Rethinking Lifelong Sequential Recommendation with Incremental Multi-Interest Attention

Yongji Wu1∗, Lu Yin3, Defu Lian2, Mingyang Yin3, Neil Zhenqiang Gong1, Jingren Zhou3, Hongxia Yang3
1Duke University, 2University of Science and Technology of China, 3Alibaba Group
{yongji.wu769, neil.gong}@duke.edu, liandefu@ustc.edu.cn, theodore.yl, hengyang.ymy, jingren.zhou, yang.yhx}@alibaba-inc.com

ABSTRACT
Sequential recommendation plays an increasingly important role in many e-commerce services such as display advertisement and online shopping. With the rapid development of these services in the last two decades, users have accumulated a massive amount of behavior data. Richer sequential behavior data has been proven to be of great value for sequential recommendation. However, traditional sequential models fail to handle users’ lifelong sequences, as their linear computational and storage cost prohibits them from performing online inference. Recently, lifelong sequential modeling methods that borrow the idea of memory networks from NLP are proposed to address this issue. However, the RNN-based memory networks built upon intrinsically suffer from the inability to capture long-term dependencies and may instead be overwhelmed by the noise on extremely long behavior sequences. In addition, as the user’s behavior sequence gets longer, more interests would be demonstrated in it. It is therefore crucial to model and capture the diverse interests of users. In order to tackle these issues, we propose a novel lifelong incremental multi-interest self-attention based sequential recommendation model, namely LimaRec. Our proposed method benefits from the carefully designed self-attention to identify relevant information from users’ behavior sequences with different interests. It is still able to incrementally update users’ representations for online inference, similarly to memory network based approaches. We extensively evaluate our method on four real-world datasets and demonstrate its superior performances compared to the state-of-the-art baselines.

CCS CONCEPTS
- Information systems → Recommender systems; Personalization.

KEYWORDS
sequential recommendation, lifelong sequential modeling, incremental attention

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Woodstock ’18, June 03–05, 2018, Woodstock, NY
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06 . . . $15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION
With the rapid development of e-commerce businesses such as online shopping, display advertisement, and video streaming services, a huge amount of behavior data has been produced by users. This provides us with great opportunities to utilize the increasingly richer sequential behavior data, and learn better user representations for a variety of tasks such as CTR prediction and recommendation [18, 27, 39]. The value of rich behavior history (long behavior sequences) has been demonstrated in [27] for user modeling. We focus on the sequential recommendation task [3, 16, 18], which plays a vital role in various online services, including the aforementioned ones.

However, modeling long sequences in sequential recommendation is not a trivial task, and it poses a series of great challenges for us. The traditional sequential recommendation models need to store the whole sequence and perform inference over it [7, 34, 39, 39]. These methods suffer from immense computational and storage cost as it scales linearly with the sequence length. The strict storage and latency constraints have therefore hampered us from employing long sequences for real-time inference in online systems [25]. Furthermore, as the user behavior sequence gets longer, it is more likely to contain items belonging to different categories, revealing the multi-facet nature of users’ interests [25, 27]. The diverse interests of users also contribute to the complex temporal dynamics in long behavior sequences, where various temporal dependency patterns may exhibit [27]. If we fail to identify the different interests manifested in the user’s behavior history, we would instead be overwhelmed by noise brought by irrelevant behaviors using long sequences.

Recently, a few pioneering studies have been conducted to address this issue under the click-through rate (CTR) prediction task. [27] first formulates the problem of lifelong sequential modeling. It borrows the idea of memory networks from the NLP domain, where a personalized behavior memory is maintained for each user to memorize his/her behavior sequences. The RNN-based hierarchical memory serves as the user representation and can be incrementally updated in constant time when the user clicks a new item. Pi et al. [24] propose another memory-based framework which adopts a
different memory update strategy from NTM [10]. Both of the two methods rely on the same set of RNN parameters to update the memory for each user. Therefore, they fail to adapt to each individual user’s personalized behavioral transition patterns, as different users’ behaviors have different temporal dynamics. Besides, RNN-based models pass sequential information through hidden states, they tend to forget about the past [2]. Though architectures like LSTM [2], GRU [5] and NTM [10] have been proposed to alleviate this problem, they still suffer from the \( O(l) \) path length between positions and fail to capture long-term dependencies if the sequence is too lengthy [16, 32].

Search-based methods are introduced in [25, 26] for CTR prediction. They consider that the whole behavior history contains much noise for the sequential models to directly make use of. Instead, they propose search strategies (a hand-crafted one in [25] and a model-based one in [26]) to retrieve only the relevant sub-sequence for the recommender to model. However, these methods utilize the target item to perform the search. Therefore, they cannot be applied to the sequential recommendation task, or the matching stage of recommendation systems, where target items are not available as model inputs.

The self-attention presented in Transformers [32] has seen incredibly successful in many sequential modeling tasks across a variety of fields, such as machine translation [9, 9, 35] and computer vision [1, 23, 33]. The self-attention mechanism uses the sequence to attend to itself. It adaptively assigns different weights to different positions of the sequence according to their relative importance, and then aggregates the sequence using the computed weights. Through this way, we can filter out the irrelevant behaviors in the user’s historical sequence [16]. Compared to RNNs, the self-attention structure has a maximal path length of \( O(1) \). This consequently facilitates self-attention in learning long-term dependencies. [32] Self-attention networks have also demonstrated superior performances in sequential recommendation [16, 20, 30, 37], which greatly outperform the RNN/CNN-based approaches.

Despite self-attention’s triumph in sequential recommendation, and its intrinsic advantages in identifying relevant information from long sequences and capture long-term dependencies, we find that it is quite difficult to apply it to users’ lifelong sequences in the online inference setting, since it bears the same aforementioned problem of linear computational and storage cost. Therefore it is not sufficient for online inference as we require the cost to be constant [24, 27]. In the vanilla self-attention, when a new click action is generated by the user, we have to recompute the user representation using the entire sequence appended with the new action. Recently, linear self-attention methods have been introduced [6, 17], where (linear) dot products of kernel feature maps are used as attention weights. We find that in our sequential recommendation scenario, the linear attention mechanism empowers us to perform incremental attention over the user’s sequence. Through it, we can incrementally update the result of the self-attention operation as new click actions coming from users in constant time. This frees us from the need to store the user’s whole historical sequence and enables us to perform real-time online inference, while still enjoying the benefits brought by the vanilla self-attention mechanism.

