Learn over Past, Evolve for Future: Search-based Time-aware Recommendation with Sequential Behavior Data

Jiarui Jin[1,*], Xianyu Chen[1,*], Weinan Zhang[1,†], Junjie Huang[1], Ziming Feng[2], Yong Yu[1].
[1]Shanghai Jiao Tong University, China; [2]China Merchants Bank Credit Card Center, China
{jinjiarui97,xianyujun,legend0018,wnzhang,yyu}@sjtu.edu.cn,zimingfzm@cmbchina.com

ABSTRACT

The personalized recommendation is an essential part of modern e-commerce, where user's demands are not only conditioned by their profile but also by their recent browsing behaviors as well as periodical purchases made some time ago. In this paper, we propose a novel framework named **S**earch-based **T**ime-**A**ware **Rec**ommendation (**STARec**), which captures the evolving demands of users over time through a unified search-based time-aware model. More concretely, we first design a search-based module to retrieve a user's relevant historical behaviors, which are then mixed up with her recent records to be fed into a time-aware sequential network for capturing her time-sensitive demands. Besides retrieving relevant information from her personal history, we also propose to search and retrieve similar user's records as an additional reference. All these sequential records are further fused to make the final recommendation. Beyond this framework, we also develop a novel label trick that uses the previous labels (i.e., user's feedbacks) as the input to better capture the user's browsing pattern. We conduct extensive experiments on three real-world commercial datasets on click-through-rate prediction tasks against state-of-the-art methods. Experimental results demonstrate the superiority and efficiency of our proposed framework and techniques. Furthermore, results of online experiments on a daily item recommendation platform of Company X show that STARec gains average performance improvement of around 6% and 1.5% in its two main item recommendation scenarios on CTR metric respectively.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; **Collaborative search**.

KEYWORDS

Search-based Model, Time-aware Sequential Network, Label Trick

ACM Reference Format:
Jiarui Jin, Xianyu Chen, Weinan Zhang, Junjie Huang, Ziming Feng, Yong Yu. 2022. Learn over Past, Evolve for Future: Search-based Time-aware Recommendation with Sequential Behavior Data. In *Proceedings of the ACM Web Conference 2022 (WWW '22), April 25–29, 2022, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3485447.3512117

[*]Jiarui Jin and Xianyu Chen contributed equally to the work.
[†]Weinan Zhang is the corresponding author.

1 INTRODUCTION

Due to the rapid growth of user historical behaviors, it becomes an essential problem to build an effective recommendation model to help users to find their desired items from a huge number of candidates. Classical recommendation methods, including collaborative filtering based models [1, 11, 19] and factorization machine based models [5, 18, 36], have mainly focused on modeling user's general interests to find her favorite products; while less exploring the user's demands with an aspect of time. As stated in [2, 44, 48], time is definitely an important factor that can significantly influence user's demands and result in periodical user behaviors. Therefore, a branch of recent attempts are proposed to capture user's sequential patterns through either memory networks [35], recurrent neural networks [44, 48], or temporal point processes [2, 9]. However, most existing approaches can only be applied for user behavior data with length scaling up to hundreds due to the computation and storage limitations in real online system [31, 32, 46, 47].

To tackle this issue, we consider combining it with recently proposed search-based models [31, 32], whose key idea is to first search the effective information from the user's historical records to capture specific interests of the user in terms of different candidate items, which are then used to make the final prediction. However, it's non-trivial to do that due to the following challenges:

- **(C1)** How to incorporate the user's sequential patterns into these search-based models? Existing search-based methods [31, 32] overlook user's sequential patterns (i.e., the effect of time factor). As a consequence, when a teen has purchased a lipstick, these methods are likely to recommend the same or similar products before she gets tired of or runs out of the purchased one. Hence, it's essential to take the time information into account, as the user's demands are highly time-sensitive.
- **(C2)** How to leverage the label information (i.e., user feedbacks) from historical data in the recommendation model? The principal way to use the user historical feedbacks is to treat this feedbacks as the label to supervise the model. However, as discussed in [38, 42], combining the information from both label and feature as the input to train the model can significantly improve its performance. As directly mixing up all this information will definitely lead to the label leakage issue, then how to smartly enrich the model with the label information needs to investigate.
- **(C3)** How to design a learning algorithm to simultaneously train a search-based model and a prediction model in an end-to-end

fashion? Previous attempts either manually design a mixed loss function [31] or apply a reinforcement learning (RL) [32] in training. As the performance of the former one largely relies on the loss design and hyper-parameter tuning, and the latter one usually suffers from the sample inefficiency of the RL algorithm, the training algorithm design also is another significant challenge.

In this paper, we propose a novel sequential recommendation framework named **S**earch-based **T**ime-**A**ware **Rec**ommendation (**STARec**) which captures user's evolving demands over time through a unified search-based time-aware model.

Concretely, noticing that *category* plays an essential role in search models [31], we first construct an embedding vector for each category. We then search and retrieve items either by a hard-search strategy based on category IDs or a soft-search strategy based on the similarities between their category embeddings. The intuition of using category for search and retrieval is straightforward. Taking Figure 1(a) as an instance, the motivation of the teen u_1 buying the lipstick i_1 can either lie in that she is running out of her old lipstick i_2, or that she needs an accessory for her new purchases (e.g., lip gloss i_4), but not likely due to her purchased iPhone i_3. Note that our search-based module using category embeddings instead of item embeddings would make the whole framework much easier to train. We also design a novel adaptive search mechanism that can gradually transfer from the hard-search strategy to the soft-search one when the embedding vectors are well-tuned.

In order to mine the hidden time-aware patterns, we then mix up the retrieved items together with recent browsed ones and feed them into a time-aware sequential network that considers not only the sequential orders but also their time intervals. Besides the user's own histories, we also attempt to enrich the model by similar users' historical records. As shown in Figure 1(b), when recommending i_1 to u_1, we argue that referring to similar historical records from similar users such as u_2 would be helpful; while the histories of dissimilar users such as u_3 would be noise. This user similarity can be either softly estimated through the inner-product of their embedding vectors or can be hardly measured by counting the number of purchased items with the same category with i_1.

Different from current prevailing methods using user's feedbacks (e.g., click, purchase) only as the supervision signals. As Figure 1(c) shows, we propose to involve the user's previous feedbacks as input, where the label of the target item is set as a randomized value. We call this technique *label trick*. Its intuition is straightforward that if a user finds her desired items, it's unlikely for her to click or purchase other similar items.

In summary, the contributions of the paper are three-fold:

- We propose a novel framework named STARec, which captures the user's time-sensitive evolving demands via combining a search-based module and a time-aware module.
- We propose to involve the user's previous feedbacks as input and reveal that this label information can improve the performance.
- We design a new adaptive search mechanism, which gradually transfers from the hard-search strategy to the soft one.

We conduct extensive experiments on three industrial datasets, and experimental results exhibit that the superiority of STARec over the state-of-art methods. We successfully deploy STARec in two main item recommendation scenarios in Company X, and share

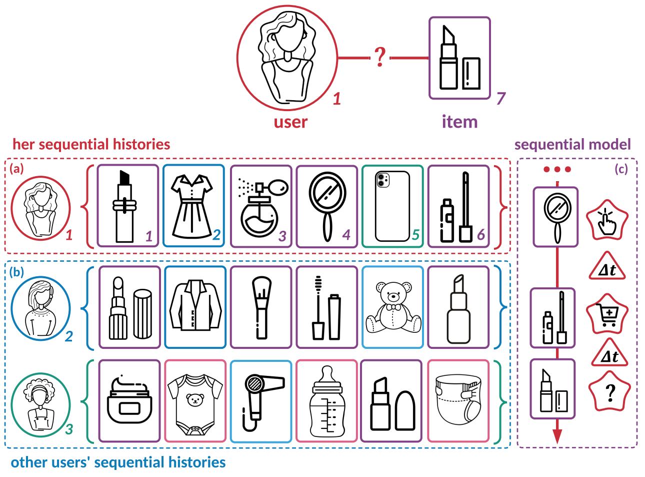

Figure 1: An illustrated example for motivations of STARec: For search-based module, (a) among historical records of a user u_1, we search items (e.g., i_1) with the same or similar category to the target item i_7; (b) to further enrich the information, we involve similar users' (e.g., u_2's) related histories as reference. For the time-aware module, (c) we develop a sequential network, and design a label trick to involve the user's previous feedbacks as input. In this case, the label of the target item (denoted as ?) is set as a randomized value.

our hands-on experience and discuss the potential extensions to ranking tasks in the appendix.

