Explore User Neighborhood for Real-time E-commerce Recommendation

Xu Xie[1]   Fei Sun[2]*   Xiaoyong Yang[2]   Zhao Yang[2]   Jinyang Gao[2]   Wenwu Ou[2]   Bin Cui[1♯†]

[1]*School of EECS & Key Laboratory of High Confidence Software Technologies (MOE), Peking University*

♯*Center for Data Science, Peking University & National Engineering Laboratory for Big Data Analysis and Applications*

†*Institute of Computational Social Science, Peiking University(Qingdao), China* [2]*Alibaba Group*

[1]{*xu.xie, bin.cui*}@*pku.edu.cn* [2]{*ofey.sf, xiaoyong.yxy, haoyi.yz, jinyang.gjy, santong.oww*}@*alibaba-inc.com*

Abstract—**Recommender systems play a vital role in modern online services, such as Amazon and Taobao. Traditional personalized methods, which focus on user-item (UI) relations, have been widely applied in industrial settings, owing to their efficiency and effectiveness. Despite their success, we argue that these approaches ignore local information hidden in similar users. To tackle this problem, user-based methods exploit similar user relations to make recommendations in a local perspective. Nevertheless, traditional user-based methods, like userKNN and matrix factorization, are intractable to be deployed in the real-time applications since such transductive models have to be recomputed or retrained with any new interaction. To overcome this challenge, we propose a framework called self-complementary collaborative filtering (SCCF) which can make recommendations with both global and local information in real time. On the one hand, it utilizes UI relations and user neighborhood to capture both global and local information. On the other hand, it can identify similar users for each user in real time by inferring user representations on the fly with an inductive model. The proposed framework can be seamlessly incorporated into existing inductive UI approach and benefit from user neighborhood with little additional computation. It is also the first attempt to apply user-based methods in real-time settings. The effectiveness and efficiency of SCCF are demonstrated through extensive offline experiments on four public datasets, as well as a large scale online A/B test in Taobao.**

Index Terms—**Candidate generation, User neighborhood, Real time**

I. INTRODUCTION

Enormous amounts of products have induced great challenges to keep users satisfied in modern online services, such as e-commerce and social media. Recommender systems play a crucial role in alleviating this information overload. They usually suggest a small set of items from millions (even billions) of products, which are appealing to the specific user. For large scale recommender systems, a common approach is a two-stage procedure, i.e., "candidate generation" step and "ranking" step. Candidate generation step selects a small subset of candidate items out of the entire item set for each user. Then a ranking model predicts a score for each candidate item and select the top items based on the estimated scores. This two-step procedure is widely adopted in the large scale industry recommender systems owing to its scalability and fast inference performance [1, 2, 3, 4, 5, 6, 7, 8, 9]. In this

paper, we focus on the candidate generation stage [10], which is usually referred to as the *top-N recommendation* [11] in the academic area.

To cope with the candidate generation tasks, collaborative filtering (CF) is one of the most pervasive methods. It captures users' interests from their historical interactions (e.g., clicks and purchases). Through the information it focuses on, it can broadly fall into three categories: item-based approaches [12, 13, 14], user-item (UI) approaches [10, 15, 16], and user-based approaches [17, 18, 19]. Item-based approaches rely on item-item relations which are usually stable. They compute the item similarity matrix in advance and then recommend items which are similar to what the users have interacted with. While promising, these methods compute item similarities in a user-irrelevant manner, thus restricting their performance. To tackle this problem, UI methods takes more personalized information into account. They project users and items into a shared latent space and then recommend items which are close to the user in the space.

Despite the prevalence and effectiveness of UI approaches, we argue that they only capture the global structure of the item-item relations like item-based models [20]. There could exist pairs of items whose similarities tend to be small but display a strong correlation in a specific subset of users. For example, consumers usually do not buy beer and diapers together, while new parents tend to purchase them together. Therefore, though beer and diapers exhibit weak relations in a global aspect, they display strong relations in the new parents. However, item-based methods and UI approaches project these pairs towards average position according to the whole users' interactions. This drawback restricts their performances.

To address the above challenge, we attempt to model the local relations from a user-based aspect [16, 17, 18, 21, 22]. For each user, we identify a set of users who have similar historical interactions, and then recommend items those similar users ever interacted with. Traditional user-based methods, such as UserKNN [21], identify similar users by computing user similarities directly from the original high-dimensional user interaction vectors. It is not affordable with millions even billions of items. To tackle this problem, some works factorize the user-item interaction matrix to reduce the dimension of user representations [16, 18]. However, these methods can hardly be applied in real-world applications, since they usually
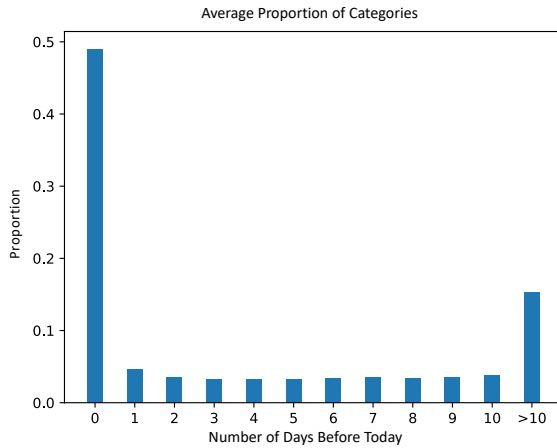
*Corresponding author.

Figure 1. The average distribution of categories which a user clicks today and also clicked in the last two weeks. Each value at the day x denotes how many categories she first clicked at x days before today.

fail to infer users' new preference in time.

To illustrate the necessity of the real-time issue, we analyze the changes of users' interests on the Taobao platform. We reflect this change by recording the proportion of categories that users click today and also clicked in the last two weeks. For each user, we record the number of categories that she clicks today. And then, for each of these categories, we look for the day she first clicked this category in the last two weeks. The results are visualizing in Figure 1. The value of x-axis denotes the number of days before today. For example, the bar of $x = 0$ demonstrates those categories are only clicked today and they were not clicked in the last two weeks. The bar of $x = 4$ demonstrates those categories, which the user clicks today, are first clicked at 4 days before today. We observe that most of the categories, around 50%, that users click today are new categories. Therefore, users' interests are sharply changing and we need to adapt to users' dynamic interests in real time to make better recommendations. However, when encountering with users' new interactions, these transductive models need to refactorize the new user-item matrix to infer new user representations, which is not affordable in the real-world services.

To tackle this challenge, we attempt to infer user representations in real time and then identify the similar users in real time. With the recent advances of latent factor and deep learning techniques [10, 15, 23], it is easy to update user vectors with new interactions in real time. With the dynamically inferred user representations, users' neighborhood can be identified timely via efficient similarity search tools like `Faiss` [24]. Specifically, we propose a general framework called **S**elf-**C**omplementary **C**ollaborative **F**iltering (**SCCF**) which captures both global and local relations in real-time settings. There are three components in SCCF. UI component derives user and item representations which encode complex global relations. Then without any auxiliary parameters, user-based component identifies each user's appropriate neighborhood with the above representations, thus discovering the local patterns shared by her similar users. Finally, a neural network fuses the aforementioned results and derives the candidate list. Hence, with the SCCF framework, we can complement the UI models with the local information hidden in user-user relations. In addition, our SCCF framework can be regarded as a post-processing plugin to any inductive UI models whether shallow latent factor models or deep learning models, and whether sequential or non-sequential models.

Our primary contributions can be summarized as follows:

- We revisit the user-based methods and reveal that it can enhance broadly used UI methods in a local aspect.
- Our framework addresses the real-time recommendation issues of traditional user-based approaches. To the best of our knowledge, we are the first to address this problem in the literature.
- We propose a framework called Self-Complementary Collaborative Filtering (SCCF), which can be seamlessly incorporated into inductive UI models and capture both global and local information.
- Extensive experiments on four public datasets demonstrate the effectiveness and efficiency of SCCF framework against state-of-the-art UI methods and user-based methods.
- We analyze the change of users' interests and also conduct an online A/B test on our e-commercial platform—Taobao. The results demonstrate the significance of capturing users' dynamic interests and verify the effectiveness of our SCCF framework.

