Sequential Recommendation with User Evolving Preference Decomposition

Weiqi Shao
Xu Chen*
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing 100872, China
shaoweiqi@ruc.edu.cn
successcx@gmail.com

Jiashu Zhao
Department of Physics and Computer Science, Wilfrid Laurier University
Canada
jzhao@wlu.ca

Long Xia
Baidu Inc, China
long.phil.xia@gmail.com

Jingsen Zhang
Gaoling School of Artificial Intelligence, Renmin University of China, Beijing 100872, China
zhangjingsen@ruc.edu.cn

Dawei Yin
Baidu Inc, China
yindawei@acm.org

ABSTRACT

Modeling user sequential behaviors has recently attracted increasing attention in the recommendation domain. Existing methods mostly assume coherent preference in the same sequence. However, user personalities are volatile and easily changed, and there can be multiple mixed preferences underlying user behaviors. To solve this problem, in this paper, we propose a novel sequential recommender model via decomposing and modeling user independent preferences. To achieve this goal, we highlight three practical challenges considering the inconsistent, evolving and uneven nature of the user behaviors. For overcoming these challenges in a unified framework, we introduce a reinforcement learning module to simulate the evolution of user preference. More specifically, the action aims to allocate each item into a sub-sequence or create a new one according to how the previous items are decomposed as well as the time interval between successive behaviors. The reward is associated with the final loss of the learning objective, aiming to generate sub-sequences which can better fit the training data. We conduct extensive experiments based on eight real-world datasets across different domains. Comparing with the state-of-the-art methods, empirical studies manifest that our model can on average improve the performance by about 9.68%, 12.4%, 8.56% and 7.13% on the metrics of Precision, Recall, NDCG and MRR, respectively.

KEYWORDS

Sequential Recommendation, Reinforcement Learning, User Behavior Analysis.

[*] Corresponding author

SIGIR-AP '23, November 26–28, 2023, Beijing, China
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0408-6/23/11…$15.00
https://doi.org/10.1145/3624918.3625312

ACM Reference Format:
Weiqi Shao, Xu Chen, Jiashu Zhao, Long Xia, Jingsen Zhang, and Dawei Yin. 2023. Sequential Recommendation with User Evolving Preference Decomposition. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region (SIGIR-AP '23), November 26–28, 2023, Beijing, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3624918.3625312

1 INTRODUCTION

Recommender system has been deployed in a wide range of applications, ranging from the fields of e-commerce [19, 22, 37], education [21, 28] to health-caring [2, 13, 46] and entertainment [1, 25, 31]. Early recommender models like matrix factorization [6, 15, 41] usually assume that the user behaviors are independent. However, user preferences in real-world scenarios is an evolving process, and successive behaviors can be highly correlated. For example, the purchasing of a mobile phone may trigger the interaction with the phone case, which may further attract user interest on phone holders. Motivated by such characters, people have designed a lot of sequential recommender models [12]. For example, FPMC [27] regards user behaviors as a Markov chain, where the current behavior is only influenced by the most recent action. GRU4Rec [16] leverages recurrent neural network to summarize all the history behaviors for the next item prediction.

While sequential recommendation has achieved many promising results, existing methods usually assume coherent user preference in a sequence. However, user personalities are complicated and diverse in practice, thus the same sequence may contain multiple user preferences. As exampled in the top block of Figure 1, the user sequentially interacts with the items of "camera → camera lens → baby chair → lens cleaner → hanging toy". Their are two types of user preference. One is on digital items, and the other is about baby products. The user starts with the first preference, and purchases the "camera" and "camera lens". And then, by interacting with the "baby chair", the user moves to the second preference. In the next, the user returns to the first preference, and continues to buy the "lens cleaner". At last, the second preference is triggered again via the interaction with the "hanging toy". In this process, the user preference evolves along different threads. For the digital items, the evolving path is "camera → camera lens → lens cleaner". For the baby products, the user preference evolves from the "baby chair" to the "hanging toy". Obviously, different preference threads have diverse evolving patterns, decomposing them and modeling each thread separately can lead to more clear history representation and capture more distinct user intents for better recommendation performance. Despite such desirable advantages, separating real-world user behaviors can be much more challenge because:

CH1: *User behaviors are inconsistent.* For different users, the number of preference threads may vary. For example, in the top and middle blocks of Figure 1, user X has two types of preferences on the digital and baby products, while the behaviors of user Y follow three preference threads along the book, sports and digital items, respectively. Even for the same user, different behavior sequences may also have various number of preference threads, which is exampled in the middle and bottom blocks of Figure 1. In practice, how to handle such inconsistency across different sequences is challenging, since one cannot pre-define a unified thread number, and manually check each sequence is too labor intensive.

CH2: *User behaviors are evolving processes.* Straightforwardly, sequence decomposition can be seen as a classification problem, where one can predict the thread label for each item separately, and assemble the items with the same label into the final thread. However, decomposing user behaviors is more complex. See the example in the middle block of Figure 1, in the beginning, the user interacted with many digital products. And then, she moved to the sports items, and purchased the sport shirt, sweatpants and smart-watch. If we classify the smart-watch independently, it may belong to the digital products, while by taking the user behaviors as an evolving process, we know that the reason of purchasing smart-watch is more likely for sports, such as timing and counting calories. Such evolving nature is important and unique for user behavior decomposition, but how to model it is still under-explored.

CH3: *User behaviors are unevenly distributed on the timeline.* Different from sequence decomposition tasks in other domains like NLP [3] and CV [11], a unique character of our task is the significance of the time interval information between successive behaviors. Intuitively, if two behaviors happen with a large time interval, then they may have less correlations, and should be decomposed into different threads. In the bottom block of Figure 1, while "C1 → C2 → C4" and "C6→ C7" both reflect user preference on the digital items, they may happen independently, since their interaction times have a large gap. How to model such temporal information is important yet not well studied.

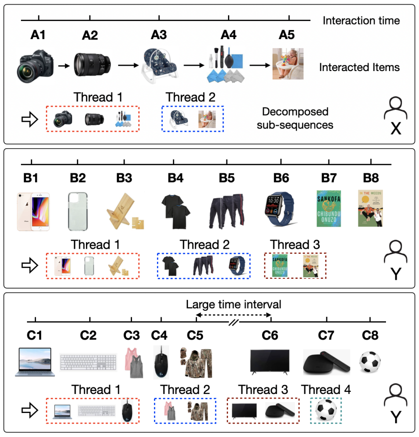


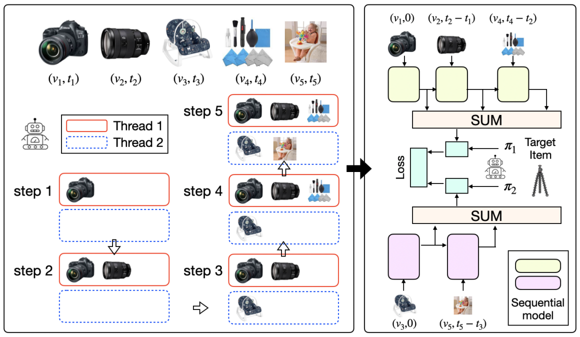
Figure 1: Examples of user inconsistent, evolving and uneven multi-thread preferences.

Table 1: Comparisons between multi-interest and our models on solving the challenges mentioned in the introduction.

