CmnRec: Sequential Recommendations with Chunk-accelerated Memory Network

Shilin Qu*, Fajie Yuan*†, Guibing Guo†, Liguang Zhang, and Wei Wei

Abstract—Recently, Memory-based Neural Recommenders (MNR) have demonstrated superior predictive accuracy in the task of sequential recommendations, particularly for modeling long-term item dependencies. However, typical MNR requires complex memory access operations, i.e., both writing and reading via a controller (e.g., RNN) at every time step. Those frequent operations will dramatically increase the network training time, resulting in the difficulty in being deployed on industrial-scale recommender systems. In this paper, we present a novel general **Chunk** framework to accelerate MNR significantly. Specifically, our framework divides proximal information units into chunks, and performs memory access at certain time steps, whereby the number of memory operations can be greatly reduced. We investigate two ways to implement effective chunking, i.e., PEriodic Chunk (PEC) and Time-Sensitive Chunk (TSC), to preserve and recover important recurrent signals in the sequence. Since chunk-accelerated MNR models take into account more proximal information units than that from a single timestep, it can alleviate the influence of noise in the user-item interaction sequence to a large extent, and thus improve the stability of MNR. In this way, the proposed chunk mechanism can lead to not only faster training and prediction, but even slightly better results. The experimental results on three real-world datasets (weishi, ml-10M and ml-latest) show that our chunk framework notably reduces the running time (e.g., with up to 7x for training & 10x for inference on ml-latest) of MNR, and meantime achieves competitive performance.

Index Terms—Sequential Recommendation, Memory Network, Chunk, RNN.

1 Introduction

With the rapid development of Web 2.0, the speed of data production and streaming has gone up to a great extent. Meanwhile, Internet users can easily access various online products and services, which results in a large amount of action feedback. The extensive user feedback provides a fundamental information source to build recommender systems, which assist users in finding relevant products or items of interest. Since users generally access items in chronological order, the item a user will next interact with may be closely relevant to the accessed items in a previous time window. The literature has shown that it is valuable to consider time information and preference drift for better recommendation performance [1], [2], [3], [4], [5]. In this paper, we focus on the task of sequential (a.k.a., session-based) recommendation, which is built upon the historical behavior trajectory of users.

A critical challenge for sequential recommendation is to effectively model the preference dynamics of users given the behavior sequence. Among all the existing methodologies, Recurrent Neural Networks (RNN) have become the most prevalent approaches with remarkable success [1], [2]. Different from feedforward networks, the weights of RNN can be well preserved and updated over time via its internal state, which endows RNN with the ability to process sequence. However, learning vanilla RNN for long-term dependencies remains a fundamental challenge due to the vanishing gradient problem [6], and it is noted that long-range user sessions widely exist in real applications. For example, users on TikTok [1] can watch hundreds of mico-videos in an hour since the average playing time of each video takes only 15 seconds. To model long-term item dependencies for the sequential recommendation problem, previous attempts have introduced Long Short-Term Memory (LSTM) [7] & Gated Recurrent Units (GRU) [1], temporal convolutional neural network architecture with dilated layers [4], [8], attention machine [9], [10], and external memory [11], [12].

Among these advanced methods, the External Memory Network (EMN) [13], [14] mostly resembles human cognitive architecture due to its enhanced external memory mechanism. EMN is composed of a neural controller, e.g., RNN, and the external memory, which can be regarded as an extension of standard RNN, including LSTM & GRU. Unlike RNN, EMN stores useful past information by external memory rather than a squeezed vector. EMN has shown high potentials in areas, such as visual reasoning [15], question answering [16], natural language processing [17]. Since 2018, researchers started to apply it in the field of recommendation to improve the accuracy of existing recurrent models