2 PRELIMINARIES

2.1 Related Work

Search-based Recommendation Model. Classical recommendation methods are proposed to recommend desired items to users based on rich user-item interaction histories either in tabular format [10, 18, 20, 36], sequence structure [2, 46, 47], or graph structure [16, 41]. However, as stated in [32], since the users are accumulating more and more behavioral data nowadays, it's non-trivial to train the model from the whole user logs due to the limitations from the online computations. One feasible solution is only to focus on user's recent records and generate personalized recommendations based on a short sequence instead of a long history [4, 8, 13, 29, 34, 37, 39]. However, as recently proposed works [31, 32] suggest, these methods are not able to encode the periodicity or long-term dependency, which leads to sub-optimal solutions. Based on this observation, Pi et al. [31], Qin et al. [32] further propose to build a search model either following a hard-search or soft-search strategy over the whole behavioral history. In this way, they can use those relevant items instead of the whole set of user-browsed items to efficiently learn a recommendation method. Unfortunately, these existing search-based methods overlook effects from time intervals among user's behaviors and thus can not fully use user's browsing sequence.

Time-aware (Sequential) Model. Recent studies [15, 48] have paid attention on leveraging the time intervals among user's behaviors to better capture user's preferences, for which traditional

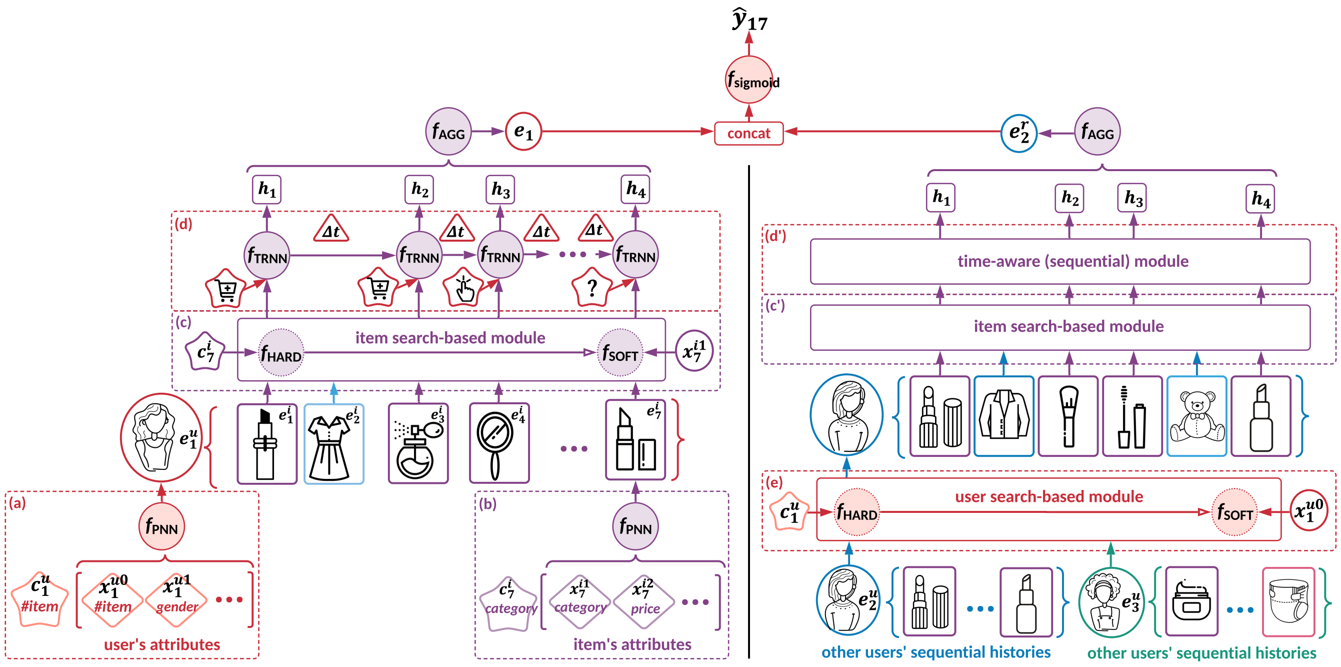

Figure 2: The overview of STARec. In (a)(b), we use PNN to encode the categorical attributes for both users and items, if available. Notably, x_i^{u0} is a manually created feature that denotes the embedding vector of c_1^u, and c_1^u is the number of items in user u_1's browsed histories sharing the same category with the target item i_7. In (c)(d), for each user-item pair, we construct an adaptive search-based module to select relevant items from the whole browsing logs and then feed them into a time-aware (sequential) module. Moreover, in (e), we regard the browsing histories from similar users as the additional reference to assist the final prediction (i.e., $\widehat{y}_{17}$ for user u_1 and item i_7) making. We illustrate the proposed label trick in (d), where previous user feedbacks are used as the input to recover the label of the current item.

sequential architectures [3, 12, 13, 17, 22] are insufficient. One direction [24, 45, 48] is to develop the specific time gates to control the short-term and long-term interest updates. For example, Zhao et al. [45] introduces a distance gate based on a recurrent network to control the short-term and long-term point-of-interest updates. Another way [2, 6, 15, 27, 40] to integrate time interval information is to formulate the user's sequential histories by a point process, in which the discrete events in user's histories can be modeled in continuous time. For example, Mei and Eisner [27] proposes a neural hawkes process model which allows past events to influence the future prediction in a complex and realistic fashion. We argue that despite computation cost and time complexity, directly feeding long sequential user behavioral data into these methods will bring much more noise which makes it nearly impractical to capture the rich sequential patterns in the user logs.

In this paper, we combine the advantages from both search-based and time-aware models to efficiently retrieve relevant items and mine sequential patterns in an end-to-end way. Our paper is also related to the label trick proposed in [38, 42] based graph structure. Instead, our work focuses on the label usage in the sequence cases, which, notably, is also different from the masking technique in existing sequential models such as BERT [7] performing on the feature dimension instead of the label dimension.

2.2 Problem Formulation

We begin by clarifying the recommendation task we study and introducing associated notations we use.

Definition 2.1. **Search-based Time-aware Recommendation**[1]. Given a tuple $\langle \mathcal{U}, \mathcal{I}, \mathcal{C} \rangle$ where $\mathcal{U}$ is the set of users, $\mathcal{I}$ is the set of items, $\mathcal{C}$ is the set of items' categories. For each user $u_m \in \mathcal{U}$, her historical records can be formulated as a sequence of items sorted by time $\mathcal{H}_m = \{i_1, i_2, \ldots, i_T\}$ where $i_t \in \mathcal{H}_m$ is the item browsed by user u at time t. For each item $i_n \in \mathcal{I}$, let c_n^i denote its category (ID). We use x_m^u, x_n^i to denote the feature of the m-th user, the n-th item respectively, and further use x_m^{up}, x_n^{ip} to denote their p-th categorical features. The goal of the recommendation is to infer the probability of the user u_m clicking or purchasing the item i_n at a future time $T+1$ conditioning on the retrieved user historical records, denoted as $\widehat{\mathcal{H}}_m = \{\widehat{i}_1, \widehat{i}_2 \ldots, \widehat{i}_{\widehat{T}}\}$ where $\widehat{T}$ is the length of retrieval.

For convenience, in the following sections, we use the 1-th categorical feature of each item i_n to represent its category (ID) (e.g., cosmetics). Namely, x_n^{i1} denotes the feature for n-th item's category. For each user u_m, we also manually calculate the number of items sharing the same category with target item and further use c_m^u to

[1]Notably, a more precise way for notations is to use them in the continuous setting, e.g., define $\mathcal{H}_u$ as $\mathcal{H}_u = \{i_{t_1}, i_{t_2}, \ldots, i_{t_T}\}$ where $t_1, t_2, \ldots, t_T$ denote continuous timestamps, and replace $T+1$ by $T+\epsilon$.

denote the number. Note that this number is determined by each user-item pair, not solely depends on user. Moreover, for each user u_m and item i_n we introduce $\widehat{\mathcal{S}}_m = \{\widehat{u}_1, \widehat{u}_2, \ldots, \widehat{u}_{\widehat{M}}\}$ to denote a set of users similar to u_m being aware of i_n. The computations of $\widehat{\mathcal{H}}_m$ and $\widehat{\mathcal{S}}_m$ are introduced in the following Eqs. (2) and (4).

As discussed in [2], regardless of search-based part, this time-aware recommendation task (called continuous-time recommendation in [2]) can be regarded as a generalized sequential recommendation problem of the next-item and next-session/basket problems. Notably, different from existing solutions to this problem, our method, with the help of the search-based module, can particularly model the time intervals among the relevant items to answer "*How often does she purchase cosmetics?*" instead of "*How often does she purchase items?*" Easy to see, the answers to the former question are much more informative than the latter one.

3 THE STAREC MODEL

3.1 Intuition and Overview

The basic idea of STARec is to combine the advantages from search-based and time-aware modules, which is based on the following intuitions (as illustrated in Figure 1):

- **(I1)** When predicting a user's (e.g., u_1) interest in an item (e.g., i_1), we need to answer "*whether u_1 wants to buy a lipstick*" and "*whether i_1 (or its price) is suitable for u_1*", both of which motivate us to search and retrieve the relevant items from her records instead of using the whole browsing items.

