, \mathbf{e}_{n-1}\}$ of $S_{1:n-1}^{<i_t}$:

$$\mathbf{H}_c^0 = \begin{bmatrix} \mathbf{h}_t + \mathbf{p}_1 \\ \mathbf{e}_1 + \mathbf{p}_2 \\ \vdots \\ \mathbf{e}_{n-1} + \mathbf{p}_n \end{bmatrix}, \tag{5}$$

where $\mathbf{H}_c^0 \in \mathbb{R}^{n\times e}$ 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 $\mathbf{H}_c^0$ as we do in the encoder.

Next, the reverse generator uses a unidirectional transformer with L_c layers to update $\mathbf{H}_c^0$ as Eq. 6:

$$\mathbf{H}_c^l = \mathrm{Trm}_{\mathrm{uni}}(\mathbf{H}_c^{l-1}), \tag{6}$$

where $\mathrm{Trm}_{\mathrm{uni}}$ represents a unidirectional transformer block; again, please see [15, 38] for details. $\mathbf{H}_c^l \in \mathbb{R}^{n\times e}$ is the hidden representation matrix at the l-th layer. $\mathbf{H}_c^{L_c}$ is the last hidden representation matrix, and we denote it as $\mathbf{H}_c$ for short.

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

$$P(\hat{i}_n \mid S_{1:n-1}^{<i_t}, S) = \mathrm{softmax}(\mathbf{E}^\top \mathbf{h}_n), \tag{7}$$

where $\mathbf{E}$ is the item embedding matrix, $\mathbf{h}_n \in \mathbb{R}^e$ is the hidden representation at the last position of $\mathbf{H}_c$. In particulare, the first inserted item i_1 is generated based on $\mathbf{H}_c^0 = [\mathbf{h}_t + \mathbf{p}_1]$, so we define $P(\hat{i}_1 \mid S_{1:0}^{<i_t}, S) = P(\hat{i}_1 \mid S)$.

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

3.4 Recommender

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

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

$$\mathbf{H}_r^l = \mathrm{Trm}_{\mathrm{bi}}(\mathbf{H}_r^{l-1}), \tag{8}$$

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

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

$$P(\hat{i}_t \mid S) = \mathrm{softmax}(\mathbf{E}^\top \mathbf{h}_t), \tag{9}$$

where $\mathbf{E}$ is the item embedding matrix, which is also used in the encoder and the corrector. When training, the item $i_t \in \widetilde{I}^r$ ($\widetilde{I}^c$) and the sequence S is $\widetilde{S}^r$ ($\widetilde{S}^c$). When testing, the item i_t is $i_{|S^r|+1}$ ($i_{|S^c|+1}$) and the sequence S is S^r (S^c), so we can get $P(i_{|S^r|+1} \mid S^r)$ ($P(i_{|S^c|+1} \mid S^c)$) 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^r 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 $P(S^r \mid S^m)$, as shown in Eq. 10:

$$\begin{aligned} L_1 &= -\log P(S^r|S^m) \\ &= -\left(\log P(O|S^m) + \sum_{i \in I^{ins}} \log P(S^{<i}|S^m)\right) \\ &= -\left(\sum_{t=1}^{|S^m|} \log P(\hat{o}_t = o_t|S^m) + \sum_{i \in I^{ins}} \sum_{n=1}^{|S^{<i}|} \log P(\hat{i}_n = i_n|S_{1:n-1}^{<i}, S^m)\right), \end{aligned} \tag{10}$$

where L_1 is the loss for the corrector.

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

$$\begin{aligned} L_2 &= -\left(\log P(\widetilde{I}^r \mid \widetilde{S}^r) + \log P(\widetilde{I}^c \mid \widetilde{S}^c)\right) \\ &= -\left(\sum_{i \in \widetilde{I}^r} \log P(\hat{i} = i \mid \widetilde{S}^r) + \sum_{i \in \widetilde{I}^c} \log P(\hat{i} = i \mid \widetilde{S}^c)\right), \end{aligned} \tag{11}$$

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:

$$L = L_1 + L_2. \tag{12}$$

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 99 randomly sampled negative items that the user has not interacted 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 1: Statistics of the datasets after preprocessing.

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

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.

We also report "Recommender", which is a variant of the recommender in STEAM that is trained by the masked item prediction task without joint training with the corrector. For STEAM, we report its performance on corrected item sequences by default.

4.4 Metrics and implementation

We adopt two widely used evaluation metrics to evaluate the performances of all sequential recommendation methods: HR@k (hit ratio) and MRR@k (mean reciprocal rank) [6], where $k \in \{5, 10\}$.