- * *Equal contribution.*
- † *Corresponding authors.*
- *Shilin Qu is with Monash University, Melbourne , Australia (affiliation). A part of this work was finished when Shilin was a master student at Northeastern University, an intern at Tencent Kandian Group, and an research assistant at Westlake University.*
- *Fajie Yuan is with Westlake University, Hangzhou 310024, P.R. China. A part of this work was finished when Fajie was AI researcher at Tencent Kandian Group. E-mail: yuanfajie@westlake.edu.cn*
- *G. Guo is with Northeastern University, Shenyang 110819, P.R. China. E-mail: guogb@swc.neu.edu.cn*
- *L. Zhang is with Kandian, PCG, Tencent, Shenzhen 518055, P.R. China. E-mail: liguangzhang@tencent.com*
- *W. Wei is with Huazhong University of Science and Technology, Wuhan 430074, P.R. China. E-mail: weiw@hust.edu.cn*

Manuscript received April 6, 2021; revised November 22, 2021; accepted December 23, 2021.

1. https://www.tiktok.com/en/

Fig. 1. Illustration of the Chunk mechanism for better memorization. Numerics and alphabets are chunked into numeric units and words for faster and easier remembering.

[11], [12], [18], [19], in the following referred to as Memory-based Neural Recommenders (MNR).

In order to remember more information, MNR implementations require to repeat the memory accessing operations, including both reading and writing, at every time step [11], [12], [13], [14], [18], [19]. The reading and writing accessing operations are much more expensive than the controller in terms of time complexity, which becomes a severe efficiency problem when modeling long-range sequences. One possible way to speed up MNR is to optimize the specific memory operations directly. However, there are many different implementations of accessing operations. For example, RUM [11] updates the entire memory slot iteratively as a first-in-first-out queue, and NMRN [12] addresses specific memory slots with attention and updates them. It is a challenge to find commonalities in these accessing operations to improve on. In this paper, we focus on developing a general acceleration framework that applies to various types of MNR.

Our central idea to accelerate MNR in this paper takes inspiration from the chunk [20] technique in cognitive psychology, where the concept of it is introduced to improve human's memory. Chunk here refers to a meaningful unit of information that can be reorganized based on certain rules. For example, giving the letter sequence "***m-e-m-o-r-y***", we can remember it as six separate letters, or memorize it by the word "***memory***", as illustrated in Figure 1. The latter method can greatly reduce our memory burden but maintain the same amount of information. As such, we believe that applying the chunk strategy for MNR is a promising way to improve the efficiency issue of MNR.

In this paper, we propose a sequential recommendation framework with chunk-accelerated memory network (CmnRec for short), which speeds up the memory network by reducing the number of memory operations. Our chunk framework consists of the chunk region, chunk rule and attention machine. Specifically, the chunk region temporarily stores the information units (the output vector of the controller) generated in the non-chunk time. The chunk rule determines when (i.e., chunk time) to perform memory operations. The attention machine extracts the most valuable information in the chunk region, generating new information units to perform memory operations. Through the functions of these modules and rules, chunk compresses the information ingested in the past with high quality, which substantially reduces the workload of memorization so as to improves the recommendation efficiency.

To sum up, the main contributions of this paper include:

- We propose a general chunk-based sequential recommendation framework, which significantly accelerates various MNRs without harming the accuracy. To the best of our knowledge, this is the first work to evidence that using less memorization can enable comparable accuracy for the recommendation task.
- We present two effective implementations for CmnRec: **pe**riodic **c**hunk (PEC) considering the input of each time to be equally important and **t**ime-**s**ensitive **c**hunk (TSC) taking into account both long and short-term dependencies.
- We compare CmnRec with state-of-the-art sequential recommendation methods on three real-world datasets. Our experimental results demonstrate that CmnRec offers competitive and robust recommendations with much less training and inference time.

2 Related Work

This work can be regarded as an integration of sequential recommendation and memory networks. In the following, we briefly review related literature in the two directions.