We observe that the attention mechanism can be considered as a soft-search operation where more relevant positions to a given query are assigned larger weight, hence they play more dominant roles in weighted aggregation. The attention mechanism therefore, naturally empowers us to capture users’ multi-facet interests from their behavior sequences. As different behaviors of users are triggered by different underlying interests, we could utilize the attention mechanism to soft-search over the user’s lifelong sequence and extract sub-sequences related to different interests of the user. Once again, to address the computational and storage inefficiency of the vanilla attention in online inference, we design a novel multi-interest extraction module under our incremental attention framework. Combined with the incremental self-attention blocks introduced in the previous paragraph, we present our novel lifelong incremental multi-interest self-attention based sequential recommendation model, namely LimaRec.

We summarize our contributions as follows:

- We propose a novel incremental self-attention based method for lifelong sequential recommendation, which goes beyond the limitations of RNN-based memory networks, while also possesses their ability to incrementally update the user representation for online inference.
- Under the same incremental attention framework, we further propose a novel multi-interest extraction module to soft-search the whole sequence for the sub-sequences related to various interests of users.
- We conduct extensive experiments on four real-world datasets and obtain superior performance than state-of-the-art baselines. We also empirically investigate how does the performance of different methods vary on user sequences with increasing length.

## 2 RELATED WORK

### 2.1 Sequential Recommendation

Early works on sequential recommendation usually utilize Markov chains (MCs) to capture the sequential patterns from users’ sequential behaviors. FPMC [28] fuses MCs and matrix factorization to capture both long-term preferences and short-term transitions. Higher-order MCs are used to consider more previous items [12, 13]. With the advent of deep learning, deep sequential models like recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have been extensively investigated in recent years and achieved great success [4, 14, 15, 19, 22, 31, 36].

Recently, self-attention networks [32] have demonstrated superior performance in sequential recommendation due to full parallelism and the capacity to capture long-range dependencies. [16] uses a binary cross-entropy loss based on inner product scores to train a self-attention network, while Zhang et al. [37] propose to optimize a triplet margin loss based on Euclidean distance preference. The Cloze objective, which is proposed for training BERT [9], is used to improve the performance sequential recommendation in [30]. However, these methods have to model the whole behavior sequence to perform inference and have a computational cost linear to the sequence length. Thus they can only be used to handle recent user behaviors in online inference.

### 2.2 Lifelong Sequential Modeling

The problem of lifelong sequential modeling is first introduced in [27] for CTR prediction, where the authors propose a hierarchical
periodic memory network (HPMN) to memorize users’ behavior sequences. HPMN maintains hierarchical memories for each user to retain the information contained in his/her lifelong sequence. The memories are updated with different periods to capture the multi-scale sequential patterns of users. [24] is another method utilizing memory networks. Different from [27], which employs GRU to update the memory, [24] adopts the neural turning machine from [10]. They further propose a memory utilization regularization strategy to reduce the variance of update weights across different memory slots, as well as a memory induction unit to capture high-order information. Compared to self-attention networks, RNNs have limited ability to capture long-term dependencies [32]. This restrains the performance of the methods which rely on RNN-based memory networks.

On the other hand, [25] and [26] propose to address the issue from a different perspective. They argue that since it is computationally expensive to model the whole sequence, and there is much noise in the user’s lifelong sequence, we should search for a limited number of the most relevant and appropriate user behaviors. The retrieved behaviors are then fed into a network to generate the final prediction. [25] uses the category ID of items to find most relevant behaviors, while [26] learns a search strategy with reinforcement learning. However, both of the two methods are designed for CTR prediction, they rely on the information of the target item to perform the search. Since in sequential recommendation, the target item is not available in advance, these methods fail to adapt to our setting.

### 2.3 Modeling Users’ Diverse Interests

The limit of using a single fixed-length representation to express users’ multi-facet interests has been pointed out in [39], where they introduced a local activation unit to adaptively learn the representations of user interests from behavior history, with respect to a given ad. DIEN [38] proposed to capture the interest evolving process of users with an interest extractor layer and an interest evolving layer, which use GRU as building blocks. A behavior refine layer is introduced in [34] to capture better user historical item representations for multi-interest extraction. The aforementioned methods are all devised for the CTR prediction task.

Distinguished from the multi-interest models for CTR prediction, [18] proposes to model users’ diverse interests in the sequential recommendation setting. They design a multi-interest extraction layer based on dynamic capsule routing from [29]. A different routing logic is used in [3] to extract interests, while another extraction method based on attention mechanism is also introduced. However, none of these methods is applicable in lifelong sequential recommendation, as they have a computational cost that scales linearly with the sequence length, while online inference requires $O(1)$ costs.

### 3 METHODOLOGY

#### 3.1 Preliminaries

**3.1.1 Problem Formulation.** For each user $u$, we have an ordered historical sequence of items $[u_1^{(u)}, u_2^{(u)}, \ldots, u_{L(u)}^{(u)}]$ that $u$ has already interacted with (e.g., clicked), where $L^{(u)}$ is the current length of $u$’s historical sequence and $v_l^{(u)}$ is the index of the $l$-th item clicked by $u$. For simplicity, we omit the superscript $(u)$ from now.

In the lifelong sequential recommendation setting, we consider that the user’s sequence is dynamically updated according to the user’s latest actions in the online serving scenario. The user $u$ may have clicked a new item $u_{L+1}$, which is appended to his/her historical sequence $v$. We then use this sequence to predict the next item (or the next set of items) $u$ might click. We consider the matching (candidate generation) stage of industrial recommendation systems [3, 7, 18], where we compute a preference score of user $u$ for each item $v$ based on the inner product similarity between the latent representation of $u$’s updated historical sequence (which includes $u_{L+1}$) and the embedding vector of $v$. The preference scores are then sorted to retrieve the top-$k$ items as the candidate set for $u$. When the user $u$ generates more actions $u_{L+2}, u_{L+3}, \ldots$, they are appended to the user’s historical sequence, and the same recommendation procedure is repeated for online inference with respect to the user’s real-time evolving behaviors.