Specifically, for the first one, although there must be numerous personalized reasons for buying a lipstick, we argue that the popular ones either lie in running out of her old one i_2 or wanting an accessory for her new purchases (e.g., lip gloss i_4), but not likely due to her purchased iPhone i_3. Also, as for the second one, the prices of these relevant items in u_1's browsing history (e.g., her previous lipstick i_2) can give us a useful hint for the suitable price of lipstick in her mind while those irrelevant ones (e.g., her purchased iPhone i_3) are much less informative.

- **(I2)** User's interests are naturally diverse and always drifting, which can be captured from their behaviors. However, each interest has its own evolving process. For example, a teen may purchase lipsticks weekly, and phones yearly, and purchasing lipsticks only has a slight effect on purchasing phones. It supports us to build a time-aware module for each class of items.
- **(I3)** User's current behavior can be significantly influenced by her previous ones. For example, a user is likely to stop browsing after clicking or purchasing an item since she has already found her favorite. It motivates us to include user feedbacks (i.e., labels) as the input.

Figure 2 shows the overview of STARec. First, we use a product-based neural network (PNN) [33] to model the correlations of categorical features (if available) for each user and item, as shown in (a)(b). After that, we develop a novel adaptive search-based module to retrieve relevant items based on the similarities between their categories and the target item's category, and then use a time-aware module to mine their sequential patterns, as (c)(d) illustrate. Moreover, we also retrieve those similar users' histories and regard this information as the additional references to assist the final prediction making, as (e) shows. Note that besides this architecture, we propose to involve the user's previous feedbacks (i.e., labels) in the input, as illustrated in (d).

3.2 Object Modeling

If we are not accessible to the rich categorical features for each user and item, we can build an embedding vector (i.e., $\boldsymbol{e}^u_m$ and $\boldsymbol{e}^i_n$) for each user (ID) u_m and item (ID) i_n. Otherwise, we need to consider rich correlations among these features, which play an important role in user interest modeling. For instance, in Figure 2, u_1 is likely to purchase i_1 because she wants to buy a lipstick **AND** its price is suitable for her. As discussed in [33, 36], this "AND" operation can not be solely modeled by classical neural network (e.g., multi-layer perceptron (MLP)) but can be captured by the product-based neural network (PNN) model [33]. Therefore, we adopt PNN to capture the hidden correlations among the categorical features for each user and item. Specifically, its output of n-th item can be formulated as

$$\boldsymbol{e}^i_n := f^i_{\mathsf{PNN}}(\boldsymbol{x}^i_n) = \boldsymbol{v}^i_n \odot \boldsymbol{x}^i_n + \sum_{p=1}^{P}\sum_{p'=p+1}^{P}(\boldsymbol{v}^i_p \odot \boldsymbol{v}^i_{p'})\boldsymbol{x}^{ip}_n \boldsymbol{x}^{ip'}_n, \quad (1)$$

where $\odot$ denotes the element-wise product operation and $\boldsymbol{v}^i_n, \boldsymbol{v}^i_p$ denote learnable weights. In Eq. (1), the first term shows the first-order feature interactions, and the second term illustrates the second-order feature interactions. As for each user, similarly, we can define $\boldsymbol{e}^u_m := f^u_{\mathsf{PNN}}(\boldsymbol{x}^u_m)$ as the output of the m-th user where $f^u_{\mathsf{PNN}}(\cdot)$ and $f^i_{\mathsf{PNN}}(\cdot)$ share the same formulation but with different parameters.

3.3 Search-based Module

Item Search-based Module. As discussed in [31], the categories are one of the most powerful tools to measure the similarity (i.e., relevance) between the user's browsed items and target item. Based on this, we can easily derive a hard-search strategy. Formally, we first construct a set of items for each user u_m defined as

$$\widehat{\mathcal{H}}_m := \{i_{n'} | i_{n'} \in \mathcal{H}_m \wedge f^i_{\mathsf{HARD}}(c^i_{n'}, c^i_n) \geq \epsilon\} \cup \mathcal{H}^{\mathsf{RECENT}}_m, \quad (2)$$

where except $\mathcal{H}^{\mathsf{RECENT}}_m$ denotes a set of recent browsed items, the first term (i.e., $i_{n'} \in \mathcal{H}_m$) limits the retrieved items to come from u_m's browsing history and the second term (i.e., $f^i_{\mathsf{HARD}}(c^i_{n'}, c^i_n) \geq \epsilon$) selects the relevant ones, $\epsilon \in [0, 1]$ denotes the threshold, $c_{n'}, c_n$ are one-hot vectors directly represent their categories without any learnable parameter. In this case, $f^i_{\mathsf{HARD}}(\cdot, \cdot)$ can be defined as $f^i_{\mathsf{HARD}}(c^i_{n'}, c^i_n) := -|c^i_{n'} - c^i_n|$ and $\epsilon = 0$.

The computation cost for this hard-search strategy is $|\mathcal{H}_m|$ for each user u_m and item i_n. It is very efficient, but sometimes too hard-and-fast. In other words, it can only find those items exactly sharing the same category with the target item. It doesn't work in the following case where *a teen purchases a beautiful dress and she needs lipstick to make up.* To handle these cases, we further introduce $f^i_{\mathsf{SOFT}}(\cdot, \cdot)$ defined as $f^i_{\mathsf{SOFT}}(\boldsymbol{x}^{i1}_{n'}, \boldsymbol{x}^{i1}_n) := \cos(\boldsymbol{x}^{i1}_{n'}, \boldsymbol{x}^{i1}_n) = ({\boldsymbol{x}^{i1}_{n'}}^\top \cdot \boldsymbol{x}^{i1}_n)/(|\boldsymbol{x}^{i1}_{n'}| \cdot |\boldsymbol{x}^{i1}_n|)$ where $\cos(\cdot, \cdot)$ denotes cosine similarity.

One can obtain retrieved items by the soft-search strategy through replacing $f^i_{\mathsf{HARD}}(c^i_{n'}, c^i_n)$ by $f^i_{\mathsf{SOFT}}(\boldsymbol{x}^{i1}_{n'}, \boldsymbol{x}^{i1}_n)$ and assigning $0 < \epsilon < 1$ in Eq. (2). In this case, the performance of the soft-search largely depends on how well learnable vectors $\boldsymbol{x}_{n'}^{i1}, \boldsymbol{x}_{n}^{i1}$ are trained. Existing methods either introduce a mixed loss function [31] or apply a reinforcement learning algorithm [32] in training. Instead, we propose a simpler and more effective way: an adaptive search strategy, which combines the advantages from both the hard-search and soft-search strategies and also enables the whole architecture to be trained in an end-to-end fashion.

We first employ a sign function denoted as $\mathtt{sgn}(\cdot)$ to re-scale the hard-search and use a softmax function denoted as $f_{\mathtt{softmax}}(\cdot,\cdot)$ to re-scale the soft-search. Formally, we define our adaptive search strategy $f^i_{\text{ADA}}(c^i_{n'}, c^i_n, \boldsymbol{x}^{i1}_{n'}, \boldsymbol{x}^{i1}_n)$ as

$$\begin{aligned} f^i_{\text{ADA}}(c^i_{n'}, c^i_n, \boldsymbol{x}^{i1}_{n'}, \boldsymbol{x}^{i1}_n) &:= -\frac{\mathtt{sgn}(|c^i_{n'} - c^i_n|)}{1-\tau} + f_{\mathtt{softmax}}(\cos(\boldsymbol{x}^{i1}_{n'}, \boldsymbol{x}^{i1}_n), \tau) \\ &= -\frac{\mathtt{sgn}(|c^i_{n'} - c^i_n|)}{1-\tau} + \frac{\exp(\cos(\boldsymbol{x}^{i1}_{n'}, \boldsymbol{x}^{i1}_n)/\tau)}{\sum_{i_{n''}\in\mathcal{H}_m}\exp(\cos(\boldsymbol{x}^{i1}_{n''}, \boldsymbol{x}^{i1}_n)/\tau)}, \end{aligned} \tag{3}$$

where $\tau \in (0,1)$ denotes temperature hyper-parameter to balance the hard-search and the soft-search. In practice, we set the initial temperature as 0.99, then gradually reduce the temperature until 0.01 during the training process. One can see that at the beginning, the first term (i.e., hard-search) plays a major part in the search, and as the training goes on, the second term (i.e., soft-search) is playing a more and more important role in the search. Therefore, with the help of this adaptive search, our whole architecture is able to be trained in an end-to-end fashion.