2.1 Sequential Recommendation

Sequential (a.k.a., session-based) recommender systems are an emerging topic in the field of recommendation and have attracted much attention in recent years due to the advance of deep learning. Existing sequential recommendation models can be mainly categorized into three classes according to the models they involved [21]: Markov chain-based methods [22], [23], factorization based methods [24], [25], [26], and deep learning-based methods [1], [3], [4], [27]. Specifically, due to the efficiency consideration, Markov chain based recommenders are typically built on the first-order dependency assumption, and thus only capture the first-order dependency over items. As a result, these methods usually do not perform well when modeling long-term and higher-order item dependencies. Factorization-based recommenders (a.k.a., Factorization Machines [24]) deal with previous user actions as general features by merely summing all their embedding vectors, and are not able to explicitly model the sequential dynamic and patterns in the user session. Thanks to the development of deep neural networks, various deep learning-based sequential models have been proposed and shown superior performance in contrast to the above-mentioned conventional methods by utilizing the complex network architectures.

To be specific, a pioneering work by Hidasi et al. [1] introduced RNN into the field of recommender systems. They trained a Gated Recurrent Unit (GRU) architecture to model the evolution of user interests, referred to as GRU4Rec. Following this idea, several other RNN variants have been proposed in the past three years. [28] proposed an improved GRU4Rec by introducing data augmentation and embedding dropout techniques. Hidasi and Karatzoglou [27] further proposed a family of alternative ranking objective functions with effective sampling tricks to improve the cross-entropy and pairwise ranking losses. [2] proposed a personalized sequential recommendation model with hierarchical recurrent neural networks, while [29], [30] explored how to leverage content and context features to improve the recommendation accuracy further. Recently, researchers have proposed several other neural network architectures, including convolutional neural networks (CNN) Caser [3]

and NextItNet [4], self-attention models SASRec [10]. Compared with RNN models, CNN and attention architectures are much easier to be parallelized on GPUs.

2.2 EMN and MNR

More recently, External Memory Network (EMN) has attracted significant attention in research fields that process sequential data. Generally, EMN involves two main parts: an external memory matrix to maintain state, and a recurrent controller to operate (i.e., reading and writing) the matrix [11]. Compared with standard RNN models compressing historical signals into a fixed-length vector, EMN is more powerful in dealing with complex relations and long distances due to the external memory. EMN has successfully applied in domains, such as neural language translation [31], question answering [32] and knowledge tracking [33]. Recently, researchers in [11], [12], [18] have applied it in recommender systems to capture user sequential behaviors and evolving preferences.

As the first work that introduces EMN into the recommendation system, sequential recommendation with user memory network (a.k.a., RUM) [11] has successfully demonstrated superior advantages over traditional baselines. Similarly, neural memory streaming recommender networks with adversarial training [12] proposes a key-value memory network for each user to capture and store both short-term and long-term interests in a unified way. Meanwhile, Ebesu et al. proposed collaborative memory network [18] that deals with all user embedding collections as user memory matrix and utilizes the associative addressing scheme of the memory operations as a nearest neighborhood model.

Ignoring the implementation of external memory networks and the memory operation, all EMN-style models need to perform memory reading and writing operations at every timestep. Such persistent memory operations significantly increase the model complexity and training/inference time, which limits the applications of MNR in large-scale industrial recommender systems. In general, efficiency can be achieved by either reducing the complexity or the frequency of memory access [34], [35]. Since there are many ways to implement EMN, we hope to propose a general acceleration framework. To achieve this goal, we propose reducing the number of memory operations, which aims to accelerate MNR with various implementations of memory operations.

3 Memory-based Neural Recommendation (MNR)

In this section, we will introduce the generic architecture of memory-based sequential recommendation. Let $\mathbb{I}$, $\mathbb{S}$ and $\{x_1, x_2, x_3, ..., x_T\}$ (interchangeably denote by $x_{1:T}$) be the set of all items, sequences and items in a specific sequence, respectively. Denote $I = |\mathbb{I}|$ and $S = |\mathbb{S}|$ as the size of item and sequence sets. The corresponding item embedding vectors are $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, ..., \boldsymbol{v}_T\}$.