The rest of the paper is organized as follows. We first list the relevant existing methods in section 2. Then, in section 3, we represent the details of SCCF framework and explain how to employ it to existing inductive UI models. We provide the evaluation methodology, the datasets characteristics, and the experimental results on both public datasets and our online platform in secion 4 to support our claims. Finally, we provide some concluding remarks in section 5.

II. RELATED WORK

In this section, we review CF algorithms in the literature. We classify them into three categories, including item-based models, user-item models, and user-based methods.

A. Item-based Methods

Item-based models [12, 13, 25] estimate a user's preference on an item via measuring its similarity to the user's historical interacted items. Since the item-item relations are usually stable, the item similarity table can be pre-built offline. Then recommendations can be made in real time with just a series of look-up operations. Owing to their efficiency and scalability, this type of methods is widely adopted in industry environments [26, 27, 28].

There are two kinds of item-based methods including memory-based and model-based. Memory-based methods calculate item similarities directly from the historical interactions. They produce results fast with a sacrifice on recommendation quality. To tackle this problem, model-based methods introduce learnable parameters to add the model capacity. For example, Ning and Karypis [14] introduce an aggregation coefficient matrix learned by an linear regression to estimate item similarity (SLIM). Christakopoulou and Karypis [29] extend SLIM by considering local item relations (GLSLIM). Different from our SCCF which identifies each user's dynamic neighbors, GLSLIM clusters users into fixed groups, thus restricting its performance and failing to adapt to users' dynamic tastes. Later, Barkan and Koenigstein [30] equip item-based methods with word2vec [31, 32]. It models items as latent vectors and computes item similarities by their inner-products.

B. User-Item Representation Methods

Item-item relations are usually user-independent, thus restricting the performance of item-based methods. To tackle this problem, UI methods [33, 34, 35] focus on exploiting user-item relations and provide user-relevant recommendations. Early works of UI methods leverage latent factor methods, such as Matrix Factorization [21, 36, 37]. For example, Paterek [37] projects users and items into a common latent space and then predicts the preference with the user-item inner-product (NSVD). Kabbur et al. [15] improve NSVD by excluding diagonal entries to avoid the self-similarity problem (FISM). SVD++ [16] captures users' interests more thoroughly by representing each user with user_id and her/his interactions.

Recently, deep learning has been widely applied in recommender systems [10, 38]. These works can be broadly classified into two categories: matching function based methods and representation based methods. Matching function based methods learn simple user/item representations and then utilize a complex matching model to capture user/item relations [38, 39]. Their high latency in the matching step is not affordable in the real-world scenario. Representation based methods focus on utilizing complex models to acquire meaningful user/item vectors, while their matching functions are usually very simple, like, inner-product, which is easy to be applied in real-world applications. For example, Xue et al. [40] extend MF with neural network to derive meaningful vectors (DeepMF). Chen et al. [41] adopt attention [42] mechanism to extract different weights on users' historical items and then obtain better user representations. Meanwhile, some methods attempt to leverage sequence information in users' behaviors [23, 43, 44]. For example, Tang and Wang [45] propose a Convolutional Sequence Model (Caser) to learn sequential patterns using both horizontal and vertical convolutional filters. Kang and McAuley [23] model long-term sequential semantics by self-attention mechanism (SASRec), which achieves state-of-the-art performance. In this paper, we focus on the Representation based manner. We adopt the FISM and SASRec as our base model to evaluate the performance of our SCCF framework.

C. User-based Methods

Despite the success of UI methods, they just consider global item relations and neglect local correlations in the specific user neighborhood. To tackle this problem, user-based methods [17, 21] seek to leverage user neighborhood to capture local patterns. They first identify a set of similar users for each user and then recommend items that these similar users prefer. Recently, many works attempt to enhance traditional user-based methods by introducing learnable parameters. For example, Sedhain et al. [18] propose a variant of SLIM (LRec) which learns a user-user similarity matrix and achieves better performance than traditional methods. Many works [19, 21] also attempt to exploit latent factor models to estimate the user-user similarity through user representation embedding. In addition, a few works attempt to incorporate user-based methods with deep learning approaches. For example, Zhu et al. [46] leverage auto-encoders to jointly learn user-user and item-item correlations (JCA). However, JCA uses high dimensional sparse vectors as the input for users and items, which restrics it from being applied in the large scale industry scenario. Later, Ebesu et al. [22] equip collaborative filtering with Memory Network (CMN) [47], which also takes user neighborhood into consideration.

Despite the success of the above user-based methods, they fail to be applied in a real-time scenario where user-user relations are usually not stable [26]. Users' interests usually change sharply overtime. Therefore, the similar users are also dynamic for each user. To adapt to users' dynamic interests with their new interactions and identify new user-user relations, these methods need to be retrained since most of the above user-based methods are transductive models. However, retraining a model in a streaming manner is usually infeasible in real-world applications, thus restricting them from real-time candidate generation tasks. Though CMN is an inductive model, it regards someone as the neighbor if she/he has any connection with the specific user's interacted items, which is also not affordable in real-world applications. Different from the above methods, our SCCF addresses the real-time problems existing in the user-based literature by leveraging user neighborhood through the inductive UI model's representations.

III. SELF-COMPLEMENTARY COLLABORATIVE FILTERING

In this section, we propose our framework SCCF as illustrated in Figure 2. In general, it consists of three components: a UI model, a user-based component, and an integrating component. The UI model aims to obtain global user/item representations from users' historical behaviors. The user-based component then discovers the local patterns through the above representations. Finally, the integrating component fuses the above patterns and derives the candidate list. We first introduce notations used in the following parts and propose the research problem. Then we introduce SCCF in detail.
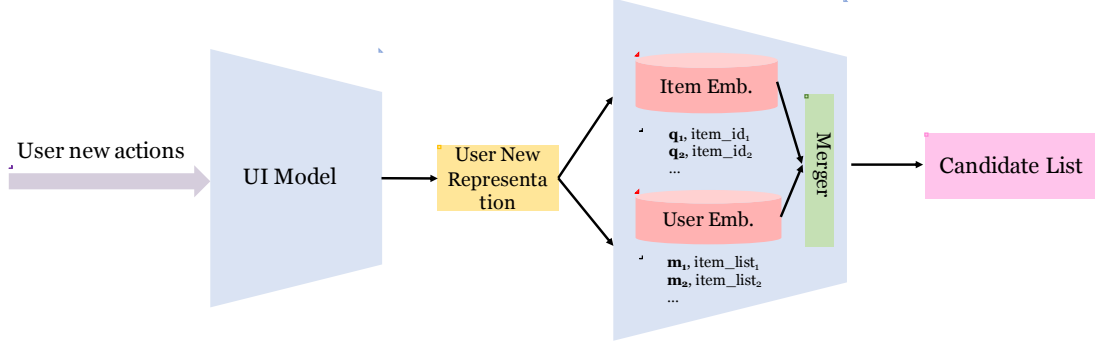
Figure 2. The pipeline of our proposed SCCF framework. When a user interacts with new items, the UI model of our SCCF first infers her/his new representation on the fly. Then our SCCF suggests two candidate lists in both global and local aspects. Finally the integrating component fuses together.

A. Notations

In this paper, all column vectors are represented by bold italic lower case letters (e.g., $\boldsymbol{p}$, $\boldsymbol{q}$). All matrices are represented by bold upper case letters (e.g., $\boldsymbol{R}$). The i-th row of a matrix $\boldsymbol{R}$ is represented by $\boldsymbol{R}_i^\top$. We use calligraphic letters to denote sets (e.g., $\mathcal{U}$, $\mathcal{I}$).

$\mathcal{U}$ and $\mathcal{I}$ denote the user/item set respectively, with $|\mathcal{U}| = n$ and $|\mathcal{I}| = m$. Our framework can be applied to non-sequential models and sequential models. For non-sequential models, $\boldsymbol{R} \in \{0,1\}^{m \times n}$ is used to represent the user-item implicit feedback matrix. $\mathcal{R}_u^+$ denotes the set of items that user u has interacted with and $\mathcal{R}_u^-$ is defined as $\mathcal{I} - \mathcal{R}_u^+$. For sequential models, $\mathcal{S}_u = [i_1^{(u)}, i_2^{(u)}, \ldots, i_{|\mathcal{S}^u|}^{(u)}]$ represents the interaction sequence in chronological order for the user u. And $i_t^{(u)}$ denotes the item which user u interacts at the time step t.