Model	CH1	CH2	CH3
SUM [20]	-	-	-
SINE [33]	-	-	-
DMIN [40]	-	-	-
MDSR [8]	-	-	-
ComiRec [5]	-	-	-
MCPRN [37]	-	-	-
MIND [18]	✓	-	-
Octopus [23]	✓	-	-
PIMI [7]	-	-	✓
Our Model	✓	✓	✓

In order to overcome the above problems, in this paper, we design a reinforcement learning (RL) method to decompose user evolving preferences. The agent simulates the generation of user behaviors, which outputs a set of sub-sequences representing user multi-thread preferences. At each step, it decides which sub-sequence the current item should be allocated to or creates a new sub-sequence. The reward is associated with the loss of the learning objective, aiming to obtain the allocation schemes which can better fit the training data. In addition, we introduce auxiliary rewards to encourage that the items in a sub-sequence are coherent, while the representations of different sub-sequences are as independent as possible. To demonstrate the effectiveness of the designed model, we conduct extensive experiments by comparing with the state-of-the-art baselines based on a series of real-world datasets. We have noticed that there are some studies on multi-interest recommendation. For example, SUM [20] and LimaRec [39] leverage attention model to determine item allocations. MIND [18] and ComiRec [5] use capsule network and dynamic routing to separate different sub-sequences. However, most of these models are not fully aware of the above challenges (*i.e.*, CH1 to CH3), which may lead to sub-optimal model designs and lowered recommendation performance. For clearly understanding the differences between our models and these previous work, we compare them in Table 1.

In a summary, the main contributions of this paper can be concluded as follows: (1)We propose to build sequential recommender models via decomposing user independent preferences, where we highlight three practical challenges brought by the inconsistent, evolving and uneven nature of the user behaviors. (2) We design a reinforcement learning model to seamlessly overcome the above challenges in a unified framework, where we adaptively separate the complete user behavior sequence into many independent sub-sequences for better fitting the training data. (3) We conduct extensive experiments based on eight real-world datasets to demonstrate the superiority of our model.



(a) Item Allocator (b) Sub-sequence Models

Figure 2: Illustration of our model: (a) is the item allocator, which sequentially assigns the items into different sub-sequences. (b) is the sub-sequence modelers, which process the decomposed user behaviors from (a) to generate the final recommendation list.

2 PRELIMINARIES

2.1 Sequential Recommendation

In sequential recommendation, the current user behavior is predicted by taking its history information into consideration. Formally, we have a user set $\mathcal{U}$ and an item set $\mathcal{V}$. The interactions of each user $u \in \mathcal{U}$ are chronologically organized into a set $O_u = \{(v_1^u, t_1^u), (v_2^u, t_2^u), ..., (v_{l_u}^u, t_{l_u}^u)\}$, where t_i^u is the interaction time of item $v_i^u \in \mathcal{V}$, and l_u is the number of interacted items. We denote by $O = \{O_u\}$ the set of all user-item interactions. Then given $\{\mathcal{U}, \mathcal{V}, O\}$, sequential recommendation aims to learn a model f, which can accurately predict the next item $v_{l_u+1}^u$ for user u based on O_u and $t_{l_u+1}^u$. It should be noted that, as a common simplification, the time information can be omitted [17, 19, 32].

In the training process, O_u is recurrently separated into many samples $\{H_{T+1}^u, v_{T+1}^u\}_{T=1}^{l_u-1}$, where $H_{T+1}^u = [u, (v_1^u, t_1^u), ..., (v_T^u, t_T^u), t_{T+1}^u]$. Then, the binary cross-entropy loss is used to learn f, that is, $L_1 = -\sum_{u=1}^{|\mathcal{U}|} \sum_{T=1}^{l_u-1} [\log f(H_{T+1}^u, v_{T+1}^u) + \sum_{k \in S^-(v_{T+1}^u)} \log(1 - f(H_{T+1}^u, v_k^u))]$, where the output of $f(H_{T+1}^u, v_{T+1}^u)$ is the probability of interacting with v_{T+1}^u given the history information H_{T+1}^u. $S^-(v_{T+1}^u)$ is the set of negative samples, which can be sampled from the non-interacted items of user u. To model user diverse user personalities, recent years have witnessed many multi-interest sequential recommender models (MIRM) [7, 20, 37, 39, 40]. However, as mentioned before, these models fully or partially ignore the challenges of CH1 to CH3 (see Table 1), which are key to decompose user behaviors in real-world scenarios. By designing models tailored for these challenges, we propose a sequential recommender model with adaptive user evolving preference decomposition (called SPLIT for short), which significantly differs from the previous work.

3 THE SPLIT MODEL

There are two major components in our model (see Figure 2). The first one is an item allocator, which aims to project the items into different preference threads. The second one is a set of sub-sequence[1] modelers, which are leveraged to handle the decomposed user behaviors. In the training process, the item allocator is learned to separate each user behavior sequence into many sub-sequences, such that the training loss can be lowered. In addition, by designing auxiliary rewards, different items within the same sub-sequence are expected to be coherent in semantic, while the representations of different sub-sequences should be as independent as possible. In order to control the number of the generated sub-sequences, we also design a reward to penalize the action of "creating a new sub-sequence" in a curriculum learning manner. In the serving process, the item allocator first decomposes the complete user behavior sequence, and then the final recommendation is produced by pooling the results from the sub-sequence modelers. In the following, we introduce these components more in detail.

3.1 Item Allocator

In most multi-interest sequential recommender models [5, 8, 18], the sub-sequences are generated by processing each item independently. However, as analyzed above, the observed user behaviors are the evolution results of user multi-thread preferences. In order to accurately simulate the evolution process, we regard the decomposition of user preferences as a Markov decision process (MDP), and design an RL based item allocator to separate the user behavior sequences. Formally, suppose we have a training sample (H_{T+1}, v_{T+1}), where $H_{T+1} = [u, (v_1, t_1), (v_2, t_2), ..., (v_T, t_T), t_{T+1}]$. Then, our task is to decompose H_{T+1} into different sub-sequences[2], which can better predict v_{T+1}. To achieve this goal, we define the following Markov decision process:

State ($\boldsymbol{st}_i$): The state at step i is $(\boldsymbol{s}_i, t_i)$. $\boldsymbol{s}_i$ is the set of existing sub-sequences before time step i, that is,

$$\boldsymbol{s}_i = \{\boldsymbol{s}_{i,b}\}_{b=1}^{k} = \{[u, (v_{i,b,1}, t_{i,b,1}), ..., (v_{i,b,l_b}, t_{i,b,l_b})]\}_{b=1}^{k}, \tag{1}$$

where $v_{i,b,1}$ is an item in $\{v_k\}_{k=1}^{T}$, and $t_{i,b,1}$ is the corresponding interaction time. k is the number of sub-sequences, which is initialized as 1, and gradually increased in the item allocation process. l_b is the number of items in sub-sequence b. Obviously, $\sum_{b=1}^{k} l_b = i - 1$.

Action (a_i): The action at step i determines how to allocate the current item v_i. If v_i is coherent with an existing sub-sequence, then it will be put into it. Otherwise, the agent will create a new sub-sequence initialized by v_i.

Formally, the action space is $[1, 2, ..., k, k + 1]$, where $a_i = b$ ($b \in [1, k]$) means allocating v_i to sub-sequence b, and $a_i = k+1$ indicates creating a new sub-sequence. Remarkably, such action design is the key to accurately simulate user evolving preference, where the user can not only continue her old preferences, but also can launch a new interest. After taking action a_i, $\boldsymbol{s}_i$ is updated to $\boldsymbol{s}_{i+1}$ in a deterministic manner, that is:

$$\boldsymbol{s}_{i+1} = \begin{cases} \{\boldsymbol{s}_{i,1}, ..., \boldsymbol{s}_{i,b} \cup [(v_i, t_i)], ..., \boldsymbol{s}_{i,k}\}, & a_i = b \in [1, k] \\ \{\boldsymbol{s}_{i,1}, ..., \boldsymbol{s}_{i,k}, [(v_i, t_i)]\}, & a_i = k + 1 \end{cases} \tag{2}$$

The state at step $i + 1$ is $\boldsymbol{st}_{i+1} = (\boldsymbol{s}_{i+1}, t_{i+1})$.

Agent (π): At step i, the agent outputs the action distribution for allocating v_i given the state $\boldsymbol{st}_i$. In specific, each sub-sequence in $\boldsymbol{s}_i$ is firstly processed by a sequential model g_s to derive the sub-sequence representation, that is, $\boldsymbol{c}_{i,b} = g_s(\boldsymbol{s}_{i,b})$, where we delay the

[1] From now on, we interchangeably use the terms of "sub-sequence" and "preference thread".