Since we are handling with lifelong user sequences which might contain thousands of or even more behaviors, it would be infeasible to directly model the whole expanding user sequence in real-time online inference, due to the tremendous computational and storage costs. In addition, with the increasing sequence length, it becomes more and more difficult to precisely capture the complex multi-interests of users and the intricate temporal dynamics of users’ evolving behavior patterns, without overwhelmed by massive noise. Therefore, lifelong sequential recommendation is a non-trivial task facing these challenges. To tackle these issues, we require the recommender to have the following properties under the lifelong sequential recommendation setting:

| Table 1: Important notations. |
|------------------------------|
| Notation | Description |
| $v_l$ | The $l$-th item clicked in the user’s historical sequence. |
| $e_l$ | The embedding vector of the $l$-th item of the sequence. |
| $p_l$ | The positional embedding of the $l$-th position. |
| $H(i)$ | The stacked latent representation of the sequence at each position after the $i$-th self-attention block. |
| $h_l^{(i)}$ | The latent representation up to the $l$-th position (i.e., the $l$-th row of $H(i)$). |
| $S(i)$ | The stacked output of the $i$-th self-attention sub-layer (just before FFN). |
| $s_l^{(i)}$ | The $l$-th row of $S$. |
| $W_Q(i), W_k(i), W_V(i)$ | The key/query/value projection matrices of the $i$-th self-attention sub-layer. |
| $r_l^{(i)}, c_l^{(i)}$ | The maintained hidden states for incremental attention of the $i$-th self-attention block at $l$-th step. |
| $W_K(i), W_V$ | The key/value projection matrices of the multi-interest extraction module. |
| $\tilde{r}_l, \tilde{c}_l$ | The maintained hidden states for the multi-interest extraction module at the $l$-th step. |
| $\mu_k$ | The trainable vector to encode the incentive behind the $k$-th interest. |
| $\varphi_l^{(k)}$ | The disentangled representation of the sequence under the $k$-th interest at the $l$-th step. |
The recommender should maintain a latent representation for each user’s history in a fixed size (constant with respect to the sequence length), instead of storing the whole historical sequence.

The maintained representation should efficiently preserve the behavior information contained in the user’s lifelong sequence. It should also capture the intrinsic multiple interests of the user and the involved temporal dynamics of the user’s behaviors.

The recommender should be able to continuously (incrementally) update the user representation to adapt to the new click behaviors generated by the user under this online setting.

Note that our task of lifelong sequential recommendation is different from that in [24, 27], where they focus on the CTR prediction task, in which the target item is used as model input. However, the target item is not required in our setting.

### 3.1.2 Sequential Recommendation with Self-Attention

Kang and McAuley [16] introduce the SASRec model for sequential recommendation, which borrows the powerful self-attention mechanism from [32]. Self-attention allows SASRec to identify the relevant behaviors in a user’s history for next item prediction, through capturing the intrinsic multiple interests of the user from their lifelong sequence. The recommender should maintain a latent representation for each user’s history in a fixed size (constant with respect to the sequence length), where $h_{l}^{(i)}$ is the embedding vector of the $i$-th item and a trainable positional embedding $p_{l}$. We then stack the embedding vectors $h_{l}^{(i)}$ into a matrix $H^{(0)} \in \mathbb{R}^{L \times d}$. Self-attention is performed on $H^{(0)}$, using each element of the sequence as query to attend to the sequence itself. Concretely, we compute the following output:

$$ S^{(1)} = \text{Attn}(H^{(0)}W^{(1)}_{Q}, H^{(0)}W^{(1)}_{K}, H^{(0)}W^{(1)}_{V}) $$

$$ = \text{softmax} \left( \frac{H^{(0)}W^{(1)}_{Q} (H^{(0)}W^{(1)}_{K})^{\top}}{\sqrt{d}} \right) H^{(0)}W^{(1)}_{V}^{\top} $$

where $W^{(1)}_{Q}, W^{(1)}_{K}, W^{(1)}_{V} \in \mathbb{R}^{d \times d}$ are the trainable projection matrices.

After the self-attention sub-layer, the output $S^{(1)}$ is fed to a position-wise Feed-Forward Network (FFN) sub-layer, which is applied to each position in the sequence independently:

$$ h_{l}^{(1)} = \text{ReLU}(W^{(1)}_{FFN,1}s^{(1)}_{l} + b^{(1)}_{FFN,1})W^{(1)}_{FFN,2} + b^{(1)}_{FFN,2} $$

where $W^{(1)}_{FFN,1}, b^{(1)}_{FFN,1}, W^{(1)}_{FFN,2}, b^{(1)}_{FFN,2} \in \mathbb{R}^{d}$ are the parameters of the FFN sub-layer.

Subsequently, a second self-attention block is applied using $H^{(1)}$ as input. A self-attention sub-layer performs the same computation as Eq. 2 on $H^{(1)}$, but with a different set of projection weights $W^{(2)}_{Q}, W^{(2)}_{K}, W^{(2)}_{V}$. The result $S^{(2)}$ then goes through another FFN sub-layer to produce the final output $H^{(2)}$, where $h_{l}^{(2)}$ is the latent representation of the user’s historical sequence up to the $l$-th position. Residual connection and layer normalization are also used in SASRec to stabilize the network training.

We use $h_{l}^{(2)}$ as the up-to-date user representation (since it encodes the entire historical sequence $[v_{1}, \cdots, v_{L}]$), and compute its inner product with $e_{j}$ as the preference score for any item $j$.

### 3.2 Modeling the Lifelong Sequence with Incremental Attention

Through using the self-attention mechanism to both capture the long-term dependencies in the user’s historical sequence, and focus only on the informative (relevant) part to reduce noise, SASRec outperforms state-of-the-art MC/CNN/RNN-based methods. However, a fatal problem arises naturally when we try to apply SASRec to lifelong sequences. Assume the user has clicked a new item $v_{L+1}$, then we have to update the user representation by computing the latent representation of the sequence appended with $v_{L+1}$. For the first self-attention block, we need to use $h_{L+1}^{(0)}$ as query to attend to every position of the sequence:

$$ s_{L+1}^{(1)} = \text{softmax} \left( \frac{(W^{(1)}_{Q}Q_{V}h_{L+1}^{(0)})^{\top} (H^{(0)}W^{(1)}_{K})^{\top}}{\sqrt{d}} \right) H^{(0)}W^{(1)}_{V}^{\top} $$

$$ = \frac{\sum_{l=1}^{L} \exp \left( (W^{(1)}_{Q}Q_{V}h_{L+1}^{(0)})^{\top} (W^{(1)}_{K}h_{l}^{(0)}) \right) (W^{(1)}_{V}h_{l}^{(0)})}{\sum_{l=1}^{L} \exp \left( (W^{(1)}_{Q}Q_{V}h_{L+1}^{(0)})^{\top} (W^{(1)}_{K}h_{l}^{(0)}) \right) } $$

Since we use softmax to normalize the scores, we have to compute the inner products (after projection) between $h_{l}^{(0)}$ and $h_{l}^{(0)}$ for $i = 1, \ldots, L+1$, take the exponents, and then normalize them so that they are all positive and sum up to 1. Therefore, we must store the whole sequence $[h_{0}^{(0)}, \cdots, h_{L+1}^{(0)}]$ in order to compute these inner products and perform the weighted average over this sequence. The same applies to the second self-attention block. This leads to a computational and storage cost of $O(L)$, which is prohibitive in the online setting of lifelong sequential recommendation. Note that when the new action $v_{L+1}$ arrives, we do not have to recompute $h_{i}^{(1)}, \cdots, h_{L+1}^{(1)}$ (where $i = 1, 2$), since we prevent previous positions from attending to subsequent positions.