To handle the candidate generation task, we aims to predict the most possible N items which the user u will click/purchase at the time step $|\mathcal{S}_u| + 1$.

B. UI model

1) Infer User Representations: Since our SCCF has no special requirements for the inductive UI methods, we take two typical models as cases to introduce our framework, latent factor model FISM [15] and deep sequential recommendation model SASRec [23].

FISM. Different from traditional matrix factorization methods, FISM just leverages users' history interactions to predict their preference without user identifier information. It generates the representation for each user by aggregating the vectors of items which the user have interacted with. Then, it predicts the recommendation score for a user on an unrated item by calculating the inner-product of their vectors. For the input layer, one-hot encoding is applied to represent the ID feature of $\mathcal{R}_u^+$. And then, we adopt an embedding mechanism to map the sparse one-hot vector to an informative and lower-dimensional vector. In this way, for each item $i \in \mathcal{R}_u^+$, we project it to an embedding vector $\boldsymbol{p}_i \in \mathbb{R}^d$. Therefore, the output of the input layer is a set of vectors $\mathcal{P}_u = \{\boldsymbol{p}_i | i \in \mathcal{R}_u^+\}$.
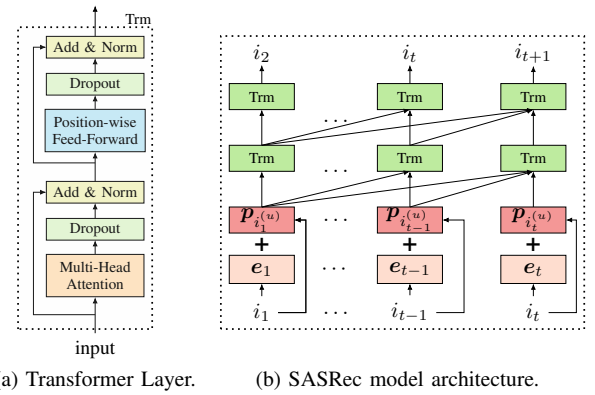

(a) Transformer Layer. (b) SASRec model architecture.

Figure 3. Brief architecture of Transformer Encoder Layer and SASRec model.

Then FISM copes with the vector set $\mathcal{P}_u$ as follows:

$$\boldsymbol{m}_u = \frac{1}{(|\mathcal{R}_u^+|)^\alpha} \sum_{j \in \mathcal{R}_u^+} \boldsymbol{p}_j, \tag{1}$$

where α is the normalization hyper-parameter that controls the degree of agreement between the items in $\mathcal{P}_u$[15]. If α is set to 1, it becomes the average pooling. In contrast, with the value of 0, it turns to sum pooling .

SASRec is different from FISM, since it utilizes sequential information when making recommendations [23]. Equipped with Transformer encoder, SASRec is a left-to-right unidirectional model which predicts next item sequentially. Transformer encoder consists of three sub-layers, an *Embedding* layer, a *Multi-Head Self-Attention* sub-layer and a *Position-wise Feed-Forward Network* sub-layer. The architectures of Transformer Encoder Layer and SASRec model are illustrated in Figure 3.

Embedding Layer. SASRec leverages position embedding $\boldsymbol{E} \in \mathbb{R}^{k \times d}$ to represent position information and it is added to the input embedding as:

$$[\hat{\boldsymbol{S}}_u]_i^\top = \boldsymbol{p}_{i_t^{(u)}} + \boldsymbol{e}_t, \tag{2}$$

2467

where $\hat{\boldsymbol{S}}_u$ is the output matrix of Embedding Layer. Here, we adopt the learnable position embedding for better performance. However, it also induces a restriction on the maximum sequence length L. To alleviate the problem, if $|\mathcal{S}_u| > L$, we truncate $\mathcal{S}_u$ to the last L items:

$$\mathcal{S}_u = \left[i^{(u)}_{|\mathcal{S}_u|-L+1}, i^{(u)}_{|\mathcal{S}_u|-L+2}, \cdots, i^{(u)}_{|\mathcal{S}_u|} \right]. \qquad (3)$$

Multi-Head Self-Attention. Multi-head Self-attention [48] focuses on integrating information from different subspaces at each position. It consists of scaled dot-product attention mechanism, self-attention mechanism, and multi-head mechanism. The scaled dot-product attention is defined as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}} \right) \boldsymbol{V}, \qquad (4)$$

where $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ represent the queries, the keys, and the values, respectively. When it comes to self-attention method, it leverages the same object and converts it to the query, the key, and the value with different linear projections as:

$$\boldsymbol{H} = \text{SA}(\hat{\boldsymbol{S}}_u) = \text{Attention}(\hat{\boldsymbol{S}}_u \boldsymbol{W}^Q, \hat{\boldsymbol{S}}_u \boldsymbol{W}^K, \hat{\boldsymbol{S}}_u \boldsymbol{W}^V), \quad (5)$$

where $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V \in \mathbb{R}^{d \times d}$ are learnable parameters. On the basis of self-attention, multi-head self-attention first projects $\hat{\boldsymbol{S}}_u$ into h subspaces with different linear projections, then applies self-attention in parallel, and finally produces the output by concatenating the intermediates and projecting it once more.

Position-wise Feed-Forward Network. Though Multi-head Self-attention can aggregate all previous items' information, it is mainly based on linear projections. To endow the model with nonlinearity and interactions between different dimensions, we apply a Position-wise Feed-Forward Network to each position of the above sub-layer's output with sharing parameters:

$$\begin{aligned} \text{PFFN}(\boldsymbol{H}) &= \left[\text{FFN}(\boldsymbol{h}_1)^\top; \text{FFN}(\boldsymbol{h}_2)^\top; \ldots; \text{FFN}(\boldsymbol{h}_{|\mathcal{S}_u|})^\top \right] \\ \text{FFN}(\boldsymbol{h}) &= \text{RELU}(\boldsymbol{h}\boldsymbol{W}_1 + \boldsymbol{b}_1)\boldsymbol{W}_2 + \boldsymbol{b}_2. \end{aligned} \qquad (6)$$

Stacking More Blocks. It is commonly beneficial to learn more complex item transition patterns by stacking more blocks. However, it becomes more difficult to converge as it goes deeper. We employ several mechanisms to alleviate this problem [48], a residual connection, a layer normalization, and a dropout module as follows:

$$\text{LayerNorm}(\boldsymbol{x} + \text{Dropout}(\text{sublayer}(\boldsymbol{x}))), \qquad (7)$$

where sublayer$(\cdot)$ is the above sub-layer operations, including multi-head self-attention and feed-forward network.

After several Transformer blocks, the output at each position has already extracted information from previous items. Since we intend to predict the item which the user u will interact with at the timestamp $|\boldsymbol{S}_u| + 1$, we set the representation for user u as:

$$\boldsymbol{m}_u = [\text{Trm}^l(\mathcal{S}_u)]^\top_{|\boldsymbol{S}_u|}, \qquad (8)$$

where Trm is the abbreviation of the Transformer encoder mentioned above, l is the number of Trm layers, and $\boldsymbol{m}_u$ is the $(|\boldsymbol{S}_u|)$-th row of the output.

2) Parameters Learning: The above parameters need to be learned to fit the users' historical interactions. To fit the users' behaviors, different types of models adopt different strategies to derive training instances. For non-sequential models, they usually predict each item $i \in \mathcal{R}^+$ with other items in $\mathcal{R}^+$. When it comes to sequential recommendation models, they usually train the model by predicting the next item for each position in the input sequence. Specifically, the target of the input sequence $[v_1, \cdots, v_{|\mathcal{S}_u|-1}]$ is $[v_2, \cdots, v_{|\mathcal{S}_u|}]$ which is a shifted version of the input. However, in the implicit feedback setting, we only have examples of what users prefer, but no examples of which kind of items they dislike, thus making it difficult to train the models. To tackle this problem, we follow the settings in [23, 39] to train the model with negative sampling and view the task as a binary classification problem. In this way, we treat the observed user-item interactions as positive instances and sample negative instances from the remaining unobserved ones.