Figure 2 is a classical RNN-based recommendation architecture (e.g. GRU4Rec). From bottom to top, the model includes an embedding layer, controller layer, feedforward and the softmax layer of predicted items.

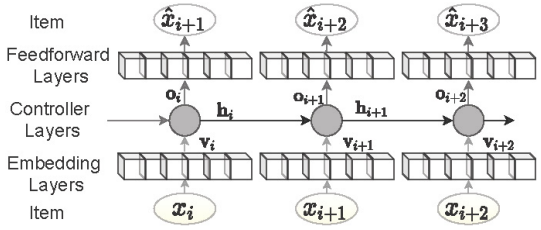


Fig. 2. RNN-based Recommendation.

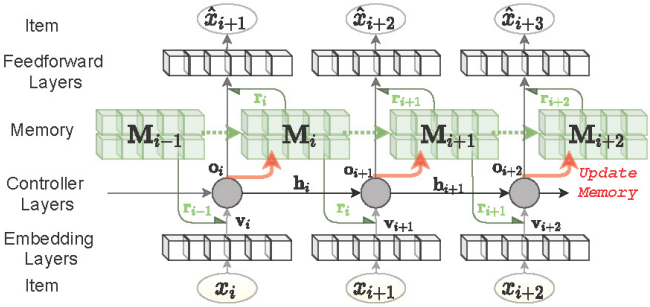


Fig. 3. Memory-based Neural Recommendation (MNR).

Figure 3 is a generic memory-based neural recommendation architecture. Similar to GRU4Rec, it also includes embedding, controller, feedforward and the softmax layer. The embedding and controller layers perform exactly the same manner as a classic RNN. The essential difference between MNR and GRU4Rec lies in the memory network layer. In fact, MNR can also be seen as an extension of RNNs with external memory network $\boldsymbol{M} \in \mathbb{R}^{m*n}$, where m is the number of memory slots and n is the embedding size of memory slot.

As shown in Figure 3, each controller will concatenate the embeddings of the current input item $\boldsymbol{v}_i$ and the memory $\boldsymbol{r}_{i-1} \in \mathbb{R}^n$ (read from $\boldsymbol{M}$) at the previous moment as an external input. The memory storage will be updated according to the output of the controller $\boldsymbol{o} \in \mathbb{R}^h$. Finally, both the controller output and updated memory $\boldsymbol{r}_i$ will be fed into the feedforward layer, which generates the probabilities of the next interacted item, formulated as follows:

$$\boldsymbol{f}_i = softmax(\Gamma_f(\boldsymbol{o}_i, \boldsymbol{r}_i)) \tag{1}$$

$$\hat{x}_i = maxID(\boldsymbol{f}_i) \tag{2}$$

where $\Gamma_f(\cdot,\cdot)$ is a feedforward operation that performs a linear transformation of the final hidden layer and returns a feature vector $\boldsymbol{f}_i$ as the output. $maxID(f_i)$ is a function to find the item ID with the maximum value in the vector, that is, the maximum occurrence probability at the i-th moment predicted by MNR.

Let the hidden state of the last moment be $\boldsymbol{h}_{i-1} \in \mathbb{R}^h$. The memory writing operation $\Gamma_w(\cdot,\cdot)$ and reading operation $\Gamma_r(\cdot,\cdot)$ can be represented as Eq.(3) and Eq.(4):