Recently, linear self-attention methods were introduced in to reduce the computational complexity of the self-attention operation from quadratic to linear. In these methods, the exponent similarity score (softmax) is replaced with kernels, the inner product of kernel features and perform the weighted average over this sequence. The same applies to the second self-attention block. This leads to a computational and storage cost of $O(L)$, which is prohibitive in the online setting of lifelong sequential recommendation. Note that when the new action $v_{L+1}$ arrives, we do not have to recompute $h_{i}^{(1)}, \cdots, h_{L+1}^{(1)}$ (where $i = 1, 2$), since we prevent previous positions from attending to subsequent positions.

According to the associativity of matrix multiplication, we can write it in the following form:

$$ V_{i} = \frac{\sum_{j=1}^{N} \varphi(Q_{i})^{\top} \varphi(K_{j}) V_{j}}{\sum_{j=1}^{N} \varphi(Q_{i})^{\top} \varphi(K_{j})} $$

$$ V_{i} = \frac{\sum_{j=1}^{N} \varphi(Q_{i})^{\top} \varphi(K_{j}) W_{i}^{(1)}}{\sum_{j=1}^{N} \varphi(Q_{i})^{\top} \varphi(K_{j})} $$

Thus, we just need to compute the summations in the numerator and denominator once, and they can be repeatedly used for all queries.
Choromanski et al. [6] proposes the random feature map, \( \varphi(x) = \frac{1}{\sqrt{m}} \exp(-\frac{\|x\|^2}{2})(\exp(\omega_1^T x), \ldots, \exp(\omega_m^T x)) \), to serve as an unbiased and low-variance approximation of the softmax self-attention. Here the vectors \( \omega_1, \ldots, \omega_m \in \mathbb{R}^d \) are drawn i.i.d. from \( \mathcal{N}(0, L_j) \).

In our setting, we restrict the i-th position to attend only to all the positions \( l' \) that \( l' \leq l \). Therefore, \( s_l^{(i)} \) can be computed as:

\[
s_l^{(i)} = \frac{\varphi \left( W_Q h_l^{(i-1)} \right)^\top \sum_{l'=1}^l \varphi \left( W_K h_{l'}^{(i-1)} \right) (W_V h_{l'}^{(i-1)})^\top}{\varphi \left( W_K h_{l'}^{(i-1)} \right) \sum_{l'=1}^l \varphi \left( W_K h_{l'}^{(i-1)} \right)} \tag{6}
\]

We find that this restriction also empowers us the ability to perform attention over the whole sequence in an incremental manner. We only need to maintain two hidden states. When the sequence is updated with a new action \( v_{l+1} \), we can compute the result of using \( v_{l+1} \) to attend to the historical sequence from \( v_{l+1} \) and the hidden states alone, without the need to store the entire sequence. Concretely, we maintain and incrementally update two hidden states \( r_l^{(i)} \) and \( z_l^{(i)} \) for the i-th self-attention block (\( i = 1, 2 \)) according to the following rules:

\[
r_l^{(i)} = \begin{cases} 0, & l = 0 \\ r_{l-1}^{(i)} + \varphi \left( W_K h_{l-1}^{(i-1)} \right) (W_V h_{l-1}^{(i-1)})^\top, & l > 0 \end{cases} \tag{7}
\]

\[
z_l^{(i)} = \begin{cases} 0, & l = 0 \\ z_{l-1}^{(i)} + \varphi \left( W_K h_{l-1}^{(i-1)} \right), & l > 0 \end{cases} \tag{8}
\]

Here \( r_l^{(i)}, z_l^{(i)} \) are the states of \( r_l^{(i)}, z_l^{(i)} \) at the i-th step (which considers the historical sequence up to the i-th action \( v_l \)). In this way, the output of the attention at the i-th position, which uses \( v_l \) to query the sequence), can be computed as follows:

\[
s_l^{(i)} = \frac{\varphi \left( W_Q h_l^{(i-1)} \right)^\top r_l^{(i)}}{\varphi \left( W_K h_{l}^{(i-1)} \right)^\top z_l^{(i)}} \tag{9}
\]

As we can see, the computation of \( s_l^{(i)} \) depends only on the current output from previous layer \( h_l^{(i-1)} \) (note that \( h_l^{(0)} \) is the embedding of \( v_l \)) and the up-to-date hidden states \( r_l^{(i)}, z_l^{(i)} \). The user sequence before the i-th position becomes irrelevant, and we can discard it. The output of the FFN sub-layer at each self-attention layer can be computed solely from \( s_l^{(i)} \). Therefore, for each user, the two hidden states of each self-attention block are all we need to store, and to dynamically update when a new action comes from the user. Since we use two self-attention blocks in our model, we need to maintain four hidden states in total for each user. Note that \( r_l^{(i)} \in \mathbb{R}^{d \times d} \) and \( z_l^{(i)} \in \mathbb{R}^d \). Although \( r_l^{(i)} \) is a \( d \times d \) matrix, we have \( L \gg d \) in our scenario, hence the \( O(d^2) \) cost to store the hidden states is negligible compared to the \( O(Ld) \) cost to store the whole sequence. Through maintain fixed-size states, we reduce the computational and storage complexity from \( O(L) \) to \( O(1) \) in online inference with newly arrived actions.

Compared to the RNN-based memory networks [24, 27], which utilizes trainable gates to update the memory, our incremental attention based method aggregate all the positions of the historical sequence weighted by their "importance". The (unnormalized) score assigned to the i-th position is:

\[
\varphi(W_Q h_l^{(i-1)})^\top \varphi(W_K h_{l'}^{(i-1)}) \tag{10}
\]

Self-attention possess the ability to filter out irrelevant part of the historical sequence (by assigning negligible weights) [16], and our method enables it in an incremental fashion. This is particularly helpful for lifelong sequences, where the richer historical behavior data also leads to more noise. On the other hand, RNN-based methods may get overwhelmed by noise when modeling long sequences. Besides, they use the same set of RNN parameters to update the memory for each user. However, different users’ behaviors have distinct transition patterns, using shared parameters fail to adapt to the personalized transition pattern of each individual user.

### 3.3 Extracting Users’ Multi-Facet Interests

With the increasing length of the user’s historical sequence, the more diverse the interests may contain in this sequence. Hence, if we directly model the whole sequence with a single representation, we cannot capture the multiple interests that the user has. Instead, we may get entangled with the various interests and fail to correctly identify the different driving forces behind the user’s each action. Therefore, ideally, we should disentangle the user’s whole sequence into multiple sub-sequences, each containing only the part related to a specific interest.