Then the learning object is as follows:

$$\mathcal{L} = -\frac{1}{M} \Big(\sum_{(u,i) \in \mathcal{R}^+} \log \sigma(\hat{r}^{\text{UI}}_{ui}) + \sum_{(u,j) \in \text{Neg}} \log(1 - \sigma(\hat{r}^{\text{UI}}_{uj})) \Big) + \lambda \|\boldsymbol{\Theta}\|^2, \tag{9}$$

where M denotes the number of total training instances, $\text{Neg} \subset \mathcal{R}^-$ denotes the negative sample set, $\hat{r}^{\text{UI}}_{ui}$ denotes the the vector dot product of user u and item i, and σ is the sigmoid function that converts the output to a probability. The hyper-parameter λ controls the strength of ℓ_2 regularization to prevent overfitting. To optimize the objective function, we adopt mini-batch Adam [49], a variant of Stochastic Gradient Descent (SGD).

3) Candidate generation: After inducing user representations, we can provide a personalized top-N candidate list from the UI aspect. It contains two stages, calculating the preference on the whole item set for each user and then selecting the top-N items that are most likely to be clicked/purchased.

UI methods compute the preference for a user u on an unobserved item i (denoted by $\hat{r}_{ui}$) with dot product as follows:

$$\hat{r}^{\text{UI}}_{ui} = \boldsymbol{m}_u^\top \boldsymbol{q}_i, \qquad (10)$$

where $\boldsymbol{m}_u$ and $\boldsymbol{q}_i$ are embedding for user u and embedding for item i, respectively. And $\hat{r}^{\text{UI}}_{ui}$ can be regarded as the correlation between user u and item i. Note that in some work [15], there exist two kinds of item embedding, input item embedding $\boldsymbol{p}_i$ and output item embedding $\boldsymbol{q}_i$ to add the model capacity. In our framework, to reduce the model size and alleviate overfitting, we utilize a homogeneous item embedding, where $\boldsymbol{q}_i$ is equivalent to $\boldsymbol{p}_i$ like SASRec [23].

Then given a specific user u, to suggest a small set of relevant candidates from a tremendous pool of items, we calculate the preference $\hat{r}^{\text{UI}}_{ui}$ for each item i in the set $\mathcal{R}^-_u$. Then, we generate a candidate set by selecting the highest N items from them.

C. User-based Component

1) Candidate generation: As elaborated above, we have already acquired meaningful user representations and derived a candidate set from the global aspect. In the second step, we concentrate on the perspective of local neighborhood information and recommend the products which the user u's neighbors prefer.

In this way, user-based component first computes the similarities between each user as follows:

$$\text{sim}(u,v) = \cos(\boldsymbol{m}_u, \boldsymbol{m}_v). \tag{11}$$

Then it can detect a few similar neighbors for each user to construct her/his neighborhood. We denote the neighborhood as $\mathcal{N}_u = \{v_1, v_2, \ldots, v_\beta\}$, where β is a hyper-parameter. Note that $u \notin \mathcal{N}_u$ and neighbors in $\mathcal{N}_u$ are ordered descending by similarity scores.

With the information of neighborhood $\mathcal{N}_u$, we can derive another personalized top-N candidate list from the local aspect. We set the user similarity $\text{sim}(u,v)$ as the weight for the items user v clicks/purchases, calculate another preference score on the whole item set, and then select the top-N items as candidates:

$$\hat{r}_{ui}^{\text{UU}} = \sum_{v \in \mathcal{N}_u} \delta_{vi}\text{sim}(u,v) \tag{12}$$

$$\delta_{vi} = \begin{cases} 1, & i \in \mathcal{R}_v^+, \\ 0, & i \notin \mathcal{R}_v^+. \end{cases}$$

where $\hat{r}_{ui}^{\text{UU}}$ denotes the preference of the user u for item i in the local perspective. Furthermore, we assume that user u will not click items in $\mathcal{R}_u^+$ once more, so we do not recommend items in $\mathcal{R}_u^+$.

2) Complexity Analysis: Here, we discuss why previous user-based models are not affordable in real-time scenarios, but our SCCF user-based component can adapt to it.

Neighborhood user-based models. The core of neighborhood user-based models is to estimate a user-user similarity matrix which is measured as follows:

$$\text{sim}(u,v) = \frac{|\mathcal{R}_u^+ \cap \mathcal{R}_v^+|}{|\mathcal{R}_u^+||\mathcal{R}_v^+|} \tag{13}$$

Then they follow the equation 12 to obtain the prediction. Calculating the similarity matrix is usually time consuming since it needs to compute similarities of high dimensional sparse vectors. And its complexity is growing linearly with the number of items, which is not scalable. Meanwhile, the neighborhood user-based models are transductive models. If a user interacts with new items, these models need to recompute the similarity matrix, whose high latency is not tolerated in real-world services.

Factorized user-based methods. Factorized user-based methods factorize the user-item matrix to acquire user representations. Then they obtain the users' preference similar to our SCCF user-based component. However, the factorized methods are transductive models as well. If a user

clicks/purchases a new item, these methods need to re-factorize the new user-item matrix to adapt to the latest user interest, which is infeasible in real-world services.

Our SCCF is based on an inductive UI model which can derive user representations from new behaviors by inference, not training. Therefore, when a user clicks a new item, the user embedding can be inferred without retraining in real-time, thus satisfying the need for online recommendation tasks.

D. The Integrating Component

In this subsection, we discuss how to integrate the two candidate sets and select the final top-N candidates. The two candidate lists for user u are as follows:

$$\begin{aligned} \mathcal{C}_{\text{UI}}^u &= \{c_1^{\text{UI}}, c_2^{\text{UI}}, \ldots, c_N^{\text{UI}}\} \\ \mathcal{C}_{\text{UU}}^u &= \{c_1^{\text{UU}}, c_2^{\text{UU}}, \ldots, c_N^{\text{UU}}\}, \end{aligned} \tag{14}$$

which are ranked from the highest to the lowest by $\hat{r}_{ui}^{\text{UI}}$ and $\hat{r}_{ui}^{\text{UU}}$, respectively. Then, given a user u, it predicts the final score for all items in the two candidate sets as follows:

$$\hat{r}_{ui}^{\text{fi}} = f(\boldsymbol{m}_u, \boldsymbol{q}_i, \hat{r}_{ui}^{\text{UI}}, \hat{r}_{ui}^{\text{UU}}), \tag{15}$$

where the final score $\hat{r}_{ui}^{\text{fi}}$ takes both global and local information into account. We apply a multi-layer fully connected neural network to fuse the above features. The input of it is the concatenation of four features:

$$\begin{aligned} \boldsymbol{input}_{ui} &= [\boldsymbol{m}_u \oplus \boldsymbol{q}_i \oplus \hat{r}_{ui}^{\text{UI}} \oplus \hat{r}_{ui}^{\text{UU}}] \\ \tilde{r}_{ui}^{\text{UI}} &= (\hat{r}_{ui}^{\text{UI}} - \overline{\hat{r}_{ui}^{\text{UI}}})/\sigma(\hat{r}_{ui}^{\text{UI}}) \\ \tilde{r}_{ui}^{\text{UU}} &= (\hat{r}_{ui}^{\text{UU}} - \overline{\hat{r}_{ui}^{\text{UU}}})/\sigma(\hat{r}_{ui}^{\text{UU}}), \end{aligned} \tag{16}$$

where $\oplus$ is the concatenation operation, $\hat{r}_{ui}^{\text{UI}}$ and $\hat{r}_{ui}^{\text{UU}}$ are all normalized preference scores, and the mean and standard deviation are calculated for each user u. Then we stack several fully connected layers to the input embedding.