For all baselines and STEAM, we initialize the trainable parameters randomly with Xavier method [10]. We optimize all methods with the Adam optimizer [17] for 300 epochs, with a learning rate of 0.001 and a batch size of 256. We also apply gradient clipping [27] with range $[-5, 5]$ during training. We set the maximum raw item sequence length to 50, the maximum item sequence length after correction to 60, and the maximum number of continuous inserted items to 5.

For all baselines, we follow the instructions from their original papers to set the hyper-parameters. For the hyper-parameters of STEAM, we set the embedding size e to 64, the number of heads in transformer to 1, and the number of layers L_e, L_c and L_r to 1. We set the dropout rate to 0.5. During training, the keep probability p_k, insertion probability p_i, deletion probability p_d, and mask probability p_m are set to 0.4, 0.1, 0.5, and 0.5, respectively.

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

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 $p < 0.05$.

	Real Beauty				Real Sports				Real Yelp			
Model	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
GRU4Rec	32.95	42.59	21.63	22.90	30.58	42.85	18.35	19.97	55.40	76.57	32.23	35.05
SASRec	36.58	45.57	25.43	26.62	34.51	46.20	21.91	23.46	58.24	77.96	35.07	37.72
BERT4Rec	36.67	47.28	23.38	24.79	35.16	47.91	21.54	23.24	61.18	79.72	37.64	40.13
SRGNN	37.33	47.65	25.15	26.52	35.92	48.32	22.44	24.08	59.86	78.96	36.74	39.30
CL4SRec	39.29	48.75	27.59	28.84	37.91	49.83	24.53	26.11	62.15	80.16	39.29	41.70
DuoRec	<u>40.95</u>	<u>50.78</u>	**28.84**	**30.15**	<u>39.80</u>	<u>51.93</u>	<u>25.97</u>	<u>27.58</u>	<u>64.01</u>	<u>82.63</u>	<u>40.85</u>	<u>43.34</u>
FMLP-Rec	39.69	48.72	28.01	29.20	37.67	49.32	24.66	26.21	61.85	80.76	38.38	40.92
Recommender	35.73	46.47	22.84	24.27	35.02	47.78	21.34	23.03	61.41	80.57	37.67	40.22
STEAM	**42.57***	**52.89***	<u>28.75</u>	<u>30.14</u>	**42.14***	**55.16***	**26.87***	**28.61***	**67.22***	**84.49***	**43.45***	**45.77***

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.

	Real Beauty				Real Sports				Real Yelp			
STEAM	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
Overall-R	42.21	52.75	28.27	29.68	42.03	55.04	26.75	28.48	67.19	84.49	43.42	45.75
Overall-C	42.57	52.89	28.75	30.14	42.14	55.16	26.87	28.61	67.22	84.49	43.45	45.77
Changed-R	41.35	51.59	27.04	28.40	35.04	47.64	21.56	23.23	56.05	74.19	34.36	36.80
Changed-C	42.56	52.06	28.66	29.94	35.54	48.12	22.08	23.76	57.46	74.40	35.45	37.73
Unchanged	42.58	53.25	28.79	30.22	44.21	57.36	28.37	30.12	67.44	84.71	43.62	45.95

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.

Dataset	#Changed	#Keep	#Delete	#Insert
Real Beauty	29.91	88.60	4.03	7.37
Real Sports	23.82	95.72	4.21	0.07
Real Yelp	2.17	99.63	0.15	0.22

metrics, but the superiority of Overall-C is not obvious. We think the reason is that not all test sequences are changed after correction, so we count the percentage of changed test sequences, see Table 4. We can verify that most test item sequences do not change after correction, especially on the Yelp dataset. Based on the observation in Table 4, we group test item sequences into two groups: the changed sequence group whose corrected item sequences are different from the raw item sequences, and the unchanged sequence group where the raw item sequences remain the same after correction. We evaluate the performance of STEAM on the two groups, separately. As shown in Table 3, for the changed item sequence group, we use Changed-R and Changed-C to denote the performance of STEAM on the raw item sequences and the corrected sequences, respectively. We see that Changed-C is better than Changed-R, and the difference between them is bigger than that between Overall-C and Overall-R. Therefore, we can confirm that the corrector helps the recommender achieve better performance by correcting input sequences.