Besides using those relevant items from the item aspect, we also consider including similar experiences from relevant users as the reference. The motivation behind this is very straightforward that, as shown in Figure 1(b), the teens u_1, u_2 often share similar interests over items and similar browsing patterns, which are usually different from a young mother u_3. Hence, the browsing records of u_1, u_2 would benefit each other, and that of u_3 would be noise when modeling the browsing patterns of u_1.

Based on this observation, besides the item search-based module introduced above, we further construct a user search-based module, whose target is to find similar users for further including their records as the references to help with the final prediction making. Formally, for each user u_m and item i_n, we construct a set of retrieved users similar with u_m being aware of i_n as

$$\widehat{\mathcal{S}}_m := \{u_{m'} | u_{m'} \in \mathcal{U} \wedge f^u_{\text{HARD}}(c^u_m, c^u_{m'}) \geq \eta\}, \tag{4}$$

where analogous to Eq. (2), $c_{m'}, c_m$ are one-hot vectors directly representing the numbers of items in their histories sharing the same category with i_n without any learnable parameters, and $f^u_{\text{HARD}}(\cdot,\cdot)$ can be defined as $f^u_{\text{HARD}}(c^u_{m'}, c^u_m) := -|c^u_{m'} - c^u_m|$ and η is a threshold. Similarly, we define $f^u_{\text{SOFT}}(\cdot,\cdot)$ as $f^u_{\text{SOFT}}(\boldsymbol{e}^u_{m'}, \boldsymbol{e}^u_m) := \cos(\boldsymbol{e}^u_{m'}, \boldsymbol{e}^u_m)$, and propose an adaptive search strategy $f^u_{\text{ADA}}(c^u_{m'}, c^u_m, \boldsymbol{e}^u_{m'}, \boldsymbol{e}^u_m)$ from user aspect as

$$\begin{aligned} f^u_{\text{ADA}}(c^u_{m'}, c^u_m, \boldsymbol{e}^u_{m'}, \boldsymbol{e}^u_m) &:= -\frac{\mathtt{sgn}(|c^u_{m'} - c^u_m|)}{1-\iota} + f_{\mathtt{softmax}}(\cos(\boldsymbol{e}^u_{m'}, \boldsymbol{e}^u_m), \iota) \\ &= -\frac{\mathtt{sgn}(|c^u_{m'} - c^u_m|)}{1-\iota} + \frac{\exp(\cos(\boldsymbol{e}^u_{m'}, \boldsymbol{e}^u_m)/\iota)}{\sum_{u_{m''}\in\mathcal{U}}\exp(\cos(\boldsymbol{e}^u_{m''}, \boldsymbol{e}^u_m)/\iota)}, \end{aligned} \tag{5}$$

where $\iota \in (0,1)$ denotes temperature hyper-parameter to balance the hard and soft parts. After we obtain the set of similar users $\widehat{\mathcal{S}}_m$ for each user u_m, we then employ the item search-based module to construct a set of user browsing histories $\{\widehat{\mathcal{H}}_{m'} | u_{m'} \in \widehat{\mathcal{S}}_m\}$.

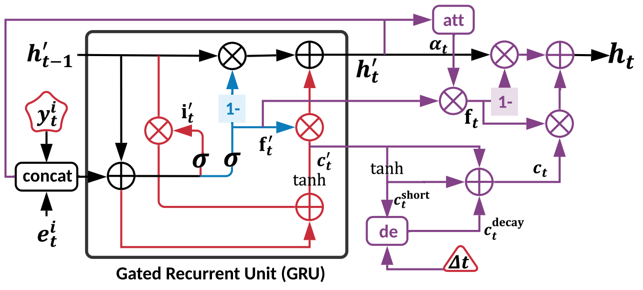

Figure 3: An illustrated example for time-aware module denoted as f_{TRNN} in Figure 2, where the input of the t-th cell is the concatenation of item's original feature x^i_t and embedding vector of label y^i_t. We incorporate modified GRU with an attention mechanism to model the user's sequential pattern, considering the effect of her previous feedbacks and time intervals in browsing history.

3.4 Time-aware Module

Given a user u_m and an item i_n, we have a set of retrieved items in u_m's browsing history $\widehat{\mathcal{H}}_m$ and a set of similar users' browsing histories $\{\widehat{\mathcal{H}}_{m'} | u_{m'} \in \widehat{\mathcal{S}}_m\}$.

For each $i_n \in \widehat{\mathcal{H}}_m$, we use y_n to denote the feedback from user u_m. It's straightforward to correspondingly build a one-hot vector or an embedding vector $\boldsymbol{y}^i_n$ here. Hence, as illustrated in Figure 3, the input of the t-th cell of our time-aware module (denoted as f_{TRNN}) is the concatenation of item's original feature and embedding vector of label (denoted as $[\boldsymbol{e}^i_t, \boldsymbol{y}^i_t]$).

In order to empower our time-aware module for modeling the user shifting interest over time, as shown in Figure 3, we first adopt a gated recurrent unit (GRU) to mine the useful sequential patterns, which can be formulated as

$$\begin{aligned} &\mathrm{f}'_t = \sigma(\boldsymbol{W}_{\mathrm{f}'}[\boldsymbol{e}^i_t, \boldsymbol{y}^i_t] + \boldsymbol{U}_{\mathrm{f}'}\boldsymbol{h}'_{t-1}),\ \mathrm{i}'_t = \sigma(\boldsymbol{W}_{\mathrm{i}'}[\boldsymbol{e}^i_t, \boldsymbol{y}^i_t] + \boldsymbol{U}_{\mathrm{i}'}\boldsymbol{h}'_{t-1}), \\ &\boldsymbol{c}'_t = \mathtt{tanh}(\boldsymbol{W}_{c'}[\boldsymbol{e}^i_t, \boldsymbol{y}^i_t] + \mathrm{i}'_t \odot \boldsymbol{U}_{c'}\boldsymbol{h}'_{t-1}),\ \boldsymbol{h}'_t = \mathrm{f}'_t \odot \boldsymbol{c}'_t + (1-\mathrm{f}'_t) \odot \boldsymbol{h}'_{t-1}, \end{aligned} \tag{6}$$

where we omit the bias term for simplicity. We further use attention mechanism to model the evolution of user interest and consider the effect from time intervals as

$$\begin{aligned} &\boldsymbol{c}^{\mathtt{short}}_t = \mathtt{tanh}(\boldsymbol{W}_c\boldsymbol{c}'_t + \boldsymbol{b}_c),\ \boldsymbol{c}^{\mathtt{decay}}_t = \boldsymbol{c}^{\mathtt{short}}_t \cdot \mathtt{de}(\Delta t), \\ &\alpha'_t = \exp(\boldsymbol{h}'_t\boldsymbol{W}_{\alpha'}[\boldsymbol{e}^i_t, \boldsymbol{y}^i_t]) / \sum_{t'=1}^{\widehat{T}+1}(\boldsymbol{h}'_t\boldsymbol{W}_{\alpha'}[\boldsymbol{e}^i_{t'}, \boldsymbol{y}^i_{t'}]),\ \mathrm{f}_t = \alpha'_t \cdot \mathrm{f}'_t, \\ &\boldsymbol{c}_t = \boldsymbol{c}'_t - \boldsymbol{c}^{\mathtt{short}}_t + \boldsymbol{c}^{\mathtt{decay}}_t,\ \boldsymbol{h}_t = \mathrm{f}_t \odot \boldsymbol{c}_t + (1-\mathrm{f}_t) \odot \boldsymbol{h}'_t, \end{aligned} \tag{7}$$

where Δt is the elapsed time between items i_{t-1} and i_t, and $\mathtt{de}(\cdot)$ denotes a heuristic decaying function. We use $\mathtt{de}(\Delta t) = 1/\Delta t$ for datasets with small amount of elapsed time and $\mathtt{de}(\Delta t) = 1/\log(e + \Delta t)$ for those with large elapsed time in practice.