Coincidentally, we observe that the attention mechanism empowers us to complete this task. Computing the attention using the sequence itself as keys and values can be thought as performing a soft-search over the sequence with the given query, where the positions relevant to the query are assigned with larger weights, and they will dominate in the weighted average step. Therefore, the attention mechanism has the inherent capability to soft-extract the sub-sequences related to a query, and we use it as the building block of our proposed multi-interest extraction module.

We may characterize the various interests into a common collection that is shared among all the users, and each individual user may manifest a subset of it in his/her historical sequence. We assume the collection consists of \( K \) potential interests of users, where \( K \) is a tunable hyper-parameter. For each interest, we represent it with a trainable model parameter \( \mu_k \in \mathbb{R}^d \), which encodes the underlying incentive behind this interest, \( \mu_k \) is used to query the user sequence (after passing through the two linear self-attention blocks):

\[
\varphi_k^l = \frac{\varphi(\mu_k)^\top \sum_{l'=1}^l \varphi(\tilde{W}_K h_{l'}^{(2)}) (\tilde{W}_V h_{l'}^{(2)})^\top}{\varphi(\mu_k)^\top \sum_{l'=1}^l \varphi(\tilde{W}_K h_{l'}^{(2)})} \tag{10}
\]

Here \( \varphi_k^l \) represents the disentangled representation of the sequence at the i-th position (which encodes \( h_l^{(1)}, \ldots, h_{l'}^{(1)} \)) under the \( k \)-th interest. \( \tilde{W}_K, \tilde{W}_V \in \mathbb{R}^{d \times d} \) are the projection weights. Through linear attention, once again, we can incrementally update \( \varphi_k^l \), like what we do for the self-attention blocks in Section 3.2. Since we use the same matrices to project keys and values for each interest, we only need to maintain two hidden states, which are shared among all the \( K \) interests. The two states \( \tilde{r} \) and \( \tilde{z} \) are updated
in the following manner:

\[
\begin{align}
\tilde{r}_t &= \begin{cases}
0, & l = 0 \\
\tilde{r}_{t-1} + \varphi \left( \tilde{W}_k h^{(2)}_t \right) \left( \tilde{W}_V h^{(2)}_t \right)^T, & l > 0
\end{cases} \\
\tilde{z}_t &= \begin{cases}
0, & l = 0 \\
\tilde{z}_{t-1} + \varphi \left( \tilde{W}_k h^{(2)}_t \right), & l > 0
\end{cases}
\tag{11}
\end{align}
\]

Armed with the up-to-date \( \tilde{r} \) and \( \tilde{z} \), we are enabled to compute \( \phi^{(k)}_t \) as \( \varphi(\mu_k)^T \tilde{r}_t \varphi(\mu_k)^T \tilde{z}_t \) in \( O(1) \) time with each incoming new behavior from the user, similar to Eq. 9. The up-to-date set of vectors \( \{\phi^{(k)}_t\}_{k=1}^K \) constitutes the final multi-interest representations of the current user sequence.

### 3.4 Regularizing the Multi-Interest Representations

The multi-interest extraction module we proposed in Section 3.3 conducts a soft search over the whole user sequence to retrieve the "sub-sequence" relevant to each interest. We would like the final representations \( \{\phi^{(k)}_t\}_{k=1}^K \) to be disparate so that they can indeed capture the diverse interests of users. Otherwise, they may contain redundant information and \( \{\mu_k\} \) may collapse to encode overlapping interests. Therefore, we should enforce \( \{\phi^{(k)}_t\}_{k=1}^K \) to distill distinct information from the user sequence. We reckon that the target item (next item) should only be inferred from a single \( \phi^{(k)}_t \) (corresponding to the one with the largest inner product score), since each user behavior should only be triggered by a single specific interest. Therefore, we propose the following regularization loss:

\[
\mathcal{L}_{\text{reg}} = \sum_{k=1}^K \exp(\phi^{(k)}_t e_t) / \sum_{k=1}^K \exp(\phi^{(k)}_t e_t), \tag{13}
\]

where \( k = \arg \max_{1 \leq k \leq K} \left( \exp(\phi^{(k)}_t e_t) \right) \) and \( e_t \) is the embedding of the target item. For the sequential recommendation task, we apply the binary cross-entropy loss:

\[
\mathcal{L}_{\text{rec}} = - \log \sigma(\phi^{(k)}_t e_t) - \mathbb{E}_{j \neq \text{target}} \log \left( 1 - \sigma(\phi^{(k)}_t e_j) \right) \tag{14}
\]

Here we still denote \( k = \arg \max_{1 \leq k \leq K} \left( \exp(\phi^{(k)}_t e_t) \right) \), and \( P_n \) is a negative sampling distribution (we simply use a uniform one as in \[16\]).

The final loss function for a user is then given by:

\[
\mathcal{L} = \mathcal{L}_{\text{rec}} + \lambda \mathcal{L}_{\text{reg}}, \tag{15}
\]

where \( \lambda \) is a hyperparameter controlling the weight of the regularization loss.

### 4 EXPERIMENTS

In this section, we empirically analyze the effectiveness of our proposed method and present the experimental results. We conduct the experiments in order to answer the following research questions:

- **RQ1**: Does modeling lifelong sequences really contribute to the recommendation performance? How does LimaRec compare with state-of-the-art baselines?
- **RQ2**: How does the performance of different sequential models vary as the length of users’ historical sequences increases?
- **RQ3**: Is it essential to consider the multi-facet interests of users in lifelong sequences? How does the number of interests we model influence the recommendation performance?

#### 4.1 Datasets

We employ the following four real-world datasets to conduct experiments:

- **MovieLens** [11]: It is a series of frequently used datasets of movie ratings for evaluating recommendation systems. We use the two versions: MovieLens 1M (ML-1M) and MovieLens 25M (ML-25M), which include 1 million and 25 million ratings, respectively.
- **XLong** [27]: The first public dataset tailored for lifelong sequential modeling. This dataset is collected from Alimama’s online advertising platform, and contains relatively longer behavior sequences for each user.
- **Industrial**: A dataset sampled from user click logs on a top e-commerce platform within a period of one year.

The statistics of the datasets are shown in Table 2.

#### 4.2 Competitors

We compare the proposed LimaRec with a variety of baselines. The first group includes the state-of-the-art lifelong sequential modeling models:

- **MIMN** [24] utilizes a memory network based on NTM [10] to memorize the user’s historical sequence. It improves the traditional NTM with a memory utilization regularization strategy and a memory induction unit.
- **HPMN** [27] enhances memory networks with a hierarchical architecture with multiple update periods to mine multi-scale patterns in users’ behavior sequence.

Note that these two methods are all developed for the CTR prediction task, where the personalized memory representation is concatenated with the feature vector of the target item and some user-side features. The concatenated vector is then fed into a multi-layer perceptron to produce the estimation of the user response probability. Since in our setting the target item is not available as model input, instead, we generate the candidate set by computing the inner product scores between the user’s representation and the items. Hence, we modify MIMN and HPMN accordingly.