Next, we compare Overall-R in Table 3 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

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

	Simulated Beauty				Simulated Sports				Simulated Yelp			
Model	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
GRU4Rec	32.22	42.13	21.28	22.59	29.96	42.26	17.99	19.61	54.64	75.87	31.66	34.49
SASRec	35.97	45.26	24.97	26.20	33.63	45.23	21.47	23.01	57.71	77.12	34.64	37.23
BERT4Rec	35.83	46.79	22.79	24.25	34.10	46.49	20.62	22.26	59.46	78.07	36.36	38.85
SRGNN	36.64	46.81	24.50	25.85	35.39	47.55	22.00	23.60	57.55	76.82	35.09	37.68
CL4SRec	38.66	48.22	26.96	28.23	37.10	48.93	23.95	25.52	61.08	78.99	38.48	40.88
DuoRec	40.26	50.13	28.39	29.71	38.87	50.95	25.36	26.96	63.06	82.07	40.24	42.78
FMLP-Rec	39.38	48.47	27.85	29.06	37.23	48.86	24.33	25.87	61.17	80.37	37.97	40.56
Recommender	35.14	45.96	22.22	23.66	33.70	46.40	20.38	22.06	60.33	79.08	36.52	39.03
STEAM	**42.09***	**52.21***	**28.45**	**29.81**	**41.72***	**54.82***	**26.43***	**28.17***	**66.46***	**84.05***	**42.83***	**45.19***

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

	Beauty				Sports				Yelp			
Model	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
GRU4Rec	-2.21%	-1.08%	-1.62%	-1.35%	-2.03%	-1.38%	-1.96%	-1.80%	-1.37%	-0.91%	-1.77%	-1.60%
SASRec	-1.67%	-0.68%	-1.81%	-1.58%	-2.55%	-2.10%	-2.01%	-1.92%	**-0.91%**	-1.08%	-1.23%	-1.30%
BERT4Rec	-2.29%	-1.04%	-2.52%	-2.18%	-3.01%	-2.96%	-4.27%	-4.22%	-2.81%	-2.07%	-3.40%	-3.19%
SRGNN	-1.85%	-1.76%	-2.58%	-2.53%	-1.48%	-1.59%	-1.96%	-1.99%	-3.86%	-2.71%	-4.49%	-4.12%
CL4SRec	-1.60%	-1.09%	-2.28%	-2.12%	-2.14%	-1.81%	-2.36%	-2.26%	-1.72%	-1.46%	-2.06%	-1.97%
DuoRec	-1.68%	-1.28%	-1.56%	-1.46%	-2.34%	-1.89%	-2.35%	-2.25%	-1.48%	-0.68%	-1.49%	-1.29%
FMLP-Rec	-0.78%	**-0.51%**	-0.57%	-0.48%	-1.17%	-0.93%	-1.34%	-1.30%	-1.10%	**-0.48%**	**-1.07%**	**-0.88%**
Recommender	-1.65%	-1.10%	-2.71%	-2.51%	-3.77%	-2.89%	-4.50%	-4.21%	-1.76%	-1.85%	-3.05%	-2.96%
STEAM	**-0.28%**	-1.02%	**+0.64%**	**+0.44%**	**-0.74%**	**-0.40%**	**-1.20%**	**-1.09%**	-1.09%	-0.52%	-1.36%	-1.22%

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 $dist = (v_{sim} - v_{real})/v_{real}$, where $dist$ represents the disturbance, v_{real} is the metric value on the real test set, and v_{sim} is the metric value on the simulated test set. Especially, for STEAM, v_{real} is the metric value on the raw item sequences of the real test set (see Overall-R in Table 2), while v_{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.

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.

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, 62202271, 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

[1] Zhi Bian, Shaojun Zhou, Hao Fu, Qihong Yang, Zhenqi Sun, Junjie Tang, Guiquan Liu, Kaikui Liu, and Xiaolong Li. 2021. Denoising user-aware memory network for recommendation. In *ACM Conference on Recommender Systems*. 400–410.

[2] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. 2021. Sequential recommendation with graph neural networks. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 378–387.

[3] Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath. 2021. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology* 12, 5 (2021), 1–32.

[4] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *International Conference on Web Search and Data Mining*. 108–116.

[5] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

[6] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. 2020. Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations. *ACM Transactions on Information Systems* 39, 1 (2020), 1–42.

[7] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. 2005. Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems* 23, 2 (2005), 147–168.

[8] Yunjun Gao, Yuntao Du, Yujia Hu, Lu Chen, Xinjun Zhu, Ziquan Fang, and Baihua Zheng. 2022. Self-guided learning to denoise for robust recommendation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1412–1422.