We optimize the network parameters on the following objective:

$$\mathcal{L}_{\text{I}} = -\sum_{u \in \mathcal{U}} \frac{1}{|\mathcal{C}_I^u|} \Big(\sum_{i \in \mathcal{I}_u^+} \log \sigma(\hat{r}_{ui}^{\text{fi}}) + \sum_{i \in \mathcal{I}_u^-} \log(1 - \sigma(\hat{r}_{ui}^{\text{fi}})) \Big) + \lambda \|\Theta\|^2 \tag{17}$$

where the set $\mathcal{C}_I^u = \mathcal{C}_{\text{UI}}^u \cup \mathcal{C}_{\text{UU}}^u$ denotes all the items in the two candidate sets, the item i_u^+ denotes the exact item which user u clicks at the time $|\mathcal{S}_u| + 1$, $\mathcal{I}_u^+ = \{i_u^+\} \cap \mathcal{C}_I^u$ denotes the positive instance set, and $\mathcal{I}_u^- = \mathcal{C}_I^u - \mathcal{I}_u^+$ denotes the negative instance set. If $i_u^+ \notin \mathcal{C}_I^u$, we will not calculate its two preference scores. Therefore, we do not use this instance to train our integrating model. This design exploits the four features in a fine grained, thus achieving well performance.

IV. EXPERIMENTS

In this section, we conduct plenty of experiments to answer the following research questions:

- **RQ1.** How does our proposed SCCF framework perform compared to the state-of-the-art baselines?
- **RQ2.** Do the gains rely on the difference between UI methods results and user-based components results?

Dataset	#users	#items	#actions	avg.length	density
ML-1M	6040	3416	1.0M	163.5	4.79%
ML-20M	138493	26744	20M	144.4	0.54%
Games	29341	23464	0.3M	9.1	0.04%
Beauty	40226	54542	0.4M	8.8	0.02%

- **RQ3.** How does our user-based component in SCCF perform in the real-time settings?
- **RQ4.** How do the key hyper-parameters impact SCCF performance?
- **RQ5.** How does our SCCF perform on the e-commercial platform?

A. Experiment Settings

1) Datasets: We evaluate our methods on four publicly datasets.

- **MovieLens**. The first two datasets are both MovieLens [50] datasets, which are widely used for evaluating recommendation algorithms. We use the version MovieLens-1M [1] (**ML-1M**) and MovieLens-20M [2] (**ML-20M**) that include 1 million and 20 million user ratings, respectively.
- **Amazon**[3]. The other two datasets are from amazon. They are the series of datasets introduced in [51], which are split by top-level product categories in amazon and are notable for their high sparsity and variability. In this work, we follow the settings in [23] and adopt two categories, "Beauty" and "Games".

For dataset preprocessing, we follow the common practice in [19, 23, 52]. We convert all numeric ratings or presence of a review to "1" and others to "0". It is worth mentioning that we follow the preprocessing procedure in [23, 52] for amazon datasets, which discards users and items with fewer than 5 related actions. And then to guarantee each user with enough interactions, we discard users with fewer than 5 actions once more. The processed data statistics are summarized in Table I.

2) Evaluation: We adopt the leave-one-out strategy to simulate the real-world applications and test our performance like [23, 38, 39]. For each user, we hold out the latest interaction as the test data, treat the item just before the last as the validation set and utilize others for training. Since we handle candidate generation tasks, we evaluate our model on the whole item set, sort the scores, and obtain the top N highest items as the final results. The performance is judged by Hit Ratio (HR) [25] and Normalized Discounted Cumulative Gain (NDCG) [53].

HR@k measures the proportion of cases that the desired item is among the top-k items in test cases. It is computed as:

$$\text{HR@}k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \mathbb{1}(\text{rank}_{u,g_u} \leq k)$$

where g_u is the ground truth item for user u, rank_{u,g_u} is the rank generated by the model for item g_u and user u, and $\mathbb{1}$ is an indicator function. HR@k does not consider the actual rank of g_u as long as it is among the top-k.

NDCG@k is a position-aware metric which assigns larger weights on higher ranks. For next item recommendation, it is calculated as :

$$\text{NDCG@}k = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{2^{\mathbb{1}(\text{rank}_{u,g_u} \leq k)} - 1}{\log_2(\text{rank}_{u,g_u} + 1)}$$

In this work, we report HR and NDCG with $k = 20, 50, 100$.

3) Baselines: To verify the effectiveness of SCCF, we validate it based on a diverse range of inductive UI models. We select FISM [15] and SASRec [23] as our UI component.

- **FISM**. This is one of widely used shallow latent factor models. We use negative sampling to optimize the model [39].
- **SASRec**. This method is one of the state-of-the-art deep sequential models. It exploits the sequential information and utilizes self-attention modules to capture users' dynamic interests.

We also compare it with the following baselines:

- **Pop**. This is a non-personalized benchmark. It ranks items by their popularity—the number of interactions.
- **ItemKNN** [12] and **UserKNN** [21]. They are traditional item-based CF and user-based CF methods proposed by Sarwar et al.. We use cosine similarity to derive the item or user similarity matrix.
- **BPR-MF** [54]. BPR-MF optimizes matrix factorization with the pairwise Bayesian Personalized Ranking loss.

4) Implementation Details and Parameter Settings: We implement BPR-MF, FISM, SASRec, and our SCCF with `TensorFlow`[4]. We follow He et al. [39] to train FISM, which forms a mini-batch from all interactions of a randomly sampled user, rather than randomly sampling a fixed number of interactions. Meanwhile, we train FISM without regularization and if overfitting is observed, we cope with it with early stop strategy like [39]. However, when training SASRec model, we follow the same settings as Kang and McAuley [23], which uses dropout mechanism to avoid overfitting.

To train the integrating model, we utilize each user's item just before the last as the training label. Then, we randomly split ten percent of the whole users as the validation set to tune the integrating model. We apply the early stop strategy to avoid overfitting. When measuring the performance on the test set, we add all validation items and users back to the training set, and derive the final model parameters.

All parameters are initialized by truncated normal distribution in the range $[-0.01, 0.01]$. We optimize the model by Adam optimizer with the learning rate of 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and linear decay of the learning rate. For the embedding sizes, we test the values of $[16, 32, 64, 128]$. Meanwhile, we test the effect of the number of user neighbors

β of $[50, 100, 200]$. We set $\alpha = 0.5$ for FISM model. And the SASRec parameter settings are the same as Kang and McAuley [23] with the layer number of 2 and head number of 1. For fair comparison, we use the same maximum sequence length as in [23], for ML-1M and ML-20M $N = 200$, and for Games and Beauty $N = 50$. Since users' interests are dynamically changed, for FISM method, we leverage the recent 15 items to infer user embeddings. And then for both methods, we recommend each user's latest 15 items to her/his similar users for user-based component.

B. Effectiveness of SCCF Framework(RQ1)

We now compare the performance of our SCCF with the above baselines. Table II summarizes the best results of all models on four datasets. Note that the improvement columns are the performance of SCCF relative to its base UI components. It can be observed that:

- The non-personalized methods, Pop and ItemKNN methods, exhibit the worst performance since they ignore users' unique interests. FISM and BPR-MF outperform the above methods since they provide personalized recommendations. Among all the baselines, SASRec achieves the state-of-the-art performance on all datasets, since it leverages sequence information and utilizes powerful self-attention mechanism.

- Despite direct training is conducted on the UI models, user-based component is capable of achieving comparable performance. When it comes to FISM baseline, the metric of user-based component is even higher than original FISM on all datasets. In contrast, for SASRec baseline, it outperforms $SASRec_{UU}$ on all the datasets. It may be the fact that SASRec captures more complex item semantic relations than FISM.

- According to the results, it is obvious that our proposed framework SCCF outperforms all the UI models on all datasets in terms of all evaluation metrics. $SASRec_{SCCF}$ gains 8.90% HR@20, 1.94% HR@100, 12.75% NDCG@20, and 7.42% on NDCG@100 on average against the start-of-the-art baseline SASRec. Our SCCF framework also obtains dramatically remarkable improvements over the FISM model on all datasets. The improvements are even 16.14% on HR@100 and 30.84% on NDCG@100 on average. These experiments verify the effectiveness of our user-based component in SCCF framework.