The second group contains the sequential recommendation methods designed to capture users’ multiple interests:

- **MIND** [18] borrows the idea of dynamic capsule routing from [29]. Different from [29], it employs shared bi-linear mapping matrix and randomly initialized routing logits.
- **ComiRec** [3] introduces two versions of multi-interest extraction modules. The first one, ComiRec-DR, follows the original
routing mechanism in [29]. The second one, ComiRec-SA, explores a self-attentive mechanism. The attention mechanism is different from the scaled dot-product attention in SASRec.

The last group is composed of:

- **GRU4Rec** [15] is the first work that introduces RNNs to model user sequential behaviours. It is originally designed for session-based recommendation.
- **SASRec** [16], the vanilla self-attention networks described in Section 3.1.2.

### 4.3 Settings

We set the maximum length of users’ lifelong sequences to 1000, as in [27], on all four datasets. For the methods that are not specially designed for lifelong sequential modeling, we also evaluate them using only recent 40 behaviors as users’ history. In other words, they are trained and evaluated on sub-sequences with length up to 40, and user sequences with more than 40 behaviors are divided into non-overlapping chunks of length 40. By comparing the performance of these methods in the two settings, we can illustrate the importance and difficulty of incorporating lifelong sequences. We note that except GRU4Rec, HPMN, MIMN and our proposed LimaRec, the models trained using lifelong sequences are not practical in the online inference setting, due to the huge computational cost.

We implement our method and all the baselines with PyTorch. We train all models using the Adam optimizer with a learning rate of 0.001 and a batch size of 128. All methods are trained for a maximum of 500 epochs and the embedding dimension is set to 32. For our model, we use a regularization coefficient $\lambda$ of 0.01 and a dropout ratio of 0.1. The number of interests $K$ on each dataset is tuned from $\{1, 2, \cdots , 9\}$. The hyperparameters of the baseline methods are also tuned to the best on each dataset.

Following [3, 16], we evaluate the performance of two widely used metrics of ranking evaluation: Hit Rate (HR) and NDCG. HR@$k$ counts the proportion of times that the ground-truth target item is among the top-$k$, while NDCG@$k$ assigns larger weights on higher positions. We report the two metrics at $k = 5$ and $k = 10$. The last item of each user’s behavior sequence is used for evaluating, and the remaining behaviors are used for training. We randomly generate 100 negative samples, pair them with the ground-truth target item for the compared methods to rank, as in [16].

Note that LimaRec, HPMN and the multi-interest baselines yield multiple representation vectors $\{\phi(k)\}_{k=1}^{K}$ for each user. We find the one who has the largest inner product score with the target item $t$: $\phi(k) = \max_{1 \leq k' \leq K}(\phi(k') e_t)$, and use the universal $\phi(k)$ to compute the scores for all the left negative items for more efficiency. The exact evaluation setting would be to, for each negative item, compute the inner product using all $K$ vectors independently, and use the maximum one as the score. That is, we should use $\max_{1 \leq k' \leq K}(\phi(k') e_j)$ as the score for negative item $j$, instead of using $\phi(k') e_j$, where $k = \arg \max_{1 \leq k' \leq K}(\phi(k') e_j)$.

### 4.4 Overall Performance Comparison

The results are summarized in Table 3 and Table 4. From the two tables, we have the following findings:

![Figure 1: Recommendation performance on users with increasingly longer behavior sequences (XLong and Industrial)](image1)

- **Finding 1**: Incorporating lifelong sequences can boost the recommendation performance. By comparing the performance of SASRec-40 with that of SASRec-1000, we find that training SASRec with lifelong sequences indeed improves the recommendation performance. Since the scaled dot-product self-attention can naturally identify relevant information in the user sequence with adaptive attention weights, SASRec can benefit from longer historical sequences without being overwhelmed by noise. We also observe that HPMN outperforms the multi-interest models in most scenarios, this further illustrates the benefits brought by lifelong sequences.

- **Finding 2**: It is not a simple task to modeling lifelong sequences. We find that most multi-interests methods and GRU4Rec, which are not tailored for lifelong sequential modeling, however, decrease in performance in most cases when feeding lifelong sequences to them. These models fail to extract richer information, but instead get consumed by noise handling far longer sequences. ComiRec-SA performs better using lifelong sequences on two of the four datasets. This again demonstrates the merit of self-attention mechanism.

- **Finding 3**: LimaRec consistently outperforms both the state-of-the-art lifelong sequential modeling baselines and the multi-interest models on all four datasets. Compared with the lifelong sequential models and SASRec, this illustrates the superiority of our incremental attention mechanism over memory networks, and the effectiveness of our proposed multi-interest extraction module.

### 4.5 Performance regarding Sequence Length

#### 4.5.1 Settings

In this section, we evaluate how does the performance of different methods varies as the length of users’ behavior
Table 3: Overall recommendation performance on MovieLens and XLong. The suffix "-40" and "-1000" indicates whether the method is trained and evaluated with lifelong sequences or using only recent behaviors. The ones without suffix use lifelong sequences.