[9] Florent Garcin, Christos Dimitrakakis, and Boi Faltings. 2013. Personalized news recommendation with context trees. In *ACM Conference on Recommender Systems*. 105–112.

[10] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 249–256.

[11] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *International Conference on Data Mining*. 191–200.

[12] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Conference on Information and Knowledge Management*. 843–852.

[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *International Conference on Learning Representations*.

[14] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. 2018. Improving sequential recommendation with knowledge-enhanced memory networks. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 505–514.

[15] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *International Conference on Data Mining*. 197–206.

[16] Youngho Kim, Ahmed Hassan, Ryen W White, and Imed Zitouni. 2014. Modeling dwell time to predict click-level satisfaction. In *International Conference on Web Search and Data Mining*. 193–202.

[17] Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

[18] Jake Lever, Martin Krzywinski, and Naomi Altman. 2016. Points of significance: Model selection and overfitting. *Nature Methods* 13, 9 (2016), 703–705.

[19] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Conference on Information and Knowledge Management*. 1419–1428.

[20] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2021. Self-supervised learning: Generative or contrastive. *IEEE Transactions on Knowledge and Data Engineering* (2021).

[21] Zhiwei Liu, Yongjun Chen, Jia Li, Philip S Yu, Julian McAuley, and Caiming Xiong. 2021. Contrastive self-supervised sequential recommendation with robust augmentation. *arXiv preprint arXiv:2108.06479* (2021).

[22] Hongyu Lu, Min Zhang, and Shaoping Ma. 2018. Between clicks and satisfaction: Study on multi-phase user preferences and satisfaction for online news reading. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 435–444.

[23] Muyang Ma, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Huasheng Liang, Jun Ma, and Maarten de Rijke. 2022. Improving transformer-based sequential recommenders through preference editing. *ACM Transactions on Information Systems* (2022).

[24] Muyang Ma, Pengjie Ren, Yujie Lin, Zhumin Chen, Jun Ma, and Maarten de Rijke. 2019. π-net: A parallel information-sharing network for shared-account cross-domain sequential recommendations. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 685–694.

[25] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 43–52.

[26] Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2020. DeLighT: Deep and light-weight transformer. In *International Conference on Learning Representations*.

[27] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*. 1310–1318.

[28] Yuqi Qin, Pengfei Wang, and Chenliang Li. 2021. The world is binary: Contrastive learning for denoising next basket recommendation. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 859–868.

[29] Ruihong Qiu, Zi Huang, Hongzhi Yin, and Zijian Wang. 2022. Contrastive learning for representation degeneration problem in sequential recommendation. In *International Conference on Web Search and Data Mining*. 813–823.

[30] Pengjie Ren, Zhumin Chen, Jing Li, Zhaochun Ren, Jun Ma, and Maarten De Rijke. 2019. Repeatnet: A repeat aware neural recommendation machine for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4806–4813.

[31] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *The Web Conference*. 811–820.

[32] Samir S Soliman and Mandyam D Srinath. 1990. Continuous and discrete signals and systems. *Englewood Cliffs* (1990).

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[34] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Conference on Information and Knowledge Management*. 1441–1450.

[35] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *International Conference on Web Search and Data Mining*. 565–573.

[36] Xiaohai Tong, Pengfei Wang, Chenliang Li, Long Xia, and ShaoZhang Niu. 2021. Pattern-enhanced contrastive policy learning network for sequential recommendation. In *International Joint Conference on Artificial Intelligence*.

[37] Trinh Xuan Tuan and Tu Minh Phuong. 2017. 3D convolutional networks for session-based recommendation with content features. In *ACM Conference on Recommender Systems*. 138–146.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*. 6000–6010.

[39] Meirui Wang, Pengjie Ren, Lei Mei, Zhumin Chen, Jun Ma, and Maarten de Rijke. 2019. A collaborative session-based recommendation approach with parallel memory modules. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*. 345–354.

[40] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. 2021. A survey on session-based recommender systems. *Comput. Surveys* 54, 7 (2021), 1–38.

[41] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z Sheng, and Mehmet Orgun. 2019. Sequential recommender systems: Challenges, progress and prospects. In *International Joint Conference on Artificial Intelligence*. 6332–6338.

[42] Wenjie Wang, Fuli Feng, Xiangnan He, Liqiang Nie, and Tat-Seng Chua. 2021. Denoising implicit feedback for recommendation. In *International Conference on*

Web Search and Data Mining. 373–381.