[2] Here we omit the upper script u when there is no confusion

specification of g_s in later sections. Given $\boldsymbol{c}_{i,b}$, the coherent score between v_i and the sub-sequence $\boldsymbol{s}_{i,b}$ is computed as:

$$cs_b = \text{dist}([\boldsymbol{c}_{i,b}; \boldsymbol{e}_i]) \cdot \phi(t_i - t_b), \tag{3}$$

where $\boldsymbol{e}_i$ is the embedding of v_i, $[\cdot;\cdot]$ is the concatenate operation. "dist" is implemented with a three-layer fully connected neural network, measuring the similarity between v_i and $\boldsymbol{s}_{i,b}$ in semantic. ϕ is a monotonically decreasing function. t_b represents the sub-sequence time, which is set as the interaction time of the last item in $\boldsymbol{s}_{i,b}$ (*i.e.*, t_{i,b,l_b}). In this equation, we derive the overall coherent score based on two aspects, that is, the semantic similarity and the temporal influence. Since ϕ is monotonically decreasing, the larger the time interval $(t_i - t_b)$ is, the smaller the overall coherent score cs_b is. This design encodes the intuition that the behaviors with larger time intervals have less correlations.

In ordinary reinforcement learning method, the action space is fixed. However, an important aspect for simulating user evolving preference is modeling the emergence of new preference threads. To satisfy this requirement, we design a simple but effective method to adaptively expand the action space. In specific, we introduce a threshold ϵ, and define a novel activation function "$\epsilon-$softmax" to implement π, that is: $\pi(a_i = b|(\boldsymbol{s}_i, t_i)) = [\mu(cs_1, cs_2, ..., cs_k, \epsilon; \zeta)]_b$, where $[\boldsymbol{x}]_b$ selects the bth element of $\boldsymbol{x}$, and $b \in [1, k+1]$. μ is the softmax operator with ζ as the temperature parameter, that is:

$$[\mu(cs_1, cs_2, ..., cs_k, \epsilon; \zeta)]_b = \begin{cases} \frac{\exp(\frac{cs_b}{\zeta})}{\sum_{i=1}^{k} \exp(\frac{cs_i}{\zeta}) + \frac{\epsilon}{\zeta}}, & b \in [1, k] \\ \frac{\exp(\frac{\epsilon}{\zeta})}{\sum_{i=1}^{k} \exp(\frac{cs_i}{\zeta}) + \frac{\epsilon}{\zeta}}, & b = k+1 \end{cases}$$

The working principle of $\epsilon-$softmax is as follows: if v_i is not coherent with any of existing sub-sequences, that is, $cs_b < \epsilon, \forall b \in [1, k]$, then the user is more likely to launch a new preference thread. Correspondingly, $\arg\max_{b \in [1,k+1]} \pi(a_i = b|(\boldsymbol{s}_i, t_i)) = k+1$. If there are many sub-sequences, satisfying $cs_b > \epsilon$, then v_i will be allocated to the one with the largest cs_b, which agrees with our expectation, i.e., allocating v_i to the most coherent sub-sequence. As a special case, if $cs_b > \epsilon, \forall b \in [1, k]$, then the agent will not create new sub-sequences, which is similar to deploy an ordinary softmax on the action space [1,k]. With $\epsilon-$softmax, the number of sub-sequences can be adaptively increased, which is able to capture the evolving nature of user preferences.

Reward (r_i): We design rewards based on the following aspects:

• Whether the decomposed sub-sequences can lead to the better fitting of the training data? In general, a model with lower loss means it can fit the data better. Thus, we use the negative loss as the reward to measure the capability of our model on fitting the data. For the sample (H_{T+1}, v_{T+1}), we denote by $l(H_{T+1}, v_{T+1})$ the loss function of predicting v_{T+1} based on H_{T+1}. Since we can only compute the loss when all the sub-sequences have been generated, the reward is defined in a delayed manner, that is:

$$r_{i,1} = \begin{cases} 0\,, & \text{if } i \in [1, T-1] \\ -l(H_{T+1}, v_{T+1})\,, & \text{if } i = T \end{cases} \tag{4}$$

• Whether the allocated items can lead to better coherence within each sub-sequence? To compute the sub-sequence coherence, we firstly compute the embedding of each sub-sequence $\boldsymbol{s}_{i,l}$ as the sum of its item embeddings, that is, $\boldsymbol{\kappa}_l = \frac{1}{|\boldsymbol{s}_{i,l}|} \sum_{k:(k,v) \in \boldsymbol{s}_{i,l}} \boldsymbol{e}_k$. Suppose the action at step i is b, $\boldsymbol{e}_i$ is the embedding of item v_i, then the reward is set as the cosine similarity between $\boldsymbol{s}_{i,b}$ and v_i, that is, $r_{i,2} = \frac{\boldsymbol{\kappa}_b^T \cdot \boldsymbol{e}_i}{||\boldsymbol{\kappa}_b||_2 ||\boldsymbol{e}_i||_2}$, where $|| \cdot ||_2$ is the L_2-norm. If $b = k+1$, we directly set the reward as the similarity between the user and item embeddings.

• Whether the allocated items can lead to better orthogonality between different sub-sequences? Ideally, each sub-sequence should exactly encode one type of user preference, thus we encourage orthogonality between different sub-sequences for avoiding information leaking. Formally, if the action at step i is b, then the reward is set as $r_{i,3} = -\sum_{b \neq j} \frac{||\hat{\boldsymbol{\kappa}}_b^T \cdot \boldsymbol{\kappa}_j||_2}{||\hat{\boldsymbol{\kappa}}_b||_2 ||\boldsymbol{\kappa}_j||_2}$, where $\hat{\boldsymbol{\kappa}}_b = \frac{1}{|\boldsymbol{s}_{i,b}|+1}(\boldsymbol{e}_i + |\boldsymbol{s}_{i,b}|\boldsymbol{\kappa}_b)$ is the embedding of $\boldsymbol{s}_{i,b}$ after incorporating v_i. Since only sub-sequence b has been changed at this step, we only check the orthogonality between $\boldsymbol{s}_{i,b}$ and the other sub-sequences.

• Whether the number of sub-sequences is appropriate? Intuitively, if the number of sub-sequences is too small, then user preferences may not be well separated. However, if there are too many sub-sequences, then each sub-sequence may only contain a few items, which can be hard to represent the user preference. In order to better control the sub-sequence number, we introduce the following reward:

$$r_{i,4} = \begin{cases} 0\,, & \text{if } b \in [1, k] \\ \lambda\,, & \text{if } b = k+1 \end{cases} \tag{5}$$

where if the current action a is "creating a new sub-sequence", then there will be a non-zero reward λ. If $\lambda > 0$, then the agent is encouraged to produce more sub-sequences, otherwise, "creating new sub-sequences" is penalized.

For better guiding the sub-sequence generation, we implement λ in a curriculum learning manner. More specifically, in the beginning of the sequence decomposition, there are only a few sub-sequences. At this time, we do not impose much constraint on "creating new sub-sequences", and set λ as a large value. As more sub-sequences are generated, we would like to control the increasing speed of the sub-sequence number, and thus lower the value of λ. To realize this idea, we set $\lambda = w_1 i + w_2$, where i is the index of the action step. w_1 and w_2 are hyper-parameters, and we set $w_1 < 0$ to ensure that λ is monotonically decreasing w.r.t. the index i, that is, as the agent takes more actions, we impose more strict control on the number of sub-sequences.

By combining $r_{i,1}$ to $r_{i,4}$, the overall reward at step i is $r_i = \sum_{d=1}^{4} r_{i,d}$. A complete running example of the above MDP can be seen in Figure 2(a), where, at each step, the current item is allocated into a preference thread based on the agent π.