- We also observe one phenomenon exhibiting in the experiment results. Our SCCF framework promotes SASRec most remarkable on ML-20M dataset than other datasets. SASRec needs to learn long sequence dependencies with enough training instances. Therefore, with less training instances (ML-1M) and fewer interactions per user (amazon datasets), it is difficult for SASRec to derive meaningful representations, thus difficult to identify accurate user neighborhood.

C. Complementary issues(RQ2)

To explore whether UI and user-based components recommend items in different ways, we investigate their recommendation results. We make this analysis on the experiment of $SASRec_{SCCF}$ on the ML-20M dataset. For ground truth, we compute the cosine similarity between the user embedding and the target item embedding for each user. When it comes to the UI and user-based components, we calculate the cosine scores between the user and each item in the candidate sets. Then for each user, we average these values to obtain the score of the whole candidate set. Finally, we summarize the number of users of each value.

The results are visualizing in Figure 4. We observe that the mean score of UI component similarity distribution is a litter higher than ground truth, while user-based component is lower than the target item distribution. This phenomenon reveals that there really exists significantly difference between UI methods and user-based component. UI method usually tends to provide items with higher similarity, since it is optimized by minimizing the distance between user representations and interacted item representations. In this way, they fail to recommend items which may be a little far from the specific user. In contrast, user-based component can capture more local information in user neighborhood and recommend items which are not very similar to the users. Therefore, UI and user-based components can complement each other in global and local manners, which enhances our SCCF to achieve better performance.

D. Efficiency Analysis(RQ3)

In this section, we conduct experiments to verify the efficiency of our SCCF user-based component in the real-time settings. Since user-based baseline models cannot be applied in large-scale applications, we simulate the real-time settings on public datasets. For the fact that factorized user-based methods are transductive models, they need to re-factorized the user/item matrix and cost more time than other methods. Therefore, we just compare with UserKNN and explore the compromise between inferring new representations and identifying neighbors. We report the time consumed by each method to make new predictions when a user interact with a new item. Results are the average of all users. Experiments are conducted on a V100 GPU with 16GB memory. To make fair comparison, we just utilize ML-1M and Video datasets since baseline methods require more GPUs with extra communication on other larger datasets. We apply SCCF on SASRec for comparison.

The results are displayed in Table III. We observe that SCCF user-based component is more efficient and scalable than UserKNN. Though our SCCF needs to spend extra time to infer user representations dynamically, it identifies user neighborhood more quickly and scalable with low dimensional vectors. And the latency of UserKNN is growing at least linearly with the number of items, thus restrcting it to be applied in real world applications with millions even billions of items.

Table II
PERFORMANCE COMPARISON OF DIFFERENT METHODS ON TOP-N RECOMMENDATION. BOLD SCORES ARE THE BEST IN METHOD GROUP.
IMPROVEMENTS ARE CALCULATED OVER UI METHODS.

Datasets	Metric	Pop	ItemKNN	UserKNN	BPR-MF	FISM	$FISM_{UU}$	$FISM_{SCCF}$	Improv.	SASRec	$SASRec_{UU}$	$SASRec_{SCCF}$	Improv.
ML-1M	HR@20	0.0596	0.1179	0.1308	0.1424	0.2189	0.2374	0.2677	22.29%	0.3447	0.2639	**0.3613**	4.82%
	HR@50	0.1356	0.2265	0.2444	0.2659	0.3740	0.3891	0.4041	8.05%	0.5227	0.4346	**0.5281**	1.03%
	HR@100	0.2190	0.3419	0.3593	0.4060	0.5017	0.5066	0.5315	5.94%	0.6409	0.5710	**0.6411**	0.03%
	NDCG@20	0.0224	0.0471	0.0513	0.0561	0.0891	0.0998	0.1177	32.10%	0.1521	0.1122	**0.1654**	8.74%
	NDCG@50	0.0372	0.0684	0.0737	0.0804	0.1200	0.1298	0.1447	20.58%	0.1874	0.1460	**0.1985**	5.92%
	NDCG@100	0.0507	0.0870	0.0923	0.1030	0.1408	0.1489	0.1653	17.40%	0.2067	0.1682	**0.2169**	4.93%
ML-20M	HR@20	0.0675	0.1279	-[1]	0.1099	0.2022	0.2220	0.2558	26.51%	0.2524	0.2327	**0.2939**	16.44%
	HR@50	0.1154	0.2302	-	0.2044	0.3325	0.3558	0.3887	16.90%	0.4075	0.3763	**0.4388**	7.68%
	HR@100	0.1832	0.3524	-	0.3061	0.4502	0.4641	0.4985	10.73%	0.5415	0.4967	**0.5569**	2.84%
	NDCG@20	0.0274	0.0520	-	0.0449	0.0852	0.0938	0.1183	38.85%	0.1065	0.0979	**0.1346**	26.38%
	NDCG@50	0.0370	0.0721	-	0.0635	0.1109	0.1203	0.1446	30.39%	0.1372	0.1263	**0.1634**	19.10%
	NDCG@100	0.0479	0.0919	-	0.0800	0.1300	0.1379	0.1624	24.92%	0.1589	0.1459	**0.1825**	14.85%
Games	HR@20	0.0333	0.0736	0.1227	0.1210	0.0974	0.1157	0.1413	45.07%	0.1325	0.1241	**0.1464**	10.49%
	HR@50	0.0591	0.1189	0.1852	0.2048	0.1684	0.1860	0.2215	31.53%	0.2300	0.2035	**0.2468**	7.30%
	HR@100	0.0919	0.1637	0.2338	0.2888	0.2445	0.2469	0.2879	17.75%	0.3260	0.2755	**0.3405**	4.45%
	NDCG@20	0.0155	0.0315	0.0546	0.0500	0.0399	0.0483	**0.0639**	60.15%	0.0536	0.0525	0.0599	11.75%
	NDCG@50	0.0206	0.0405	0.0670	0.0665	0.0540	0.0622	**0.0798**	47.78%	0.0728	0.0682	**0.0798**	9.62%
	NDCG@100	0.0259	0.0477	0.0749	0.0801	0.0663	0.0721	0.0906	36.65%	0.0883	0.0798	**0.0950**	7.59%
Beauty	HR@20	0.0134	0.0422	0.0676	0.0600	0.0425	0.0574	0.0628	47.76%	0.0648	0.0505	**0.0673**	3.86%
	HR@50	0.0248	0.0621	0.0942	0.1010	0.0705	0.0882	0.0983	39.43%	0.1045	0.0855	**0.1072**	2.58%
	HR@100	0.0398	0.0829	0.1160	0.1390	0.1006	0.1107	0.1309	30.12%	0.1460	0.1165	**0.1466**	0.42%
	NDCG@20	0.0050	0.0198	0.0320	0.0250	0.0173	0.0248	0.0278	52.63%	0.0289	0.0215	**0.0301**	4.15%
	NDCG@50	0.0072	0.0237	0.0373	0.0332	0.0228	0.0310	0.0348	52.63%	0.0367	0.0284	**0.0380**	3.54%
	NDCG@100	0.0097	0.0271	0.0408	0.0393	0.0277	0.0346	0.0400	44.40%	0.0434	0.0334	**0.0444**	2.30%

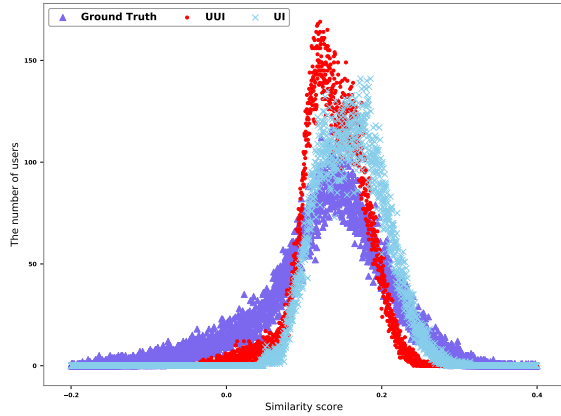[1] Running UserKNN on ML-20M will lead to timeout.