As a consequence, for each sequence $\widehat{\mathcal{H}}_m$ or one in $\{\widehat{\mathcal{H}}_{m'} | u_{m'} \in \widehat{\mathcal{S}}_m\}$, we obtain a set of hidden states $\{\boldsymbol{h}_t | i_t \in \widehat{\mathcal{H}}_m\}$ or $\{\boldsymbol{h}_t | i_t \in \widehat{\mathcal{H}}_{m'}\}$. For each set of hidden states, we employ an aggregation function $f_{\text{AGG}}(\cdot)$ to fuse these embeddings into the representation of the whole sequence, which, taking $\{\boldsymbol{h}_t | i_t \in \widehat{\mathcal{H}}_m\}$ as an instance, can be formulated as

$$\boldsymbol{e}_m = f_{\mathsf{AGG}}(\{\boldsymbol{h}_t | i_t \in \widehat{\mathcal{H}}_m\}) = \sigma(\boldsymbol{W}_m \cdot (\sum_{i_t \in \widehat{\mathcal{H}}_m} \alpha_t (\boldsymbol{h}_t \boldsymbol{W}_t)) + \boldsymbol{b}_m), \quad (8)$$

where $\alpha_t = \exp(\boldsymbol{h}_t \boldsymbol{W}_\alpha) / \sum_{t'=1}^{\widehat{T}+1} (\boldsymbol{h}_t \boldsymbol{W}_\alpha)$. Similarly, we can obtain $\{\boldsymbol{e}^r_{m'} | u_{m'} \in \widehat{\mathcal{S}}_m\}$ where $\boldsymbol{e}^r_{m'} = f_{\mathsf{AGG}}(\{\boldsymbol{h}_t | i_t \in \mathcal{H}_{m'}\})$ for sequence $\widehat{\mathcal{H}}_{m'}$ in $\{\widehat{\mathcal{H}}_{m'} | u_{m'} \in \widehat{\mathcal{S}}_m\}$.

Notably, as introduced in Eq. (2), $\widehat{\mathcal{H}}_m$ for each user u_m consists of two parts: one is a set of the recent browsed items (i.e., $\widehat{\mathcal{H}}^{\mathsf{RECENT}}_m$), the other is a set of retrieved items (i.e., $\widehat{\mathcal{H}}_m / \widehat{\mathcal{H}}^{\mathsf{RECENT}}_m$). In the implementation, we establish two sequential networks (without parameter sharing). We use one sequential network for each part to encode these items and then combine these outputs together by concatenation. We demonstrate that this way is more efficient than putting all the items in one sequential network in Section 4.3.

3.5 Optimization Objective

For each user-item pair (u_m, i_n), we generate the final prediction $\widehat{y}_{mn}$ by encoding $\boldsymbol{e}_m$ and $\{\boldsymbol{e}^r_{m'} | u_{m'} \in \widehat{\mathcal{S}}_m\}$. Specifically, we combine a sigmoid function with a MLP layer over the concatenation of these embeddings as

$$\widehat{y}_{mn} = \mathtt{sigmoid}(f_{\mathsf{MLP}}([\boldsymbol{e}_m, \{\boldsymbol{e}^r_{m'} | u_{m'} \in \widehat{\mathcal{S}}_m\}])). \quad (9)$$

After that, we adopt a log loss to update the parameter θ as

$$\mathcal{L}_\theta = - \sum_{(u_m, i_n) \in \mathcal{D}} (y_{mn} \cdot \log(\widehat{y}_{mn}) + (1 - y_{mn}) \cdot \log(1 - \widehat{y}_{mn})), \quad (10)$$

where $\mathcal{D}$ denotes the datasets containing the true label y_{mn} for each user-item pair (u_m, i_n). We provide a detailed pseudo code of the training process and the corresponding time complexity analysis in Appendix A.

4 EXPERIMENTS

4.1 Dataset and Experimental Flow

We use three large-scale real-world datasets, namely Tmall[2], Taobao[3], Alipay[4], which contain users online records from three corresponding platforms of Alibaba Group. Please refer to Appendix B.1 for detailed description of the datasets and B.3 for detailed experimental configuration.

4.2 Baselines and Evaluation Metrics

We compare our model mainly against 13 representative recommendation methods including LSTM [14], RRN [44], STAMP [23], Time-LSTM [48], NHP [27], DUPN [28], NARM [21], ESMM [26], ESM^2 [43], MMoE [25], DIN [47], DIEN [46], SIM [31]. In order to further investigate the effect from each component of STARec, we design the following three variants:

- **STARec** is our model without using user previous feedbacks for fair comparsion.
- **STARec**$^-_{\mathtt{time}}$ is a variant of STARec using a standard LSTM as the time-aware (sequential) module.
- **STARec**$^-_{\mathtt{recent}}$ is a variant of STARec where $\mathcal{H}^{\mathsf{RECENT}}_m$ is not included in $\widehat{\mathcal{H}}_m$ (see Eq. (2)).
- **STARec**$^+_{\mathtt{label}}$ is a variant of STARec using user's previous feedbacks as input.

We provide the descriptions of these baseline methods in Appendix B.2. We provide detailed descriptions of experimental settings and data pre-processing in Appendix B.3. To evaluate the performance of the above methods, we choose Area under the ROC Curve (AUC), Accuracy (ACC), LogLoss as the evaluation metric. The thresholds of ACC on Tmall and Alipay datasets are set as 0.5, while that on the Taobao dataset is set as 0.1 due to a large number of negative instances.

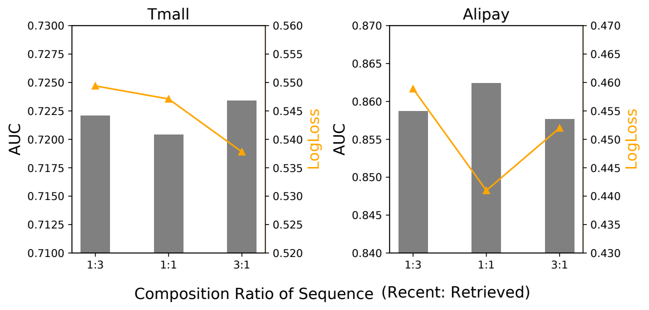

Figure 4: Comparison of performance of STARec under different composition ratios of recent and retrieved items in sequence on Tmall and Alipay datasets, in terms of AUC and LogLoss.

4.3 Result Analysis

Overall Performance. Table 1 summarizes the results. The major findings from our offline experiments are summarized as follows.

- Our model outperforms all these baseline methods including sequential models (e.g., RRN, LSTM, NHP) and tower architecture based mdels (e.g., ESMM, MMoE, ESM^2). These results may be explained as our model, unlike these methods, combining the advantages of both search-based and time-aware (sequential) models.
- Compared to those other models (e.g., ESMM, MMoE, ESM^2), most of the sequential recommendation methods (e.g., RRN, LSTM, NHP) achieve better performance. We may conclude that encoding the contextual information in the historical sequences is crucial to capture user patterns, as whether a user has already found the desired items or not holds a significant effect on user behaviors on the current item.
- With a comparison between SIM and other existing sophisticated models (e.g., DIN, DIEN), we find that SIM consistently outperforms these methods. The reason seems to be that SIM introduces a search-based module to use the retrieved relevant information instead of the whole sequences.

Impact of Recent Histories. From the comparison between STARec and STARec$^-_{\mathrm{recent}}$ in Table 1, we can observe that replacing some retrieved items with the recent items can consistently improve the performance in all the datasets. Specifically, for each dataset, the sequence length of STARec and other baselines is set as 30. Distinct from other methods, half of sequence of STARec includes retrieved

[2]https://tianchi.aliyun.com/dataset/dataDetail?dataId=42
[3]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649
[4]https://tianchi.aliyun.com/dataset/dataDetail?dataId=53

Table 1: Comparison of different (sequential) recommendation models on three industrial datasets. Results of Click-Through Rate (CTR) prediction task are reported. * indicates $p < 0.001$ in significance tests compared to the best baseline. Note that our results are not consistent with the results in [32] due to different experimental settings. Refer to details in Appendix B.3.

Recommender	Tmall			Alipay			Taobao		
	AUC	ACC	LogLoss	AUC	ACC	LogLoss	AUC	ACC	LogLoss
LSTM	0.6973	0.7054	0.5854	0.8357	0.7697	0.4713	0.5912	0.4516	0.4411
LSTM$^+_{\texttt{label}}$	**0.7662**	**0.7324**	**0.5291**	**0.9052**	**0.8449**	**0.3738**	**0.6450**	**0.5780**	**0.4094**
RRN	0.6973	0.7073	0.5866	0.8429	0.7145	0.4636	0.5102	0.1502	0.4398
Time-LSTM	0.6962	0.6796	0.5865	0.8439	0.8057	0.4874	0.5945	0.4393	0.4397
NHP	0.6979	0.6969	0.5849	0.8490	0.7922	0.4743	0.6003	0.3205	0.4393
DUPN	0.5551	0.6796	0.6269	0.8021	0.7552	0.5243	0.5525	0.1921	0.4471
NARM	0.6396	0.6839	0.6025	0.8422	0.7695	0.4860	0.5972	0.3426	0.4494
STAMP	0.6753	0.6946	0.5829	0.8178	0.7491	0.5137	0.6012	0.3592	0.4448
ESMM	0.5189	0.6796	0.6312	0.7131	0.6856	0.6080	0.5487	0.1774	0.4573
ESM^2	0.5149	0.6796	0.6310	0.7241	0.6930	0.5996	0.5030	0.1520	0.4594
MMoE	0.5060	0.6796	0.6313	0.7119	0.6834	0.6085	0.5501	0.1719	0.4565
DIN	0.6878	0.6946	0.5915	0.8496	0.7692	0.4717	0.5978	0.4422	0.4388
DIEN	0.6892	0.6962	0.5833	0.8474	0.7949	0.4668	0.5963	0.4025	0.4412
SIM	0.7005	0.7094	0.5698	0.8549	0.8069	0.4623	0.6045	0.4538	0.4271
STARec$^-_{\texttt{time}}$	0.6999	0.7097	0.5684	0.8527	0.8046	0.4547	0.6035	0.4525	0.4312
STARec$^-_{\texttt{recent}}$	0.7013	0.7081	0.5632	0.8536	0.8012	0.4672	0.6021	0.4561	0.4355
STARec	**0.7204***	**0.7150***	**0.5471***	**0.8624***	**0.8142***	**0.4410***	**0.6126***	**0.4629***	**0.4211***
STARec$^+_{\texttt{label}}$	**0.7986***	**0.7502***	**0.5059***	**0.9201***	**0.8661***	**0.3423***	**0.6771***	**0.6039***	**0.3862***