Suppose the trajectory of decomposing the item sequence is $\tau = \{\boldsymbol{st}_1, a_1, \boldsymbol{st}_2, a_2, ..., \boldsymbol{st}_T, a_T, \boldsymbol{st}_{T+1}\}$[3], then the probability of observing τ is $p(\tau) = p(\boldsymbol{st}_1) \prod_{i=1}^{T} p(\boldsymbol{st}_{i+1}|\boldsymbol{st}_i, a_i)\pi(a_i|\boldsymbol{st}_i)$, where $p(\boldsymbol{st}_1)$ is the initial state distribution, and $p(\boldsymbol{st}_{i+1}|\boldsymbol{st}_i, a_i)$ is the deterministic transition kernel defined in equation (2). The objective for learning π is $L = E_{\tau \sim p(\tau)}[\sum_{i=1}^{T} \gamma^{i-1} r_i]$, where γ is the predefined discount factor. By considering all the empirical training samples, and according to the log-trick, we can directly optimize the following

[3] Here "$\boldsymbol{st}_{j+1}$" is the end state.

Table 2: Relation between our model and the previous work.

Model	Global	Local
General recommendation	Static	-
Sequential recommendation	Evolving	-
Multi-interest recommendation	Evolving	Static
SPLIT (our model)	Evolving	Evolving

objective:

$$L_{IA}(\Theta_\pi, \Theta_e) = \sum_{u=1}^{|\mathcal{U}|}\sum_{T=1}^{l_u-1}[R_T^u \sum_{i=1}^{T} \log \pi(a_i^u|\boldsymbol{st}_i^u)], \tag{6}$$

where $R_T(\tau) = \sum_{i=1}^{T} \gamma^{i-1} r_i$, and we recover label u to indicate the user of the training samples.

Model Specification. (i) For g_s, we leverage LSTM [14], GRU [10] and Transformer [35] to implement it. Give a sub-sequence $\boldsymbol{s}_{i,b} = [u, (v_{i,b,1}, t_{i,b,1}), ..., (v_{i,b,l_b}, t_{i,b,l_b})]$, the input of g_s at each step j is $(v_{i,b,j}, t_{i,b,j} - t_{i,b,j-1})$, where if $j = 1$, then the input is $(v_{i,b,1}, 0)$. To derive the embedding of the input, we firstly project the time interval $(t_{i,b,j} - t_{i,b,j-1})$ with a linear operator, and then cancat it with the embedding of $v_{i,b,j}$. We initialize the first step of g_s with the user embedding, and the model architectures follow the original papers. At last, we compute the final sub-sequence embedding as the average of the hidden states. (ii) For ϕ, it aims to monotonically project the time interval information into a decaying constant. In our model, we use two methods to implement ϕ, that is:

$$\phi(t_i - t_b) = \begin{cases} -\kappa_1(t_i - t_b) + \kappa_2 \ , & \text{linear method} \\ e^{-\kappa_3(t_i - t_b)} \ , & \text{exponential method} \end{cases} \tag{7}$$

where $\kappa_1 > 0$ and $\kappa_3 > 0$ are pre-defined hyper-parameters.

3.2 Sub-sequence Modeler

The second component of our model is a set of sub-sequence modelers (see Figure 2(b)). For each training sample (H_{T+1}, v_{T+1}), where $H_{T+1} = (\boldsymbol{s}_{T+1}, \boldsymbol{t}_{T+1})$, we firstly decompose $\boldsymbol{s}_{T+1}$ into K sub-sequences $\{\boldsymbol{s}_{T+1,b}\}_{b=1}^{K}$ based on the above item allocator. Then a sequential model g_x is leveraged to project each $\boldsymbol{s}_{T+1,b}$ into an embedding, where we implement g_x with the same architecture as used for g_s. Suppose the output embedding of $g_x(\boldsymbol{s}_{T+1,b})$ is $\boldsymbol{x}_b$, then the learning objective is $l(H_{T+1}, v_{T+1}) = -\sum_{b=1}^{K+1} \pi_b l_{ce}(\boldsymbol{x}_b, \boldsymbol{e}_{T+1})$, where $\pi_b = \pi(a_{T+1} = b|(\boldsymbol{s}_{T+1}, \boldsymbol{t}_{T+1}))$ is the probability of allocating v_{T+1} into an existing sub-sequence (when $b \in [1, K]$) or creating a new one initialized by v_{T+1} (when $b = K+1$). $\boldsymbol{e}_{T+1}$ is the embedding of item v_{T+1}. l_{ce} is the binary cross entropy loss, that is,

$$l_{ce}(\boldsymbol{x}_b, \boldsymbol{e}_{T+1}) = \log \sigma(\boldsymbol{x}_b^T \boldsymbol{e}_{T+1}) + \sum_{k \in S^-(v_{T+1})} \log(1 - \sigma(\boldsymbol{x}_b^T \boldsymbol{e}_k)), \tag{8}$$

where, when $b = K + 1$, we assign $\boldsymbol{x}_b$ with the user embedding.

At last, the overall learning objective is derived by summing the losses of all the training samples, that is:

$$L_{SSM}(\Theta_e, \Theta_x) = \sum_{u=1}^{|\mathcal{U}|}\sum_{T=1}^{l_u-1} l(H_{T+1}^u, v_{T+1}^u), \tag{9}$$

where $H_{T+1}^u = (\boldsymbol{s}_{T+1}^u, \boldsymbol{t}_{T+1}^u)$, and we recover label u to indicate the training samples related with user u. Θ_x is the set of parameters related with g_x.

Algorithm 1: Learning algorithm of our model

```
Indicate the number of training batches M.
Initialize the model parameters $\Theta_\pi, \Theta_e, \Theta_x$.
Initialize a tuning parameter $\alpha$.
for batch number in [0, M] do
    for each sample $(H_{T+1}, v_{T+1})$ in the batch do
        for i in [1, T] do
            Select an action $a_i$ based on $\boldsymbol{st}_i = (\boldsymbol{s}_i, \boldsymbol{t}_i)$ and $\pi$.
            Allocate $v_i$ into sub-sequence $a_i$.
            Obtain the reward $r_i$.
            Update the state to $\boldsymbol{st}_{i+1} = (\boldsymbol{s}_{i+1}, \boldsymbol{t}_{i+1})$.
        end
        Update $\Theta_\pi, \Theta_e$ based on objective (6) and
          $\{(\boldsymbol{s}_i, \boldsymbol{t}_i), a_i, r_i\}_{i=1}^{T}$.
    end
    Update $\Theta_\pi, \Theta_e, \Theta_x$ based on objective (10).
end
```

4 LEARNING PROCESS

The complete learning process of our framework is summarized in Algorithm 1. For each training sample, the agent generates a trajectory as follows: at each step, the agent firstly takes an action based on the state, and then the reward is obtained based on $r_i = \sum_{d=1}^{4} r_{i,d}$, where the loss function in $r_{i,1}$ is derived by fixing g_x obtained from the last training batch. At last, the state $\boldsymbol{st}_i$ is updated to $\boldsymbol{st}_{i+1}$ based on equation (2). After obtaining the trajectory, the agent is optimized based on objective (6). Once the agent has generated trajectories for all the training samples in a batch, we update the parameters of $\Theta_s, \Theta_e, \Theta_x$ jointly based on the following objective:

$$L = \alpha L_{IA}(\Theta_\pi, \Theta_e) + (1 - \alpha) L_{SSM}(\Theta_x, \Theta_e), \tag{10}$$

where α is a parameter balancing different optimization targets. In the serving process, the agent has been learned, and it just need to infer actions to separate user behavior sequences, which is not time consuming.

4.1 Further Discussion

How do we overcome the challenges of CH1 to CH3? Our model is an effective remedy for the three challenges proposed in the introduction. For **CH1**, we develop a novel ϵ-softmax operator, which allows the number of user preference threads to be adaptively determined according to the target sequence. Here, we do not differentiate the sequences between different users or for the same user, our model is applicable for both of these scenarios. For **CH2**, reinforcement learning is naturally designed for modeling the evolution of sequential events. In our model, the agent π simulates how the user preference evolves, and the current item is allocated into a preference thread based on the previously generated sub-sequences. For **CH3**, we incorporate the time information into the agent, where if the current item is far from a sub-sequence, then it is less likely to be allocated into it. In fact, the above challenges are hard to be simultaneously addressed by existing multi-interest recommender models, which are mostly based on the capsule network and dynamic routing. In this paper, we follow a fundamentally

Table 3: Statistics of the datasets. We remove "Amazon-" and "Foursquare-" for saving the space.