Figure 4. The number of users with similarity score with SASRec methods on ML-20M dataset. Each point is the similarity score between the user and the item. Note that the UI and user-based (UUI) scores are calculated by averaging the user-item similarity scores of all the candidate items.

E. Hyper-parameter Study(RQ4)

In this section, we analyze how the hyper-parameters impact the performance, including the hidden dimension and the size of user neighborhood N_u. We examine the influence of one hyper-parameter each time by fixing the remaining hyper-parameters. We report HR@50 and NDCG@50 as metrics

Table III
PERFORMANCE COMPARISON SCCF USER-BASED COMPONENTS AND
OTHER USER-BASED BASELINES IN REAL-TIME SETTINGS.

Datasets		Methods	
		UserKNN	SCCF
ML-1M	Inferring time (ms)	0	1.66
	Identifying time (ms)	6.83	0.72
	Total time (ms)	6.83	2.38
Videos	Inferring time (ms)	0	0.79
	Identifying time (ms)	51.95	0.75
	Total time (ms)	51.95	1.54

for the hidden dimension experiments and we just report NDCG@50 for the neighborhood experiments due to the space limitation.

First, we examine the impact of hidden dimension. Figure 5 demonstrates the performance of different embedding sizes while keeping other hyper-parameters unchanged. We observe a few trends with the variation of embedding sizes. First, the performance of each SCCF instance tends to converge as the dimension increases on most datasets. But a large dimension does not necessarily promise better performance due to the overfitting problem. On some datasets, such as Games, high dimension even sacrifices the performance. Second, the
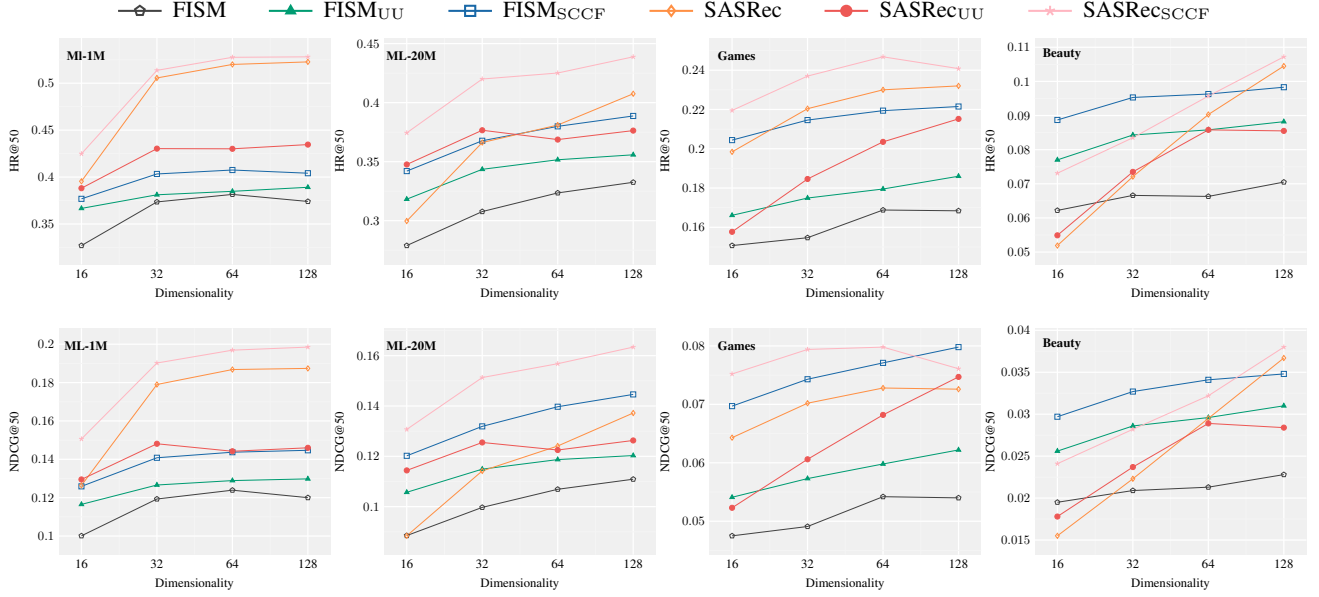
2472

Figure 5. Impact of the hidden dimensionality k on HR@50 and NDCG@50.

performance trend that our SCCF outperforms their base UI component is consistent with different embedding sizes. Even with a relatively small hidden size, our SCCF framework still outperforms its base component.

Then we study how the neighbor number β influences the performance. Table IV demonstrates the performance of different neighbor numbers. SCCF achieves stable improvements over different neighborhood sizes. We also observe that enlarging user neighborhood may decrease the model quality, since it may introduce noise users.

F. Online A/B Test Result(RQ5)

Previous offline experiment results have demonstrated the superiority of our SCCF framework. In this subsection, we deploy it on the mobile Taobao, which is the largest e-commerce platform in China. The experiments are conducted at What You May Like, the major recommendation scenario in Taobao.

It is worth mentioning the two-step process has been widely applied in the large scale recommendation scenarios [10, 27]. We also adopt this procedure on our platform. To make a fair comparison, we keep all downstream modules unchanged except the candidate generation module. We restrict the candidate set to 500 items and provide this set as input for the following stage. The baseline we deployed online is a deep model similar to the method proposed by Covington et al. [10]. We plugin our SCCF and provide exactly the same number of candidates. We randomly split some online users equally into two separate buckets. For users in the bucket A (i.e., the baseline group), we derive candidate generation step with the base approach. While for users in the bucket B (i.e., the experimental group), the candidates are suggested by our SCCF.

Table IV
PERFORMANCE COMPARISON OF DIFFEREN NEIGHBOR NUMBERS β ON NDCG@50. BOLD SCORES ARE THE BEST IN GROUPS.

Methods	Neighbors	Datasets			
		ML-1M	ML-20M	Games	Beauty
$FISM_{UI}$	$\beta = 50$	0.1200	0.1109	0.0540	0.0228
	$\beta = 100$	0.1200	0.1109	0.0540	0.0228
	$\beta = 200$	0.1200	0.1109	0.0540	0.0228
$FISM_{UU}$	$\beta = 50$	0.1289	0.1179	0.0587	0.0289
	$\beta = 100$	0.1298	0.1203	0.0622	0.0310
	$\beta = 200$	0.1287	0.1205	0.0631	0.0308
$FISM_{SCCF}$	$\beta = 50$	0.1433	0.1436	0.0778	0.0353
	$\beta = 100$	**0.1447**	**0.1446**	**0.0798**	0.0348
	$\beta = 200$	**0.1447**	0.1428	0.0788	**0.0349**
$SASRec_{UI}$	$\beta = 50$	0.1874	0.1372	0.0728	0.0367
	$\beta = 100$	0.1874	0.1372	0.0728	0.0367
	$\beta = 200$	0.1874	0.1372	0.0728	0.0367
$SASRec_{UU}$	$\beta = 50$	0.1454	0.1245	0.0645	0.0312
	$\beta = 100$	0.1460	0.1263	0.0682	0.0284
	$\beta = 200$	0.1444	0.1248	0.0691	0.0238
$SASRec_{SCCF}$	$\beta = 50$	0.1972	**0.1651**	0.0775	**0.0383**
	$\beta = 100$	**0.1985**	0.1634	**0.0798**	0.0380
	$\beta = 200$	0.1973	0.1643	0.0783	**0.0383**

The experiment is deployed online for one week from 8. Aug to 14. Aug in 2019. Each bucket contains about one million user's daily traffic. Different from offline metrics, the performance is judged by total user clicks and trades, which have strong correlations with the profit of corporations.

2473

Table V
ONLINE EXPERIMENT RESULTS

Metric	#Clicks	#Trades
Lift Rate	2.5%	2.3%

Table V shows that SCCF outperforms the baseline on both total clicks and total trade volume. The number of clicks is improved by 2.5% and trade volume is improved by 2.3% on average. The improvements are remarkable in the industrial online environment.