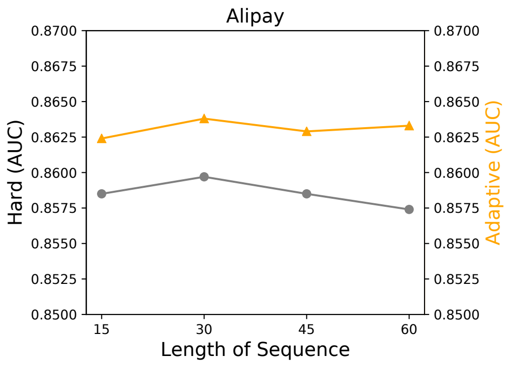

Figure 5: Comparison of performance of STARec with the hard-search or adaptive search strategies under different lengths of sequence on Alipay dataset, in term of AUC.

items, while the other half consists of recent ones. Hence, we here further investigate how the performance of STARec changes when involving more recent ones (and less retrieved ones) or less recent ones (and more retrieved ones). Figure 4 depicts the performance of STARec under three cases. It's difficult to conclude the best ratio in a general way, as the value varies for different datasets.

Impact of Search-based Module. As Table 1 shows, we can see that STARec achieves better performance than STARec$^-_{\texttt{search}}$ in all these three datasets. The observation that SIM works better than DIE and DIEN methods also verifies the superiority of search-based models. As our paper introduces a new adaptive search strategy, we further compare its performance to the hard-search strategy under different sequence lengths. From Figure 5, we see that our proposed adaptive search strategy can consistently outperform the hard-search strategy. One possible explanation is that the hard-search strategy can be regarded as a special case of our adaptive search strategy. Also, we observe that their performance gap gets bigger when the length of the sequence reaches 60. A possible explanation is that the hard-search strategy, at most, only can search and retrieve all the items whose categories are same to the target item, while our adaptive search strategy definitely searches and retrieves items in a larger scope, which can involve more useful information.

Impact of Time-aware Module. In Table 1, we compare STARec to STARec$^-_{\texttt{time}}$. Results show that taking time intervals of user behaviors into consideration can improve the performance of our model, which verifies our idea of building a time-aware module.

Impact of Label Trick. From Table 1, one can see that our label trick (using the previous user feedbacks as the input) can significantly improve the performance of STARec. We further investigate the impact of our label trick with other sequential models (e.g., LSTM). In Table 1, we design LSTM$^+_{\texttt{label}}$, a variant of LSTM that uses user previous feedbacks as the input. Comparison between LSTM and LSTM$^+_{\texttt{label}}$ shows the significant improvement from the label trick, which, to an extent, outweights the gains from more dramatic changes in the underlying user modeling architecture.

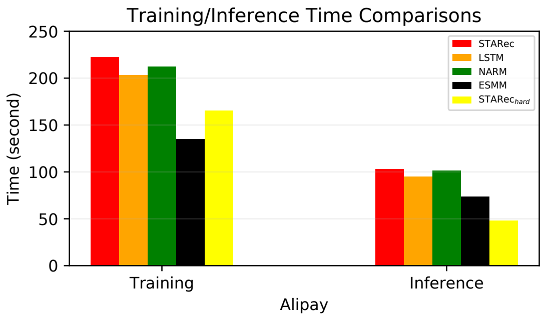

Figure 6: Training/inference time comparisons of STARec and STARec$_{\text{hard}}$ against baselines on Alipay dataset.

Complexity Analysis. We investigate the time complexity of STARec against baseline methods LSTM, NARM, ESMM, and further introduce STARec$_{\text{hard}}$ as a variant of STARec using the hard-search strategy. We then report the training and inference times for one round of the whole data. From Figure 6, we can observe that STARec$_{\text{hard}}$ is more efficient than STARec, as our adaptive search strategy needs to compute the similarity of category embeddings. More importantly, we also can see that the training and inference times of STARec$_{\text{hard}}$ are comparable to, or even smaller than, other baselines. One explanation is that we employ two sequential networks to model the recent items and retrieved items in STARec and STARec$_{\text{hard}}$. Hence, the length of our time-aware module is half of the length of these baselines leading to an efficient implementation.

5 REAL-WORLD DEPLOYMENT

In order to verify the effectiveness of STARec in real-world applications, we deploy our method in two main item recommendation scenarios (called "Guess You Like" and "Information Flow") in Company X, a main-stream bank company. This App has millions of daily active users who create billions of user logs every day in the form of implicit feedbacks such as click behavior. Please refer to discussions on deployment in Appendix C.

Table 2: Comparison of different (sequential) recommendation models on real-world recommendation scenarios.

Recommender	Guess You Like		Information Flow	
	AUC	CTR	AUC	CTR
DIN	0.7828	1.47%	0.8409	1.26%
DIEN	0.8068	1.52%	0.8493	1.30%
STARec	**0.8909**	**1.61%**	**0.9159**	**1.32%**

5.1 Offline Evaluation

For the offline experiment, we use a daily updated dataset collected from September 11th, 2021 to October 10th, 2021 for training and evaluation. Concretely, for each user, we use the last 31 user behaviors as the test set and use the rest records as the training set. The task is CTR prediction, and the overall performance is shown in Table 2. From Table 2, we see that STARec achieves 13.8% and 8.9% improvement over DIN [47], 10.7% and 7.8% improvement over DIEN [46] on AUC metric in "Guess You Like" and "Information Flow" scenarios respectively.

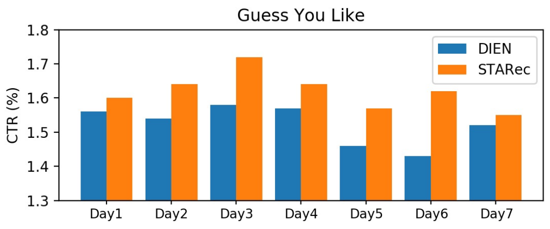

Figure 7: Daily results of online A/B test in "Guess You Like" scenario on CTR metric.

5.2 Online Evaluation

For the online experiment, we conduct A/B testing in two recommendation scenarios in Company X's online App, comparing the proposed model STARec with the current production baseline methods DIN and DIEN. The whole online experiment lasts a week, from October 14, 2021 to October 21, 2021. In the "Guess You Like" scenario, 24.1% and 26.68% of the users are presented with the recommendation by DIN and DIEN, respectively, while 24.2% of the users are presented with the recommendation by STARec. And, in "Information Flow", 25.4% and 24.8% of the users are presented with the recommendation by DIN and DIEN respectively; while 24.5% of the users are presented with the recommendation by STARec. We examine CTR metric defined as CTR = $\frac{\#\text{clicks}}{\#\text{impressions}}$ where #clicks and #impressions are the number of clicks and impressions. We report the average results in Table 2 and depict daily improvement of STARec over DIEN in Figure 7 in the "Guess You Like" scenario. From the table, we can see that STARec performs better in "Guess You Like" than "Information Flow". One reason is that users' browsing lengths in "Information Flow" are much smaller than the lengths in "Guess You Like", which limits the performance of our search-based and time-aware modules. Another reason would be that compared to "Guess You Like", items in "Information Flow" are much more diverse, which includes shopping coupons and cinema tickets, besides items in "Guess You Like", making searching for relevant items much harder. From the figure, we can see the CTR improvements are rather stable where the improvement of STARec fluctuates in the range of 2% to 13%.

6 CONCLUSION

In this paper, we propose a novel search-based time-aware model named STARec, where we design an adaptive search-based module to retrieve relevant items and then feed this information into a time-aware (sequential) module to capture user evolving interests. We also design a novel label trick that allows the model to use user's previous feedbacks as the input, and reveal that this label information can significantly improve the performance. For future work, we plan to further deploy search-based models in other real-world scenarios with sequential data.