Dataset	#User	#Item	#Interaction	Density	Domain
Video	5,131	1,686	37,126	0.43%	e-commence
Garden	1,687	963	13,272	0.82%	e-commence
Music	1,430	901	10,261	0.80%	e-commence
Baby	19,446	7,051	160,792	0.12%	e-commence
Beauty	22,364	12,102	198,502	0.08%	e-commence
Wechat	4.364	10,654	198,702	0.43%	micro-video
NY	1,063	3,896	40,825	0.99%	check-in
TKY	2,288	7,056	128,530	0.80%	check-in

different principle, and solve the above challenges seamlessly in a unified RL framework.

How does our model relate with the previous work? In general recommendation, the user-item interactions are modeled independently in a global manner. We call this method as a "globally static" paradigm. In sequential recommendation, the user evolving preference is considered, and all the items are modeled by a unified sequential model, which can be seen as a "globally evolving" paradigm. In multi-interest recommendation, user evolving preference is modeled in a finer-grained manner, but the items are independently allocated into the sub-sequences. We name this method as "globally evolving and locally static" paradigm. In our model, we further improve multi-interest recommendation by capturing user evolving preference when generating sub-sequences, which is a "globally evolving and locally evolving" paradigm. We summarize the above comparisons in Table 2 to clearly position our work, where we can see our model bridges an evident gap and has its unique contributions.

5 EXPERIMENTS

5.1 Experiment Setup

5.1.1 Datasets. Our experiments are conducted based on eight real-world datasets across three different domains. More specifically, **Amazon-Video**, **Amazon-Garden**, **Amazon-Music**, **Amazon-Baby** and **Amazon-Beauty**[4] are e-commerce datasets collected from Amazon.com, which contain user purchasing records from different product categories. **Wechat**[5] is a video dataset, which provides the watching behaviors of the users on the videos. **Foursquare-NY**[6] and **Foursquare-TKY** [30, 42, 43] are well-known recommendation datasets, containing user check-in information in New York and Tokyo spanning for about 10 months. The statistics of these datasets are summarized in Table 3.

5.1.2 Baselines. In order to demonstrate the effectiveness of our model, we compare it with 11 representative baselines. We introduce these baselines following a similar taxonomy as in Table 2: for general recommendation, **BPR** [26] is a well-known recommender model for capturing user implicit feedback, where we use matrix factorization as the prediction model. **NCF** [15] is a deep recommender model, which generalizes the traditional matrix factorization method for modeling user-item nonlinear relationship. For sequential recommendation, **GRU4Rec** [16] is an early method for modeling user sequential behaviors based on GRU, where the user history behaviors are recurrently feed into the model for predicting the next item. **STAMP** [22] is an attention based sequential recommender model, which believes that the most recent interaction is the key for the current user behavior. **NARM** [19] is also an attention based sequential recommender model, where different user history behaviors are discriminated by an attention neural network. **BERT4Rec** [34] is a sequential recommender model based on the self-attention mechanism. For fair evaluation, we also compare our model with two sequential recommender models, which contain the user-item interaction time information. In specific, **TLSTM** [47] is a novel time-aware LSTM architecture, which is able to incorporate the time information for user behavior modeling. **TASER** [44] is a self-attention based recommender model, where the item embeddings are enhanced by the continuous time information. For multi-interest recommendation, **MCPRN** [37] is a well known multi-interest recommender model, which associates each item with a type of user preference based on the capsule network and dynamic routing. **PIMI** [7] is a time-sensitive multi-interest recommender model, where the user preferences are decomposed by considering the time interval information between successive behaviors. **Octopus** [23] is a recently proposed multi-interest recommender model, which designs a novel "activation" mechanism, and different sequences may activate various number of interests.

5.1.3 Implementation details. For each user behavior sequence, we use the last and second last interactions as the testing and validation sets, respectively, while the others are left for model training. We evaluate different models based on the metrics including Precision [38], Recall [38], NDCG [4] and MRR [29]. In the experiments, five items are recommended from each model to compare with the ground truth. More specifically, the learning rate and user/item embedding size are tuned in the ranges of $[0.01, 0.005, 0.001]$ and $[64, 128, 256]$, respectively. The batch size is determined in the range of $[256, 512, 1024]$. The threshold ϵ is searched in $[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$. The curriculum parameters w_1 and w_2 are determined in the ranges of $[-0.1, -0.2, -0.3, -0.4, -0.5]$ and $[0.1, 0.2, 0.3, 0.4, 0.5]$, respectively. The tuning parameter α is empirically set as 0.5.

For the baselines, we set the parameters as their default values reported in the original papers or tune them in the same ranges as our model's.

5.2 Overall Performance

In this section, we compare our model with the baselines, and the results are presented in Table 4, where we can see: in general, sequential recommender models can achieve better performance than the non-sequential ones. For example, in most cases, GRU4Rec, STAMP and NARM perform better than BPR and NCF. These observations agree with the previous studies [19, 22, 32], and manifest that explicitly modeling the correlations between user behaviors is indeed useful to improve the recommendation performance. Among different sequential recommender models, the non-attentive method GRU4Rec performs worse than the other ones. We speculate that the attention mechanism can effectively discriminate the importances of different items. The information which are useful for the target item prediction is strengthened, while the noisy items

[4] http://jmcauley.ucsd.edu/data/amazon/
[5] https://algo.weixin.qq.com/
[6] https://sites.google.com/site/yangdingqi/home/foursquare-dataset

Table 4: Overall comparison between our model and the baselines. All the numbers are percentage values with "%" omitted. We remove the prefix of Foursquare-NY and Foursquare-TKY for saving the space. The best performance is labeled by bold fonts. "*" indicates the improvement of our model against the best baseline is significant under paired-t test with "$p < 0.05$".