| Method          | ML-1M Mean HR@5 | ML-1M Mean NDCG@5 | ML-1M Mean HR@10 | ML-1M Mean NDCG@10 | ML-25M Mean HR@5 | ML-25M Mean NDCG@5 | ML-25M Mean HR@10 | ML-25M Mean NDCG@10 | XLong Mean HR@5 | XLong Mean NDCG@5 | XLong Mean HR@10 | XLong Mean NDCG@10 |
|-----------------|-----------------|-------------------|-----------------|-------------------|-----------------|------------------|-----------------|------------------|-----------------|-----------------|-----------------|-----------------|
| GRURec-40       | 0.2646          | 0.1847            | 0.3805          | 0.2221            | 0.8206          | 0.6526           | 0.9047          | 0.6801           | 0.1328          | 0.0820          | 0.1603          | 0.0910          |
| GRURec-1000     | 0.2576          | 0.1775            | 0.3770          | 0.2158            | 0.8141          | 0.6484           | 0.8967          | 0.6754           | 0.1326          | 0.0822          | 0.1673          | 0.0935          |
| ComiRec-DR-40   | 0.4192          | 0.2945            | 0.5760          | 0.3451            | 0.8964          | 0.7526           | 0.9296          | 0.7637           | 0.4005          | 0.2625          | 0.5916          | 0.3241          |
| ComiRec-DR-1000 | 0.3997          | 0.2777            | 0.5531          | 0.3271            | 0.8342          | 0.6657           | 0.8960          | 0.6860           | 0.2195          | 0.1627          | 0.2842          | 0.1834          |
| ComiRec-SA-40   | 0.3901          | 0.2651            | 0.5331          | 0.3175            | 0.8094          | 0.6239           | 0.8992          | 0.6534           | 0.2977          | 0.1963          | 0.4898          | 0.2578          |
| ComiRec-SA-1000 | 0.4169          | 0.2874            | 0.5765          | 0.3389            | 0.8189          | 0.6404           | 0.9046          | 0.6685           | 0.2306          | 0.1555          | 0.3712          | 0.2004          |
| MIND-40         | 0.5232          | 0.3624            | 0.6911          | 0.4169            | 0.8783          | 0.7261           | 0.9331          | 0.7440           | 0.2106          | 0.1317          | 0.2607          | 0.1481          |
| MIND-1000       | 0.5331          | 0.3693            | 0.6995          | 0.4233            | 0.8710          | 0.7229           | 0.9325          | 0.7429           | 0.1844          | 0.1205          | 0.2350          | 0.1367          |
| MIMN            | 0.2414          | 0.1594            | 0.3747          | 0.2023            | 0.7878          | 0.5997           | 0.8902          | 0.6332           | 0.1966          | 0.1334          | 0.2385          | 0.1470          |
| HPMN            | 0.4561          | 0.3168            | 0.6149          | 0.3682            | 0.9283          | 0.8270           | 0.9559          | 0.8359           | 0.3700          | 0.2599          | 0.4796          | 0.2944          |
| SASRec-40       | 0.6631          | 0.5026            | 0.7772          | 0.5397            | 0.9265          | 0.7936           | 0.9710          | 0.8083           | 0.3337          | 0.2359          | 0.4657          | 0.2783          |
| SASRec-1000     | 0.6954          | 0.5428            | 0.8043          | 0.5786            | 0.9325          | 0.7982           | 0.9755          | 0.8124           | 0.6805          | 0.5282          | 0.8075          | 0.5675          |
| LimaRec         | 0.8699          | 0.6774            | 0.9606          | 0.7072            | 0.9699          | 0.8717           | 0.9867          | 0.8772           | 0.8207          | 0.6682          | 0.8931          | 0.6918          |

Table 4: Overall recommendation performance on the Industrial dataset.

| Method          | ML-1M Mean HR@5 | ML-1M Mean NDCG@5 | ML-1M Mean HR@10 | ML-1M Mean NDCG@10 | ML-25M Mean HR@5 | ML-25M Mean NDCG@5 | ML-25M Mean HR@10 | ML-25M Mean NDCG@10 |
|-----------------|-----------------|-------------------|-----------------|-------------------|-----------------|------------------|-----------------|------------------|
| GRURec-40       | 0.1772          | 0.1241            | 0.2258          | 0.1399            |                |                  |                  |                  |
| GRURec-1000     | 0.1744          | 0.1274            | 0.2247          | 0.1436            |                |                  |                  |                  |
| ComiRec-DR-40   | 0.4616          | 0.3361            | 0.6145          | 0.3856            |                |                  |                  |                  |
| ComiRec-DR-1000 | 0.4158          | 0.3073            | 0.5523          | 0.3514            |                |                  |                  |                  |
| ComiRec-SA-40   | 0.4502          | 0.3284            | 0.5985          | 0.3761            |                |                  |                  |                  |
| ComiRec-SA-1000 | 0.2906          | 0.1964            | 0.4263          | 0.2401            |                |                  |                  |                  |
| MIND-40         | 0.2220          | 0.1580            | 0.2766          | 0.1758            |                |                  |                  |                  |
| MIND-1000       | 0.2443          | 0.1835            | 0.3096          | 0.2047            |                |                  |                  |                  |
| MIMN            | 0.2665          | 0.2003            | 0.3476          | 0.2265            |                |                  |                  |                  |
| HPMN            | 0.3578          | 0.2961            | 0.4262          | 0.3181            |                |                  |                  |                  |
| SASRec-40       | 0.4725          | 0.3644            | 0.3782          | 0.3986            |                |                  |                  |                  |
| SASRec-1000     | 0.7701          | 0.6349            | 0.8610          | 0.6644            |                |                  |                  |                  |
| LimaRec         | 0.8000          | 0.6555            | 0.8911          | 0.6851            |                |                  |                  |                  |

Figure 3: The impact of the number of interests $K$ on the hit rate and diversity.

for ML-1M and ~25 yrs for ML-25M). For a user with hundreds of actions, these actions may span a period of several years. On the other hand, the users with few actions may generate all of them within a short period, typically a day, and never log in again. The users with longer sequences are, in fact “sparser” than the ones with few behaviors. Hence, they contain much noise and would have fuzzy behavior transition patterns, and it is difficult to model them. Nevertheless, using lifelong sequences still contributes to the recommendation performance, as we see in Table 3. Moreover, we see that on these two datasets, the performance of our proposed LimaRec decreases slower than the baseline methods when the sequence length increases, which indicates the superiority of LimaRec in modeling lifelong sequential behaviors.

4.6 Analysis of the Multi-Interest Extraction Module

4.6.1 Settings. Here we empirically investigate the effectiveness of our proposed incremental attention based multi-interest extraction module. We vary the number of interests $K$ modeled in LimaRec from 1 to 9, train the model, and measure the hit rate. To better understand the nature of users’ multi-facet interests, we also evaluate the recommendation diversity with respect to different $K$. We employ the diversity metric Diversity@$k$ in [3], which computes the...
fraction of heterogeneous item-pairs (having different categories) in the top-\(k\). The results are reported in Figure 3.

4.6.2 Findings. We observe that the hit rate using a single interest is significantly worse than utilizing multiple interests, as the lifelong sequences of users, by nature, contain diverse interests. When we set \(k = 1\), we are essentially using a single interest vector to query the latent representation of the sequence (after going through the two incremental self-attention blocks and projecting it to a new latent space). This would only amplify the noise contained in lifelong sequences. We obtain considerable performance boosts (in terms of hit rate) with increasing \(k\). Hence it is quite essential to consider the multi-facet interests of users. Since it becomes easier for the model to recommend more diverse items with a larger \(k\), we also find that, in general, the recommendation diversity tends to increase as \(k\) increases. Though there are still some fluctuations when \(k\) varies, it may be caused by the inconsistency between the \(k\) we employed and the true number of interests in the user’s historical sequence (which is unknown). One has to carefully tune \(k\) to strike a balance between the traditional metrics like hit rate/NDCG, and the recommendation diversity.

5 CONCLUSIONS
In this work, we considered the problem of lifelong sequential recommendation, where the user representation has to be continuously updated in constant time to satisfy the strict latency and storage constraints for online systems, while still be able to preserve the behavior patterns in the historical sequence. Differing from the lifelong modeling methods based on RNN memory networks, we propose a novel model built upon incremental self-attention. The self-attention mechanism enables us to identify and retrieve relevant information from the user’s lifelong sequence, without succumbing to massive noise contained in it. We further propose a multi-interest extraction module to soft-search the lifelong sequence for the behaviors relevant to each interest. Extensive experiments have demonstrated the superiority of our method.