Acknowledgments. This work is supported by China Merchants Bank Credit Card Center. The Shanghai Jiao Tong University Team is supported by Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102) and National Natural Science Foundation of China (62076161, 62177033). We would also like to thank Wu Wen Jun Honorary Doctoral Scholarship from AI Institute, Shanghai Jiao Tong University.

REFERENCES

[1] Ting Bai, Ji-Rong Wen, Jun Zhang, and Wayne Xin Zhao. 2017. A neural collaborative filtering model with interaction-based neighborhood. In *CIKM*.

[2] Ting Bai, Lixin Zou, Wayne Xin Zhao, Pan Du, Weidong Liu, Jian-Yun Nie, and Ji-Rong Wen. 2019. CTrec: A long-short demands evolution model for continuous-time recommendation. In *SIGIR*.

[3] Alex Beutel, Paul Covington, Sagar Jain, Can Xu, Jia Li, Vince Gatto, and Ed H Chi. 2018. Latent cross: Making use of context in recurrent recommender systems. In *WSDM*.

[4] Sotirios P Chatzis, Panayiotis Christodoulou, and Andreas S Andreou. 2017. Recurrent latent variable networks for session-based recommendation. In *DLRS*.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*.

[6] Hanjun Dai, Yichen Wang, Rakshit Trivedi, and Le Song. 2016. Deep coevolutionary network: Embedding user and item features for recommendation. *arXiv preprint arXiv:1609.03675* (2016).

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[8] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential user-based recurrent neural network recommendations. In *RecSys*.

[9] Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. 2016. Recurrent marked temporal point processes: Embedding event history to vector. In *KDD*.

[10] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI*.

[11] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW*.

[12] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *CIKM*.

[13] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. In *ICLR*.

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[15] Seyed Abbas Hosseini, Ali Khodadadi, Keivan Alizadeh, Ali Arabzadeh, Mehrdad Farajtabar, Hongyuan Zha, and Hamid R Rabiee. 2018. Recurrent poisson factorization for temporal recommendation. *TKDE* (2018).

[16] Jiarui Jin, Jiarui Qin, Yuchen Fang, Kounianhua Du, Weinan Zhang, Yong Yu, Zheng Zhang, and Alexander J Smola. 2020. An efficient neighborhood-based interaction model for recommendation on heterogeneous graph. In *KDD*.

[17] How Jing and Alexander J Smola. 2017. Neural survival recommender. In *WSDM*.

[18] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. 2016. Field-aware factorization machines for CTR prediction. In *Proceedings of the 10th ACM conference on recommender systems*.

[19] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD*.

[20] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[21] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *CIKM*.

[22] Qiang Liu, Shu Wu, Diyi Wang, Zhaokang Li, and Liang Wang. 2016. Context-aware sequential recommendation. In *ICDM*.

[23] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: short-term attention/memory priority model for session-based recommendation. In *KDD*.

[24] Yanchi Liu, Chuanren Liu, Bin Liu, Meng Qu, and Hui Xiong. 2016. Unified point-of-interest recommendation with temporal interval assessment. In *KDD*.

[25] Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H Chi. 2018. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *KDD*.

[26] Xiao Ma, Liqin Zhao, Guan Huang, Zhi Wang, Zelin Hu, Xiaoqiang Zhu, and Kun Gai. 2018. Entire space multi-task model: An effective approach for estimating post-click conversion rate. In *SIGIR*.

[27] Hongyuan Mei and Jason Eisner. 2017. The neural hawkes process: A neurally self-modulating multivariate point process. *Neurips* (2017).

[28] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. 2018. Perceive your users in depth: Learning universal user representations from multiple e-commerce tasks. In *KDD*.

[29] Wenjie Pei, Jie Yang, Zhu Sun, Jie Zhang, Alessandro Bozzon, and David MJ Tax. 2017. Interacting attention-gated recurrent networks for recommendation. In *CIKM*.

[30] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on long sequential user behavior modeling for click-through rate prediction. In *KDD*.

[31] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based user interest modeling with lifelong sequential behavior data for click-through rate prediction. In *CIKM*.

[32] Jiarui Qin, Weinan Zhang, Xin Wu, Jiarui Jin, Yuchen Fang, and Yong Yu. 2020. User behavior retrieval for click-through rate prediction. In *SIGIR*.

[33] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *ICDM*.

[34] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *RecSys*. 130–137.

[35] Kan Ren, Jiarui Qin, Yuchen Fang, Weinan Zhang, Lei Zheng, Weijie Bian, Guorui Zhou, Jian Xu, Yong Yu, Xiaoqiang Zhu, et al. 2019. Lifelong sequential modeling with personalized memorization for user response prediction. In *SIGIR*.

[36] Steffen Rendle. 2010. Factorization machines. In *ICDM*.

[37] Massimiliano Ruocco, Ole Steinar Lillestøl Skrede, and Helge Langseth. 2017. Inter-session modeling for session-based recommendation. In *DLRS*.

[38] Yunsheng Shi, Zhengjie Huang, Wenjin Wang, Hui Zhong, Shikun Feng, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509* (2020).

[39] Yong Kiam Tan, Xinxing Xu, and Yong Liu. 2016. Improved recurrent neural networks for session-based recommendations. In *DLRS*.

[40] Bjørnar Vassøy, Massimiliano Ruocco, Eliezer de Souza da Silva, and Erlend Aune. 2019. Time is of the essence: a joint hierarchical rnn and point process model for time and item predictions. In *WSDM*.

[41] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *SIGIR*.

[42] Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. 2021. Bag of Tricks for Node Classification with Graph Neural Networks. *arXiv preprint arXiv:2103.13355* (2021).

[43] Hong Wen, Jing Zhang, Yuan Wang, Fuyu Lv, Wentian Bao, Quan Lin, and Keping Yang. 2020. Entire Space Multi-Task Modeling via Post-Click Behavior Decomposition for Conversion Rate Prediction. *SIGIR*.

[44] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *WSDM*.

[45] Pengpeng Zhao, Haifeng Zhu, Yanchi Liu, Zhixu Li, Jiajie Xu, and Victor S Sheng. 2018. Where to go next: A spatio-temporal LSTM model for next POI recommendation. *arXiv preprint arXiv:1806.06671* (2018).

[46] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *AAAI*.

[47] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD*.

[48] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. 2017. What to Do Next: Modeling User Behaviors by Time-LSTM.. In *IJCAI*.

A PSEUDOCODE OF STAREC TRAINING PROCEDURE

In this section, we provide a detailed pseudo code of the training process in Algorithm 1. We analyze its time complexity as follows. There are two main components in STARec, namely search-based and time-aware modules. For each user-item pair (e.g., (u_m, i_n)), similar as analysis in [32], we need to retrieve $\widehat{T}$ (i.e., $|\widehat{\mathcal{H}}_m|$) items from the whole T (i.e., $|\mathcal{H}_m|$) browsed items in item search-based module, which costs $O(1)$ time; while in user search-based module, similarly, time complexity for retrieving all relevant users (i.e., $|\widehat{\mathcal{S}}_m|$) is $O(1)$. Hence, the overall cost for search-based module is $O(1 + |\widehat{\mathcal{S}}_m|)$ for each user. Assume that the average case time performance of recurrent units is $O(C)$. For each user, there are $1+|\widehat{\mathcal{S}}_m|$ sequences to be fed into the time-aware sequence module. Let $\widehat{T}$ be the average length of these sequences. Then, the overall complexity of time-aware module is $O((1 + |\widehat{\mathcal{S}}_m|) \cdot C\widehat{T})$. Combining all the modules together, the overall complexity of STARec is $O((1 + |\widehat{\mathcal{S}}_m|) \cdot (1 + C\widehat{T})MN)$.

Algorithm 1 STARec

```
INPUT: dataset $\mathcal{D} = \{(u_m, i_n)\}_{m=1,n=1}^{M,N}$ with histories $\{\mathcal{H}_m\}_{m=1}^{M}$;
OUTPUT: STARec recommender with parameter $\theta$
1:  Initialize all parameters.
2:  repeat
3:      Randomly sample a batch $\mathcal{B}$ from $\mathcal{D}$
4:      for each data instance $(u_m, i_n)$ in $\mathcal{B}$ do
5:          Calculate embedding vectors $\boldsymbol{e}^u_m$, $\boldsymbol{e}^i_n$ using Eq. (1).
6:          Construct a set of relevant items $\widehat{\mathcal{H}}_m$ using Eqs. (2)(3).
7:          Construct a set of relevant users $\widehat{\mathcal{S}}_m$ using Eqs. (4)(5).
8:          Compute the hidden states of each sequence for $\widehat{\mathcal{H}}_m$ and
            $\{\widehat{\mathcal{H}}_{m'}|u_{m'} \in \widehat{\mathcal{S}}_m\}$ using Eqs. (6)(7).
9:          Encode each sequence and obtain $\boldsymbol{e}_m$ and $\{\boldsymbol{e}^r_{m'}|u_{m'} \in \widehat{\mathcal{S}}_m\}$
            using Eq. (8).
10:         Fuse all information to generate $\widehat{y}_{mn}$ using Eq. (9).
11:     end for
12:     Update $\theta$ by minimizing $\mathcal{L}_\theta$ according to Eq. (10).
13: until convergence
```

B EXPERIMENTAL CONFIGURATION

B.1 Dataset Description

We use three large-scale real-world datasets for the evaluations, and provide the detailed decription for each dataset as follows.