Category		General Model		Sequential Model						Multi-interest Model			Our
Method		BPR	NCF	GRU4Rec	STAMP	NARM	BERT4Rec	TLSTM	TASER	MCPRN	PIMI	Octopus	SPLIT
Amazon-Garden	Precision@5	0.56	0.55	1.36	1.20	1.59	0.94	1.45	1.20	1.60	1.55	1.30	**1.71***
	Recall@5	2.79	2.73	6.82	5.99	7.95	4.69	7.24	5.97	8.01	7.01	6.52	**8.54***
	NDCG@5	1.72	1.69	4.62	4.90	5.21	2.71	5.24	4.12	5.46	4.36	3.50	**5.92***
	MRR@5	1.37	1.35	3.88	4.22	4.31	2.07	4.59	2.84	4.62	3.94	2.52	**5.07***
Amazon-Video	Precision@5	2.58	2.52	2.97	2.97	3.06	1.92	3.11	2.68	2.74	2.95	2.16	**3.43***
	Recall@5	12.88	8.18	14.83	14.84	15.95	9.62	15.54	13.39	13.72	9.81	11.59	**17.13***
	NDCG@5	8.46	5.12	10.37	11.22	11.22	7.87	11.44	8.01	10.07	9.65	8.11	**13.23***
	MRR@5	6.99	6.72	8.90	9.88	9.88	5.85	7.74	6.25	8.87	8.66	6.97	**11.93***
Amazon-Music	Precision@5	1.05	1.06	0.91	1.02	0.91	0.77	1.08	0.88	1.13	1.04	1.07	**1.33***
	Recall@5	5.25	5.32	4.55	5.11	4.55	3.85	5.39	4.41	5.67	5.18	5.23	**6.65***
	NDCG@5	3.34	3.59	3.01	3.57	3.14	2.17	3.62	3.09	3.82	3.43	3.56	**4.22***
	MRR@5	2.72	3.03	2.51	3.06	2.68	1.63	3.04	2.66	3.21	2.87	2.91	**3.42***
Amazon-Baby	Precision@5	0.23	0.20	0.52	0.49	0.50	0.29	0.55	0.52	0.54	0.42	0.29	**0.59***
	Recall@5	1.17	1.02	2.60	2.43	2.52	1.46	2.74	2.73	2.72	2.11	1.47	**3.22***
	NDCG@5	0.75	0.66	1.68	1.66	1.67	0.94	1.80	1.71	1.90	1.31	0.87	**2.02***
	MRR@5	0.61	0.54	1.38	1.41	1.40	0.77	1.49	1.44	**1.73**	1.05	0.67	1.66
Amazon-Beauty	Precision@5	0.48	0.51	0.90	0.93	0.91	0.70	1.05	0.77	1.02	0.79	0.69	**1.27***
	Recall@5	2.42	2.57	4.48	4.64	4.55	3.49	5.23	3.86	5.09	0.79	3.43	**6.35***
	NDCG@5	1.69	1.63	3.02	3.44	3.14	2.32	3.63	2.38	3.69	2.62	1.98	**3.70***
	MRR@5	1.37	1.32	2.59	3.04	2.68	1.94	3.10	1.18	3.24	2.19	1.38	**3.31***
Wechat	Precision@5	0.26	0.24	0.50	0.47	0.57	0.66	0.63	0.66	0.61	0.59	0.36	**0.73***
	Recall@5	1.31	1.21	2.50	2.34	2.87	2.93	3.14	3.12	3.05	2.93	1.81	**3.67***
	NDCG@5	0.77	0.78	1.48	1.47	1.78	1.85	1.91	1.81	1.92	1.85	1.04	**2.25***
	MRR@5	0.60	0.62	1.15	1.19	1.43	1.50	1.52	1.51	1.55	1.50	0.79	**1.79***
NY	Precision@5	1.32	1.28	1.26	1.32	1.34	1.39	1.39	1.43	1.39	1.37	1.28	**1.45***
	Recall@5	6.59	6.40	6.31	6.59	6.69	6.97	6.97	7.02	6.97	6.87	6.40	**7.25***
	NDCG@5	4.32	4.29	4.04	4.11	4.55	4.58	4.58	4.69	4.69	4.54	4.29	**4.93***
	MRR@5	3.59	3.59	3.29	3.29	3.84	3.97	3.79	3.88	3.97	3.77	3.59	**4.16***
TKY	Precision@5	1.44	1.62	2.16	1.97	2.08	2.08	1.92	1.76	1.78	1.78	1.53	**2.21***
	Recall@5	7.36	8.33	9.78	9.84	10.41	10.41	9.58	8.79	8.92	8.88	7.65	**11.06***
	NDCG@5	4.55	5.12	6.90	6.77	7.06	7.00	6.66	6.77	6.10	6.44	5.38	**7.42***
	MRR@5	3.88	4.15	6.12	5.77	5.97	5.88	5.70	6.01	5.18	5.64	4.63	**6.22***

are lower weighted. Thus, attention based models can better represent the history information, and bring superior performances. Among all the baselines, the multi-interest recommender model MCPRN can usually achieve the best performance, which verifies the effectiveness of decomposing and modeling user independent preferences. In most cases, our model can achieve the best performance across different datasets and metrics. In specific, our model can on average improve the best baselines by about 9.68%, 12.4%, 8.56% and 7.13% on the metrics of Precision, Recall, NDCG and MRR, respectively. Comparing with sequential recommender models, we are able to decompose the mixed user preferences, which facilitates more clear history representation and focused item estimation. Comparing with multi-interest recommender models, our method can simultaneously consider the inconsistent, evolving and uneven characters of the user preference, which are important for decomposing user behaviors in practice. We speculate that such designs may successfully reveal the basic rules underlying user behaviors, which introduce beneficial inductive bias, and lead to better recommendation performance.

5.3 Ablation Studies

In the above section, we compare our model with the baselines as a whole. Readers may be interested in how different components in our model contribute the final performance. To answer this question, we compare our model with its seven variants: in **SPLIT (-r_1)**, we remove the reward for better fitting the training data (*i.e.*, equation (4)). In **SPLIT (-r_2)**, the reward for promoting sub-sequence coherence (*i.e.*, $r_{i,2}$) is removed. In **SPLIT (-r_3)**, we do not impose constraints on the orthogonality between different sub-sequences, that is, we remove the reward of $r_{i,3}$. In **SPLIT (-r_4, c)**, we do not use curriculum learning in equation (5), where we fix λ as a hyper-parameter[7]. In **SPLIT (-r_4)**, we completely drop the reward of $r_{i,4}$. In **SPLIT (-ϕ)**, we drop the temporal influence, where we remove $\phi(t_i - t_b)$ from equation (3). In **SPLIT (-ϵ)**, we implement our model with fixed number of user preference threads, where we set the thread number K as a hyper-parameter and tune it to achieve the best performance. In the experiments, the model parameters follow the above settings, and we report the performance based on NDCG and the datasets of Amazon-Garden and Amazon-Video, respectively. The results on the other metrics and datasets are similar and omitted. From the results shown in Figure 3, we can see: the reward

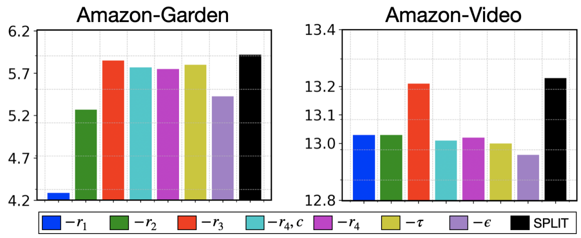


Figure 3: Results of the ablation studies.

[7] In the experiments, we tune λ to report the best performance.

Table 5: Influence of different g_s's and ϕ's.

Amazon-Garden			
g_s	LSTM	GRU	Transformer
ϕ (Linear)	8.88	13.00	11.49
ϕ (Exponential)	6.89	13.32	13.13
Amazon-Video			
g_s	LSTM	GRU	Transformer
ϕ (Linear)	2.78	5.4	5.41
ϕ (Exponential)	3.24	5.92	5.46

for better fitting the training data (*i.e.*, r_1) is very important, which is evidenced by the lowered performance of SPLIT (-r_1) comparing with SPLIT. On the dataset of Amazon-Garden, the performance of SPLIT (-r_1) is even worse than many baselines. Actually, this is not surprising, since this reward is directly related with the loss function, which is critical for the model performance. SPLIT (-r_2) and SPLIT (-r_3) perform worse than SPLIT on both datasets, which suggests that the reward on promoting intra-subsequence coherence (*i.e.*, r_2) and inter-subsequence orthogonality (*i.e.*, r_3) are both necessary. It is very interesting to see that the performances of SPLIT (-r_4, c) and SPLIT (-r_4) are similar. This manifests that if we control the number of sub-sequences in a too simple manner, *e.g.*, defining an unchanged reward, then its effect is similar to that of not setting this reward at all. By carefully designing the reward function, SPLIT is better than both SPLIT (-r_4, c) and SPLIT (-r_4), which demonstrates the effectiveness of our proposed curriculum reward. By comparing SPLIT (-ϕ) with SPLIT, we find that the temporal function ϕ is important, since removing it can lead to lowered performances on both datasets. This result actually verifies the effectiveness of addressing the third challenge mentioned in the introduction, and confirms the importance of modeling the time interval information between successive user behaviors. If we unify the number of sub-sequences for all the samples, the performance is not satisfied. For example, on the dataset of Amazon-Video, SPLIT (-ϵ) is the worst among all the variants. This observation highlights the significance of overcoming the first challenge in the introduction, that is, different user behavior sequences should be separated into various numbers of sub-sequences.

5.4 Further Experiments

In this section, we conduct further experiments to study the influence of different model implementations and the key hyper-parameters. Similar to the above settings, we base the experiments on the datasets of Amazon-Garden and Amazon-Video, and use NDCG as the evaluation metric. The results on the other datasets and metrics are similar and omitted. We set the model parameters as their optimal values tuned in section 5.2.