[43] Yu Wang, Xin Xin, Zaiqiao Meng, Joemon M Jose, Fuli Feng, and Xiangnan He. 2022. Learning robust recommenders through cross-model agreement. In *The Web Conference*. 2015–2025.

[44] Zitai Wang, Qianqian Xu, Zhiyong Yang, Xiaochun Cao, and Qingming Huang. 2021. Implicit feedbacks are not always favorable: Iterative relabeled one-class collaborative filtering against noisy interactions. In *ACM International Conference on Multimedia*. 3070–3078.

[45] Max Welling and Thomas N Kipf. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[46] Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *International Conference on Learning Representations*.

[47] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *International Conference on Web Search and Data Mining*. 495–503.

[48] Liwei Wu, Shuqing Li, Cho-Jui Hsieh, and James Sharpnack. 2020. SSE-PT: Sequential recommendation via personalized transformer. In *ACM Conference on Recommender Systems*. 328–337.

[49] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. 2019. Session-based recommendation with graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 346–353.

[50] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Lizhen Cui, and Xiangliang Zhang. 2021. Self-supervised hypergraph convolutional networks for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 4503–4511.

[51] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding, and Bin Cui. 2022. Contrastive learning for sequential recommendation. In *International Conference on Data Engineering*. 1259–1273.

[52] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Jundong Li, and Zi Huang. 2022. Self-supervised learning for recommender systems: A survey. *arXiv preprint arXiv:2203.15876* (2022).

[53] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A simple convolutional generative network for next item recommendation. In *International Conference on Web Search and Data Mining*. 582–590.

[54] Mengqi Zhang, Shu Wu, Xueli Yu, Qiang Liu, and Liang Wang. 2022. Dynamic graph neural networks for sequential recommendation. *IEEE Transactions on Knowledge and Data Engineering* (2022).

[55] Qian Zhao, Shuo Chang, F Maxwell Harper, and Joseph A Konstan. 2016. Gaze prediction for recommender systems. In *ACM Conference on Recommender Systems*. 131–138.

[56] Kun Zhou, Hui Wang, Wayne Xin Zhao, Yutao Zhu, Sirui Wang, Fuzheng Zhang, Zhongyuan Wang, and Ji-Rong Wen. 2020. S3-rec: Self-supervised learning for sequential recommendation with mutual information maximization. In *Conference on Information and Knowledge Management*. 1893–1902.

[57] Kun Zhou, Hui Yu, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Filter-enhanced MLP is all you need for sequential recommendation. In *The Web Conference*. 2388–2399.

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.

	Real Beauty				Real Sports				Real Yelp			
Model	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
Recommender	35.73	46.47	22.84	24.27	35.02	47.78	21.34	23.03	61.41	80.57	37.67	40.22
STEAM-DC	41.56	51.93	27.94	29.32	41.33	54.48	26.22	27.97	66.82	83.97	43.08	45.39
STEAM-IC	41.77	52.23	28.15	29.54	41.19	54.06	26.14	27.85	66.87	83.75	42.96	45.36
STEAM	42.57	52.89	28.75	30.14	42.14	55.16	26.87	28.61	67.22	84.49	43.45	45.77

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

	Simulated Beauty				Simulated Sports				Simulated Yelp			
Model	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
Recommender	35.14	45.96	22.22	23.66	33.70	46.40	20.38	22.06	60.33	79.08	36.52	39.03
STEAM-DC	40.94	50.93	27.54	28.87	40.73	53.77	25.71	27.45	66.02	83.48	42.22	44.57
STEAM-IC	40.87	51.47	27.43	28.83	40.33	53.29	25.41	27.14	65.75	83.04	41.86	44.31
STEAM	42.09	52.21	28.45	29.81	41.72	54.82	26.43	28.17	66.46	84.05	42.83	45.19

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.

	Real Beauty				Real Sports				Real Yelp			
STEAM-DK	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
Changed-R	32.08	43.56	20.01	21.53	34.89	47.46	21.48	23.15	53.36	69.32	33.35	35.63
Changed-C	32.92	43.86	20.56	22.03	35.41	47.92	21.98	23.64	55.12	69.96	34.81	36.80

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.

	Real Beauty				Real Sports				Real Yelp			
STEAM-IK	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10	HR@5	HR@10	MRR@5	MRR@10
Changed-R	73.36	79.27	51.62	52.44	46.79	60.29	27.92	29.59	59.11	78.22	34.83	37.36
Changed-C	75.29	80.21	55.98	56.65	47.06	61.76	28.95	31.03	60.00	79.11	35.47	38.05

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.