- **Tmall**[5] is a dataset consisting of 54,925,331 interactions of 424,170 users and 1,090,390 items. These sequential histories are collected by Tmall e-commerce platform from May 2015 to November 2015 with the average sequence length of 129 and 9 feature fields.
- **Taobao**[6] is a dataset containing 100,150,807 interactions of 987,994 users and 4,162,024 items. These user behaviors including several behavior types (e.g., click, purchase, add to chart, item favoring) are collected from November 2007 to December 2007 with average sequence length of 101 and 4 feature fields.
- **Alipay**[7] is a dataset collected by Alipay, an online payment application from July 2015 to November 2015. There are 35,179,371 interactions of 498,308 users and 2,200,191 items with average sequence length of 70 and 6 feature fields.

B.2 Baseline Description

In our paper, we compare our method against 13 strong baselines. As STARec is proposed in the context of sequential data, most of these methods are sequential models. We provide brief descriptions as follows.

- **LSTM** [14] is a standard long short memory approach widely used for modeling user's sequential pattern.
- **RRN** [44] is a representative approach using RNN to capture the dynamic representation of users and items.
- **STAMP** [23] is a user action-based prediction, which models user general preference and current interest.
- **Time-LSTM** [48] is a extension of LSTM, which considers time intervals in sequence by the time gates.
- **NHP** [27] is a neural Hawkes process approach which uses a self-modulating multivariate point process to model user behaviors.
- **DUPN** [28] is a representative learning method, which shares and learns in an end-to-end setting across user's multiple behaviors.
- **NARM** [21] is a sequential recommendation model, which uses an attention mechanism to model influence of user behaviors.
- **ESMM** [26] employs a feature representation transfer learning strategy over user's various behaviors.
- **ESM^2** [43] designs a user behavior decomposition to model user's various behaviors.
- **MMoE** [25] is a neural-based algorithm for user modeling by sharing the expert sub-models across various behaviors.
- **DIN** [47] designs a local activation unit to capture user interests from historical behaviors.
- **DIEN** [46] is an extension of DIN which captures user's evolving interests from historical behavior sequence.
- **SIM** [31] is search-based user interest model, which extracts user interests with general search units and exact search unit.

In order to further investigate the effect from each component of STARec, we design the following three variants:

- **STARec** is our model without using user previous feedbacks for fair comparsion.
- **STARec$^-_{\texttt{time}}$** is a variant of STARec using a standard LSTM as the time-aware (sequential) module.
- **STARec$^-_{\texttt{recent}}$** is a variant of STARec where $\mathcal{H}^{\mathsf{RECENT}}_m$ is not included in $\widehat{\mathcal{H}}_m$ (see Eq. (2)).
- **STARec$^+_{\texttt{label}}$** is a variant of STARec using user's previous feedbacks as input.

B.3 Experimental Setting

We split the datasets using the timestep. For simplicity, let T denote the sequence length of user browsing logs. The training dataset contains the 1st to $(T-2)$-th user behaviors, where we use 1-st to $(T-3)$-th user records to predict the user behavior at $T-2$.

[5]https://tianchi.aliyun.com/dataset/dataDetail?dataId=42
[6]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649
[7]https://tianchi.aliyun.com/dataset/dataDetail?dataId=53

In validation set, we use 1-st to $(T-2)$-th user records to predict $(T-1)$-th user behavior, and in the test set, we use 1-st to $(T-1)$-th behaviors to predict T-th behavior. The learning rate is decreased from the initial value 1×10^{-2} to 1×10^{-6} during the training process. The batch size is set as 100. The weight for L2 regularization term is 4×10^{-5}. The dropout rate is set as 0.5. The dimension of embedding vectors is set as 64. All the models are trained under the same hardware settings with 16-Core AMD Ryzen 9 5950X (2.194GHZ), 62.78GB RAM, NVIDIA GeForce RTX 3080 cards. Note that the major difference of experiment settings between our paper and [32] is that we directly use click signals in the raw data as the positive feedbacks, and the negative instances are those not clicked items; while Qin et al. [32] regards the last item as the instance receiving positive feedbacks, and randomly sample items that do not appear in the dataset as the negative samples.

C DEPLOYMENT DISCUSSION

In this section, we introduce our hands-on experience of implementing STARec in the display advertising system with top-K recommendation and learning-to-rank tasks in Company X. As industrial recommender or ranker systems need to process massive traffic requests per second, it's hard to make a long-term sequential user interest model serving in real-time industrial system. As discussed in [30, 31], the storage and latency constraints could be main bottlenecks.

C.1 Extension to Ranking Task

In seeking to reduce the computation costs, we begin with clarifying two aforementioned tasks, namely top-K recommendation and learning-to-rank tasks. As introduced in *Definition 2.1*, the original task is a point-wise recommendation, which aims to generate similarity score for each given user-item pair. However, in the real-world scenario, top-K recommender and ranker systems are always required to provide a list of K items for each user, whose formal definition is provided as follows.

Definition C.1. **Top-K Ranker or Recommender System**[8]. Given a tuple $\langle\mathcal{U},\mathcal{I},\mathcal{C},\mathcal{Q}\rangle$ where $\mathcal{Q}$ is the set of queries in ranking, the goal of the top-K ranker or recommender system is to provide a list of K items $\mathcal{L}_m=\{i_1,i_2,\dots,i_K\}$ where $i_k\in\mathcal{I}$ for each user $u_m\in\mathcal{U}$ starting at a future time $T+1$.

One principle way is to first calculate the similarity for each item and then rank candidate items at descending order of their similarities. However, the complexity of this approach prevents it to serve online, which mainly boils down to following reasons.

- **(R1)** As there are numerous items of various categories, our search-based module, which treats item category as key for search, needs to run multiple times, leading to high time computation.
- **(R2)** As existing search-based model [31] chooses the hard-search to save computation cost, thus, it's a great challenge to efficiently deploy our adaptive search-based module to the online system.

To mitigate this issue, we provide the following solutions.

Mapping Queries/Demands to Certain Categories. For the first issue, we consider to reduce the scope of candidate item categories that users may be interested in. In light of this, we introduce a mapping function building relation between user queries and item categories, namely a mapping model $f_{\mathsf{MAP}}:\mathcal{Q}\rightarrow\mathcal{C}$. For example, in Figure 1, a teen u_1 would type "lipstick" in the search box, and then f_{MAP} returns category "cosmetics". In this case, we only need to search and retrieve those items with cosmetics category for user u_1, which can significantly reduce the computation cost of searching and retrieving. Moreover, in some recommendation scenario lack of query information, we are also able to construct a mapping model, whose input is users' recent historical records and output is several item categories that users may be interested in, namely $f_{\mathsf{MAP}}:\mathcal{H}\rightarrow\mathcal{C}$. Taking Figure 1 as an instance, after viewing u_1's recent browsed items $\mathcal{H}_1$, f_{MAP} would return category "cosmetics", as most of her recent interests lie in cosmetics.

Saving Latency by Periodic Update. As introduced in [31], one practical way is to conduct the hard-search strategy, which is a trade-off between performance gain and resource consumption. We argue that the soft-search in our search-based module is based on similarities among embedding vectors of item categories instead of item themselves, which is much easier to learn and efficient to compute. Besides this, we also provide a periodic update approach. Our approach share the same spirit with [31] to build an two-level structured index for each user in an offline manner to save online latency. Based on this structure, we also pre-compute, store, and periodically update those relevant item categories $c^i_{n'}$ satisfying $f^i_{\mathsf{ADA}}(c^i_{n'},c^i_n,x^{i1}_{n'},x^{i1}_n)\geq\epsilon$ for each item category c^i_n. Considering that involving relevant users cause slight performance gain but huge computation cost, we choose to not include this part of STARec in our deployment.

C.2 Extension to Delayed Feedback

We reveal that another issue in practice is delayed feedback caused by heavy traffic in the online system. Formally, several labels in a user's (e.g., u_m's) retrieved historical records $\widehat{\mathcal{H}}_m$ would be missing. Consider that this issue would be amplified, as STARec explicitly includes the user feedbacks in the proposed label tricks. We propose to use *predicted labels* generated from our model to replaced those missing *original labels.*

[8]This definition is proposed based on, and shares the same notations with *Definition 2.1.*