5.4.1 Influence of different g_s's and ϕ's. In our model, g_s determines how to represent the generated sub-sequences[8], and ϕ indicates how the time interval information influence the coherence score. In this paper, we explore three methods to implement g_s, that is: LSTM, GRU and transformer. For ϕ, we use either linear or exponential methods to project the time interval information into a constant. In the experiment, the hyper-parameters κ_1, κ_2 and κ_3 are tuned to achieve the best performance. We compare different combinations between g_s and ϕ in Table 5, where we can see: for the sequential architecture g_s, GRU can usually lead to the best performance. We speculate that GRU is a much lighter architecture comparing with LSTM and transformer, which can be more appropriate for the sparse recommendation datasets. More redundant parameters may over-fit the training data and lead to inferior performances. From the perspective of ϕ, exponential function can be the better choice for our task in most cases. The reason can be that the exponential method can project any time-interval information into a moderate range (*i.e.*,, [0,1]), while the linear method may result in too large $\phi(t - t_b)$'s, which may overwhelm the semantic similarity in equation (3), and impact the final performance.

5.4.2 Influence of the threshold ϵ. In our agent π, the threshold ϵ determines how rigorous we would like to constrain the generation of a new sub-sequence. With a small ϵ, the agent can not easily produce new sub-sequences, since the condition "$cs_b < \epsilon$" is hard to satisfy. While if ϵ is larger, then the agent has more changes to generate new sub-sequences. In order to see the influence of ϵ on the final performance, we tune it in the range of $[0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$, and the results are presented in Figure 5, where we can see: on Amazon-Garden, the performance fluctuates dramatically when ϵ is small, and after reaching the optimal point, the performance continually goes down as ϵ becomes larger. On Amazon-Video, the performance curve is much smoother, but similarly, it has experienced the "going-up" and "going-down" processes. On both datasets, the best performance is achieved when ϵ is moderate. We speculate that too large ϵ makes the agent produce too many sub-sequences, and there is only a few items in each sub-sequence, which is limited for comprehensively representing the user preference. While if ϵ is too small, the user diverse preferences cannot be well decomposed, and the mixed preferences may confuse the sub-sequence representations and lower the model performance.

5.4.3 Influence of the curriculum parameters w_1 and w_2. In reward $r_{i,4}$ (*i.e.*, equation (5)), the number of user preference threads is controlled in a curriculum learning manner. w_1 and w_2 determine the curves of the reward. In specific, w_2 sets the initial reward at the first step, while $w_1 < 0$ indicates the reward decreasing speed. In this section, we study the influence of w_1 and w_2 on the final performance. We tune w_1 and w_2 in the ranges of $[-0.1, -0.2, -0.3, -0.4, -0.5]$ and $[0.1, 0.2, 0.3, 0.4, 0.5]$, respectively, and the results are shown in Figure 6, from which we can see: on the dataset of Amazon-Garden, smaller w_1 and w_2 can usually lead to better performances, while on Amazon-Video, the best performance is achieved when w_1 and w_2 are larger. We speculate that the number of items in Amazon-Garden is small, thus their characters can be easily covered by a few amount of independent user preference threads. To limit the thread number, smaller w_1 and w_2 can help to set a lower initial reward on "encouraging new sub-sequences", and decrease this reward sharply in the following action steps. However, in Amazon-Video, there are more items, which needs more user preference threads to cover their properties, thus w_1 and w_2 should be set as larger values to generate more sub-sequences.

[8] It should be noted that we implement g_x and g_s with the same architecture, thus we do not discuss different implementations of g_x.

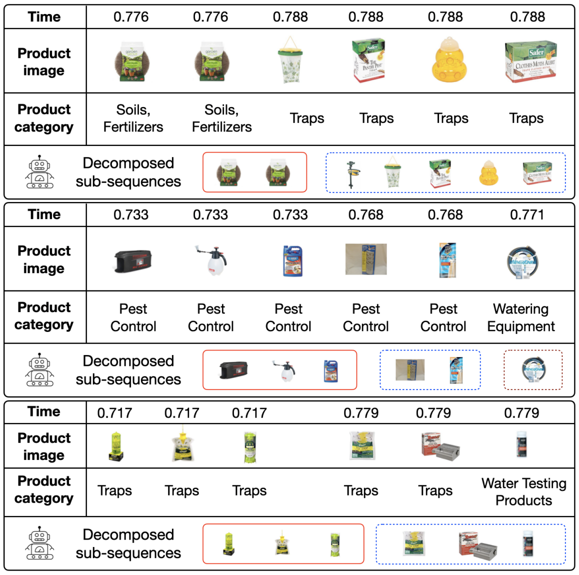


Figure 4: Case studies on the generated sub-sequences, where the time information is normalized into a value between 0 and 1.

5.5 Case Studies

In order to provide more intuitive understandings on the decomposed user behaviors, in this section, we analyze our model by presenting several case studies. We experiment with the dataset of Amazon-Garden, and the model parameters follow their optimal values determined in section 5.2. For each case, we decompose the complete user behavior sequence based on the learned item allocator. From the results presented in Figure 4, we can see: the products with different semantics can be successfully separated into different sub-sequences. For example, in the first case, the items on fertilizers are classified into the first sub-sequence, while the trap products are decomposed into the second one. This observation verifies the basic capability of our model for deriving semantically coherent sub-sequences, which may better serve the following recommendation task. For different sequences, the item allocator can adaptively separate it into different numbers of sub-sequences. For example, in the first and second cases, there are two and three sub-sequences, respectively. Similar observation can also the found between the second and third cases. This result demonstrates the effectiveness of our model in capturing user inconsistent behaviors in the sequence. The temporal information can indeed influence the sequence decomposition. For example, in the third case, while the first five products are all traps, the time interval between the third and forth ones are very large, thus they are separated into different sub-sequences. Similar result can also be found in the second case. These results intuitively illustrate how our model separate user similar preferences happened with a large time interval.

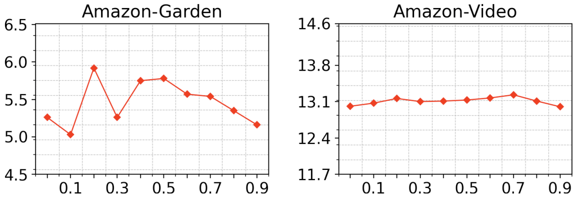


Figure 5: Influence of the threshold ϵ on the model performance in terms of NDCG@5.

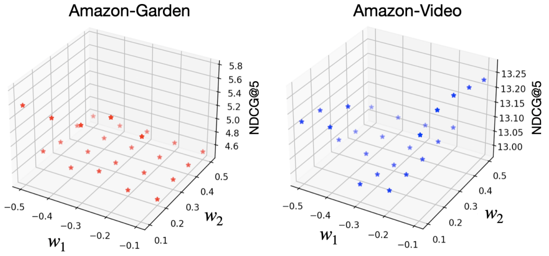


Figure 6: Influence of the curriculum parameters w_1 and w_2 on the model performance in terms of NDCG@5.

6 RELATED WORK

Sequential Recommendation. Our paper is targeted at the problem of sequential recommendation [36]. Early methods [27] in this field regard user behaviors as Markov chains, and the influence from the history behaviors is assumed to concentrate on the latest actions. To model the influence of longer user history, people have proposed a number of models based on recurrent neural network [16, 24], convolutional neural network [34, 45], memory network [9] and transformer [17, 32]. These models also have their respective advantages. For example, many of them [17, 32] can adaptively learn the importances of the previous items. Some of them [9] can store information for extremely long sequence modeling. The major difference between our model and these methods is that we decompose user mixed preferences in the behavior sequence, which facilitates more clear and focused prediction. **Multi-interest Recommendation**. Our model is also related with multi-interest recommendation (MIR). In the past few years, there are a lot of promising multi-interest recommender models. For example, [40] extracts user independent preferences based on the self-attentive mechanism, where different attention heads are regarded as the representations of various user interests. [5, 8, 37] leverage capsule network and dynamic routing to associate each item with the candidate user preferences. [7] incorporates temporal information to derive the item embeddings for multi-interest decomposition. While these models have achieved many promising results, as mentioned before, most of them fully or partially ignore the three key challenges proposed in the introduction, which is important for decomposing real-world user behaviors.