V. CONCLUSION

In this paper, we explore the effectiveness of the user-based methods and we propose an efficient framework SCCF for candidate generation. SCCF combines information from both global and local perspectives captured by the UI and the user-based components, respectively. Meanwhile, the user-based component of SCCF can be utilized in real time by dynamically inferring the user representations. In this way, SCCF is the first to address this real-time problem of user-based methods. Most importantly, it can be easily applied to existing inductive UI methods. The proposed framework is verified on both publicly datasets and an online industrial environment. Extensive experiment results show that the proposed framework achieves significant improvements and outperforms the start-of-the-art baselines.

In the future, we will investigate how to incorporate side information such as user profile to identify similar users for each user. Since user profile usually contains general user information, it can help us identify more accurate user preference. In addition, we will also attempt to apply our approach to the "ranking" step besides "candidate generation" step, since existing methods only consider user-item relation to predict the score for each candidate in the "ranking" step.

ACKNOWLEDGEMENT

REFERENCES

[1] R. Jiang, S. Gowal, Y. Qian, T. A. Mann, and D. J. Rezende, "Beyond greedy ranking: Slate optimization via list-cvae," in *Proceedings of ICLR*, 2019.

[2] C. Li, Z. Liu, M. Wu, Y. Xu, H. Zhao, P. Huang, G. Kang, Q. Chen, W. Li, and D. L. Lee, "Multi-interest network with dynamic routing for recommendation at tmall," in *Proceedings of CIKM*, 2019, pp. 2615–2623.

[3] F. Lv, T. Jin, C. Yu, F. Sun, Q. Lin, K. Yang, and W. Ng, "SDM: sequential deep matching model for online large-scale recommender system," in *Proceedings of CIKM*, 2019, pp. 2635–2643.

[4] R. Srivastava, G. K. Palshikar, S. Chaurasia, and A. Dixit, "What's next? a recommendation system for industrial training," *Data Science and Engineering*, vol. 3, no. 3, pp. 232–247, 2018.

[5] Z. Gharibshah, X. Zhu, A. Hainline, and M. Conway, "Deep learning for user interest and response prediction in online display advertising," *Data Science and Engineering*, vol. 5, no. 1, pp. 12–26, 2020.

[6] H. Chen, H. Jin, and X. Cui, "Hybrid followee recommendation in microblogging systems," *Science China Information Sciences*, vol. 60, no. 1, pp. 1–14, 2017.

[7] Y. Gong, Q. Zhang, X. Han, and X. Huang, "Phrase-based hashtag recommendation for microblog posts," *Science China Information Sciences*, vol. 60, no. 1, pp. 1–13, 2017.

[8] F. Fu, Y. Hu, Y. He, J. Jiang, Y. Shao, C. Zhang, and B. Cui, "Don't waste your bits! squeeze activations and gradients for deep neural networks via tinyscript," in *Proceedings of ICML*, 2020, pp. 3304–3314.

[9] X. Miao, L. Ma, Z. Yang, Y. Shao, B. Cui, L. Yu, and J. Jiang, "Cuwide: Towards efficient flow-based training for sparse wide models on gpus," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2020.

[10] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of RecSys*, 2016, pp. 191–198.

[11] P. Cremonesi, Y. Koren, and R. Turrin, "Performance of recommender algorithms on top-n recommendation tasks," in *Proceedings of RecSys*, 2010, pp. 39–46.

[12] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of WWW*, 2001, pp. 285–295.

[13] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, Jan. 2003.

[14] X. Ning and G. Karypis, "Slim: Sparse linear methods for top-n recommender systems," in *Proceedings of ICDM*, Washington, DC, USA, 2011, pp. 497–506.

[15] S. Kabbur, X. Ning, and G. Karypis, "Fism: Factored item similarity models for top-n recommender systems," in *Proceedings of KDD*. ACM, 2013, pp. 659–667.

[16] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proceedings of KDD*, 2008, pp. 426–434.

[17] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of SIGIR*, 1999, pp. 230–237.

[18] S. Sedhain, A. K. Menon, S. Sanner, and D. Braziunas, "On the effectiveness of linear models for one-class collaborative filtering," in *Proceedings of AAAI*, 2016, pp. 229–235.

[19] S. Sedhain, H. Bui, J. Kawale, N. Vlassis, B. Kveton, A. K. Menon, T. Bui, and S. Sanner, "Practical linear models for large-scale one-class collaborative filtering," in *Proceedings of AAAI*, 2016, pp. 3854–3860.

[20] E. Christakopoulou and G. Karypis, "Local latent space

models for top-n recommendation," in *Proceedings of KDD*, 2018, pp. 1235–1243.

[21] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system-a case study," in *WebKDD 2000 Workshop*, 2000.

[22] T. Ebesu, B. Shen, and Y. Fang, "Collaborative memory network for recommendation systems," in *Proceedings of SIGIR*, 2018, pp. 515–524.

[23] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *Proceedings of ICDM*, 2018, pp. 197–206.

[24] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, 2019.

[25] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, Jan. 2004.

[26] B. Smith and G. Linden, "Two decades of recommender systems at amazon.com," *IEEE Internet Computing*, vol. 21, no. 3, pp. 12–18, May 2017.

[27] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath, "The youtube video recommendation system," in *Proceedings of RecSys*, 2010, pp. 293–296.

[28] C. A. Gomez-Uribe and N. Hunt, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manage. Inf. Syst.*, vol. 6, no. 4, pp. 13:1–13:19, Dec. 2015.

[29] E. Christakopoulou and G. Karypis, "Local item-item models for top-n recommendation," in *Proceedings of RecSys*, 2016, pp. 67–74.

[30] O. Barkan and N. Koenigstein, "Item2vec: Neural item embedding for collaborative filtering," *Proceedings of MLSP*, pp. 1–6, 2016.

[31] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proceedings of NIPS*, 2013, pp. 3111–3119.

[32] T. Nedelec, E. Smirnova, and F. Vasile, "Specializing joint representations for the task of product recommendation," in *Proceedings of the 2Nd Workshop on Deep Learning for Recommender Systems*, 2017, pp. 10–18.

[33] R. Pan, Y. Zhou, B. Cao, N. N. Liu, R. Lukose, M. Scholz, and Q. Yang, "One-class collaborative filtering," in *Proceedings of ICDM*, 2008, pp. 502–511.

[34] R. Pan and M. Scholz, "Mind the gaps: Weighting the unknown in large-scale one-class collaborative filtering," in *Proceedings of KDD*, 2009, pp. 667–676.

[35] C. Elkan and K. Noto, "Learning classifiers from only positive and unlabeled data," in *Proceedings of KDD*, 2008, pp. 213–220.

[36] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proceedings of NIPS*, 2007, pp. 1257–1264.

[37] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *KDD Cup*, 2007.

[38] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of WWW*, 2017, pp. 173–182.

[39] X. He, Z. He, J. Song, Z. Liu, Y.-G. Jiang, and T.-S. Chua, "Nais: Neural attentive item similarity model for recommendation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 12, pp. 2354–2366, 2018.

[40] H.-J. Xue, X.-Y. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *Proceedings of IJCAI*, 2017, pp. 3203–3209.

[41] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua, "Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention," in *Proceedings of SIGIR*, 2017, pp. 335–344.

[42] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proceedings of ICLR*, 2015.

[43] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *CoRR*, vol. abs/1511.06939, 2015.

[44] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of CIKM*, 2019, p. 1441–1450.

[45] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *Proceedings of WSDM*, 2018, pp. 565–573.

[46] Z. Zhu, J. Wang, and J. Caverlee, "Improving top-k recommendation via joint collaborative autoencoders," in *Proceedings of WWW*, 2019, pp. 3483–3482.

[47] J. Weston, S. Chopra, and A. Bordes, "Memory networks," in *Proceedings of ICLR*, 2015.

[48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of NIPS*, 2017, pp. 5998–6008.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, 2015.

[50] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 19:1–19:19, Dec. 2015.

[51] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of SIGIR*. ACM, 2015, pp. 43–52.

[52] R. He, W.-C. Kang, and J. McAuley, "Translation-based recommendation," in *Proceedings of RecSys*, 2017, pp. 161–169.

[53] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender systems handbook*. Springer, 2011, pp. 1–35.

[54] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *Proceedings of UAI*, 2009, pp. 452–461.