REFERENCES
[1] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V Le. 2019. Attention augmented convolutional networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision.
[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks 5, 2 (1994).
[3] 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.
[4] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiasi Tang, Yuxin Cao, Zheng Qin, and Hongyuan Zhu. 2018. Sequential recommendation with user memory networks. In Proceedings of the eleventh ACM international conference on web search and data mining. 108–116.
[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014).
[6] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andrsee Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. arXiv preprint arXiv:2009.14794 (2020).
[7] Paul Coghlan, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM conference on recommender systems.
[8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.
[9] 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).
[10] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing machines. arXiv preprint arXiv:1410.5401 (2014).
[11] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. ACM transactions on interactive intelligent systems (tiis) 5, 4 (2015), 1–19.
[12] Ruing He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In Proceedings of the eleventh ACM conference on recommender systems. 161–169.
[13] Ruing He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In 2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 191–200.
[14] Balazs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In Proceedings of the 27th ACM international conference on information and knowledge management. 843–852.
[15] Balazs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. arXiv preprint arXiv:1511.06939 (2015).
[16] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In 2018 IEEE Conference on Data Mining (ICDM).
[17] Angelos Katharopoulos, Apostov Vyas, Nikolaos Pappas, and Francois Fleuret. 2020. Transformers are riffs: Fast autoregressive transformers with linear attention. In International Conference on Machine Learning. PMLR, 5156–5165.
[18] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipi Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Liu 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.
[19] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Liu, 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] Defu Lian, Yongji Wu, Yong Ge, Xing Xie, and Enhong Chen. 2020. Geography-aware sequential location recommendation. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.
[21] Valerii Likhosherstov, Krzysztof Choromanski, Jared Davis, Xingyou Song, and Adrian Weller. 2020. Sub-Linear Memory: How to Make Performers SLiM. arXiv preprint arXiv:2012.11346 (2020).
[22] Qiang Liu, Shu Wu, Dyi Wang, Zhaokang Li, and Liang Wang. 2016. Context-aware sequential recommendation. In 2016 IEEE 16th International Conference on Data Mining (ICDM). IEEE, 1053–1058.
[23] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In International Conference on Machine Learning. PMLR, 4055–4064.
[24] 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 Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.
[25] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Leijian 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 Proceedings of the 29th ACM International Conference on Information & Knowledge Management.
[26] Jiarui Qin, Weian Zhang, Xin Wu, Jiarui Jin, Yuchen Fang, and Yong Yu. 2020. User Behavior Retrieval for Click-Through Rate Prediction. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval.
[27] Kan Ren, Jiarui Qin, Yuchen Fang, Weian 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 Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval.
[28] 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.
[29] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic Routing Between Capsules. In Advances in Neural Information Processing Systems, Vol. 30.
[30] 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 26th ACM international conference on information and knowledge management. 1441–1450.
[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems.

[33] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. 2018. Non-local Neural Networks. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[34] 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.

[35] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 1192–1200.

[36] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A dynamic recurrent model for next basket recommendation. In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval. 729–732.

[37] Shuai Zhang, Yi Tay, Lina Yao, and Aixin Sun. 2018. Next item recommendation with self-attention. arXiv preprint arXiv:1808.06444 (2018).

[38] 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 Proceedings of the AAAI conference on artificial intelligence, Vol. 33. 5941–5948.

[39] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Juen Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 1059–1068.
A APPENDIX

A.1 Dataset Pre-processing
For MovieLens datasets we treat the presence of a movie rating as implicit feedback (i.e., we consider it as an action in the user’s behavior history). For all datasets, we drop users and items with less than 5 related actions. For the Industrial dataset, 1% users are randomly sampled from the ones who are active on February 1, 2021, and their click histories are collected. The full user-item interaction data contains 71,086,374 items, and 20,059,521,441 actions from 42,995,680 users. The negative items used in evaluation for each user are randomly sampled from the set of items that the user has not interacted with using a uniform distribution. We plan to open this dataset to further nourish the community development in this field.

A.2 Parameter Settings
As mentioned above, we implement our method and all the baselines with PyTorch. We use an efficient implementation of the linear attention mechanism [6] provided by [21] for our LimaRec. All models are trained with an embedding dimension of 32, and a dropout rate of 0.1. We set the maximum number of training epochs to 500 on the ML-1M dataset, 200 on the ML-25M and XLong datasets, and 20 for Industrial. The number of self-attention blocks are set to 2 for SASRec, and we employ a single attention head, as it is pointed out in [16] that using multiple attention heads does not contribute to the recommendation performance. The regulation coefficient is set to 0.01 for HPMN and MIMN. We tune the number of interests used in ComiRec and MIND, as well as the number of hierarchical GRU layers in HPMN from 1 to 9. The number of memory slots of MIMN are set to 3.

A.3 Experiment Environment
We implement all models with PyTorch, and conduct experiments on a server with NVIDIA V100 GPUs. Regarding software versions, Python 3.6.9 and PyTorch 1.8.0 are used.

A.4 Ablation Study of Multi-Interest Extraction Module
A.4.1 Settings. In this section, we conduct an ablation analysis to further study the effectiveness of the multi-interest extraction module, and to compare the performance of incremental attention with vanilla self-attention. Hence we evaluate the recommendation performance of LimaRec without the multi-interest extraction layer, by only using the two incremental self-attention blocks. The results are presented in Table 5.

Table 5: Performance of LimaRec without the Multi-Interest Extraction Module.

|          | HR@5  | NDCG@5 | HR@10 | NDCG@10 |
|----------|-------|--------|-------|---------|
| ML-1M    | 0.6548| 0.5003 | 0.7803| 0.5410  |
| ML-25M   | 0.9172| 0.7735 | 0.9686| 0.7904  |
| XLong    | 0.6648| 0.5058 | 0.7917| 0.5470  |
| Industrial | 0.7518| 0.6147 | 0.8463| 0.6454  |

A.4.2 Findings. We find that LimaRec performs notably worse without the multi-interest extraction module, which further indicates the necessity to model and capture the multi-facet interests in users’ lifelong sequences. Compare the results with SASRec-1000 in Table 3, we observe that the performance of LimaRec without multi-interest modeling (note that in this case, LimaRec only utilizes the two incremental self-attention blocks) is only slightly worse than SASRec using lifelong sequences. This proves that the incremental attention mechanism indeed share the same power of vanilla to capture long-term dependencies in lifelong sequences, identify relevant information and filter out noise.