7 CONCLUSIONS

In this paper, we highlight the significance of building sequential recommender models via decomposing user evolving preference. We propose three key challenges one needs to face for achieving this goal, and design an RL based model to seamlessly overcome these challenges. Extensive empirical studies manifest that our model can outperform a series of state-of-the-art methods, and such superiority is consistent across different recommendation domains.

We believe this work makes a novel step towards capturing user finer-grained evolving preference, and there is much room left for improvement. To begin with, we may incorporate more comprehensive side information to provide more valuable signals for allocating the current item into a sub-sequence. In addition, one may also extend our idea to the graph setting, where the user preferences can be decomposed into various sub-graphs.

REFERENCES

[1] Deger Ayata, Yusuf Yaslan, and Mustafa E Kamasak. 2018. Emotion based music recommendation system using wearable physiological sensors. *IEEE transactions on consumer electronics* 64, 2 (2018), 196–203.

[2] Suman Bhoi, Lee Mong Li, and Wynne Hsu. 2020. Premier: Personalized recommendation for medical prescriptions from electronic records. *arXiv preprint arXiv:2008.13569* (2020).

[3] Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational linguistics* 21, 4 (1995), 543–565.

[4] Michael Buckland and Fredric Gey. 1994. The relationship between recall and precision. *Journal of the American society for information science* 45, 1 (1994), 12–19.

[5] Yukuo Cen, Jianwei Zhang, Xu Zou, Chang Zhou, Hongxia Yang, and Jie Tang. 2020. Controllable multi-interest framework for recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2942–2951.

[6] Chong Chen, Min Zhang, Yongfeng Zhang, Yiqun Liu, and Shaoping Ma. 2020. Efficient neural matrix factorization without sampling for recommendation. *ACM Transactions on Information Systems (TOIS)* 38, 2 (2020), 1–28.

[7] Gaode Chen, Xinghua Zhang, Yanyan Zhao, Cong Xue, and Ji Xiang. 2021. Exploring Periodicity and Interactivity in Multi-Interest Framework for Sequential Recommendation. *arXiv preprint arXiv:2106.04415* (2021).

[8] Wanyu Chen, Pengjie Ren, Fei Cai, Fei Sun, and Maarten De Rijke. 2021. Multi-interest Diversification for End-to-end Sequential Recommendation. *ACM Transactions on Information Systems (TOIS)* 40, 1 (2021), 1–30.

[9] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 108–116.

[10] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. [n. d.]. On the properties of neural machine translation: Encoder-decoder approaches. ([n. d.]).

[11] Domitilla Del Vecchio, Richard M Murray, and Pietro Perona. 2003. Decomposition of human motion into dynamics-based primitives with application to drawing tasks. *Automatica* 39, 12 (2003), 2085–2098.

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

[13] Fan Gong, Meng Wang, Haofen Wang, Sen Wang, and Mengyue Liu. 2021. Smr: Medical knowledge graph embedding for safe medicine recommendation. *Big Data Research* 23 (2021), 100174.

[14] Alex Graves. [n. d.]. Generating sequences with recurrent neural networks. ([n. d.]).

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[16] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939* (2015).

[17] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[18] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-interest network with dynamic routing for recommendation at Tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2615–2623.

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

[20] Jianxun Lian, Iyad Batal, Zheng Liu, Akshay Soni, Eun Yong Kang, Yajun Wang, and Xing Xie. 2021. Multi-Interest-Aware User Modeling for Large-Scale Sequential Recommendations. *arXiv preprint arXiv:2102.09211* (2021).

[21] Jinjiao Lin, Haitao Pu, Yibin Li, and Jian Lian. 2018. Intelligent recommendation system for course selection in smart education. *Procedia Computer Science* 129 (2018), 449–453.

[22] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: short-term attention/memory priority model for session-based recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1831–1839.

[23] Zheng Liu, Jianxun Lian, Junhan Yang, Defu Lian, and Xing Xie. 2020. Octopus: Comprehensive and elastic user representation for the generation of recommendation candidates. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 289–298.

[24] Chen Ma, Peng Kang, and Xue Liu. 2019. Hierarchical gating networks for sequential recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 825–833.

[25] SRS Reddy, Sravani Nalluri, Subramanyam Kunisetti, S Ashok, and B Venkatesh. 2019. Content-based movie recommendation system using genre correlation. In *Smart Intelligent Computing and Applications*. Springer, 391–397.

[26] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* (2012).

[27] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. 811–820.

[28] Tomohiro Saito and Yutaka Watanobe. 2020. Learning path recommendation system for programming education based on neural networks. *International Journal of Distance Education Technologies (IJDET)* 18, 1 (2020), 36–64.

[29] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. 2012. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*. 139–146.

[30] Max Sklar, Blake Shaw, and Andrew Hogue. 2012. Recommending interesting events in real-time with foursquare check-ins. In *Proceedings of the sixth ACM conference on Recommender systems*. 311–312.

[31] V Subramaniyaswamy, Gunasekaran Manogaran, R Logesh, V Vijayakumar, Naveen Chilamkurti, D Malathi, and N Senthilselvan. 2019. An ontology-driven personalized food recommendation in IoT-based healthcare system. *The Journal of Supercomputing* 75, 6 (2019), 3184–3216.

[32] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1441–1450.

[33] Qiaoyu Tan, Jianwei Zhang, Jiangchao Yao, Ninghao Liu, Jingren Zhou, Hongxia Yang, and Xia Hu. 2021. Sparse-interest network for sequential recommendation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 598–606.

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

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. [n. d.]. Attention is all you need. ([n. d.]).

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

[37] Shoujin Wang, Liang Hu, Yan Wang, Quan Z Sheng, Mehmet Orgun, and Longbing Cao. 2019. Modeling multi-purpose sessions for next-item recommendations via mixture-channel purpose routing networks. In *International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence.

[38] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. 2013. A theoretical analysis of NDCG ranking measures. In *Proceedings of the 26th annual conference on learning theory (COLT 2013)*, Vol. 8. Citeseer, 6.

[39] Yongji Wu, Lu Yin, Defu Lian, Mingyang Yin, Neil Zhenqiang Gong, Jingren Zhou, and Hongxia Yang. 2021. Rethinking Lifelong Sequential Recommendation with Incremental Multi-Interest Attention. *arXiv preprint arXiv:2105.14060* (2021).

[40] Zhibo Xiao, Luwei Yang, Wen Jiang, Yi Wei, Yi Hu, and Hao Wang. 2020. Deep Multi-Interest Network for Click-through Rate Prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2265–2268.

[41] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems.. In *IJCAI*, Vol. 17. Melbourne, Australia, 3203–3209.

[42] Dingqi Yang, Daqing Zhang, Zhiyong Yu, and Zhu Wang. 2013. A sentiment-enhanced personalized location recommendation system. In *Proceedings of the 24th ACM conference on hypertext and social media*. 119–128.

[43] Mao Ye, Peifeng Yin, and Wang-Chien Lee. 2010. Location recommendation for location-based social networks. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*. 458–461.

[44] Wenwen Ye, Shuaiqiang Wang, Xu Chen, Xuepeng Wang, Zheng Qin, and Dawei Yin. 2020. Time matters: Sequential recommendation with complex temporal information. In *Proceedings of the 43rd International ACM SIGIR Conference on*

Research and Development in Information Retrieval. 1459–1468.

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

[46] Xiaokang Zhou, Yue Li, and Wei Liang. 2020. CNN-RNN based intelligent recommendation for online medical pre-diagnosis support. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* (2020).

[47] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. [n. d.]. What to Do Next: Modeling User Behaviors by Time-LSTM. ([n. d.]).