$$\boldsymbol{M}_i = \Gamma_w(\boldsymbol{o}_i, \boldsymbol{M}_{i-1}) \tag{3}$$

$$\boldsymbol{r}_i = \Gamma_r(\boldsymbol{o}_i, \boldsymbol{M}_i) \tag{4}$$

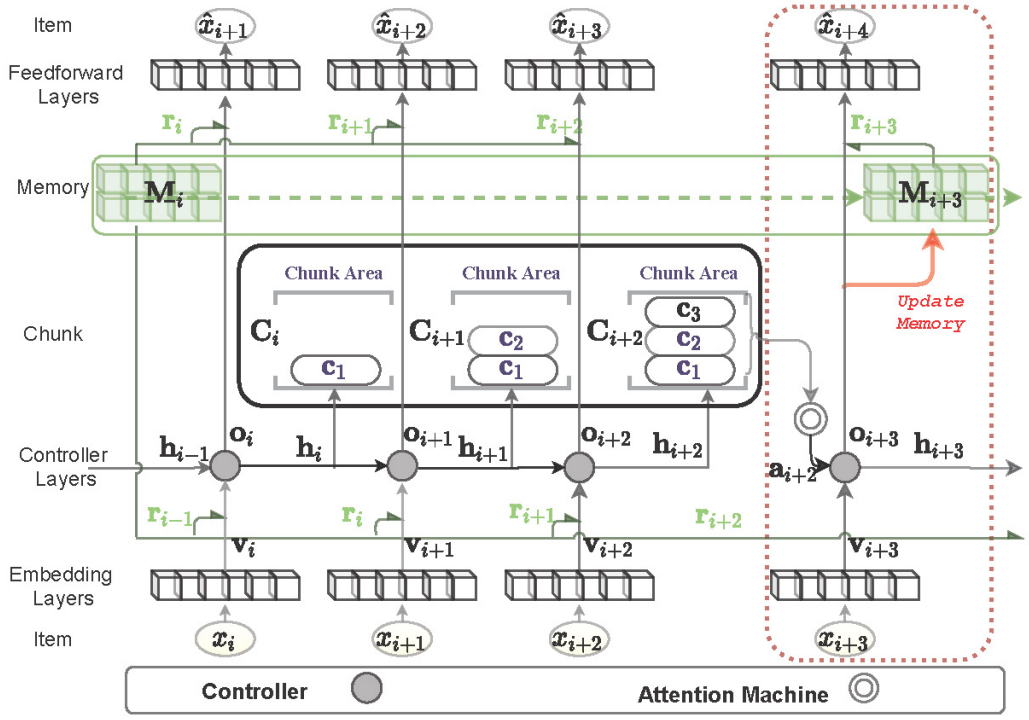


Fig. 4. Chunk acceleration on Memory-based Neural Recommendation (MNR). During non-chunk time, information units (the hidden state from the controller) are put into the chunk area. When the chunk time comes, the attention machine will extract the most valuable information in the chunk area and generate a new information unit to replace the current hidden state. Then, the information units stored in the chunk area will be emptied; the memory reading and writing operations are triggered. The dotted red box on the right denotes that MNR performs a complete process from information input and memory update to the final generation of prediction results.

The controller output and hidden state are updated as $\Gamma_o(\cdot,\cdot)$ and $\Gamma_h(\cdot,\cdot)$:

$$\boldsymbol{o}_i = \Gamma_o(\boldsymbol{r}_{i-1}, \boldsymbol{h}_{i-1}, \boldsymbol{v}_i,) \tag{5}$$

$$\boldsymbol{h}_i = \Gamma_h(\boldsymbol{r}_{i-1}, \boldsymbol{h}_{i-1}, \boldsymbol{v}_i,) \tag{6}$$

Normally, Γ_o and Γ_h are implemented as RNNs. As for Γ_w and Γ_r, they have different implementations depending on the selected (external) memory type. In this paper, we adopt the implementations of DNC [14] given its generality.[2]

4 CMNREC

In this section, we will give a detailed description of the chunk framework, followed by the concrete implementations.

4.1 From Psychology to Recommendation

Psychology points out that people unconsciously use chunk strategies to reduce the "things" to be remembered to improve the efficiency of memorization [20]. Taken inspiration from this, our central idea of chunk acceleration for MNR is to combine nearby information units according to specific rules and generate new information units so as to reduce the frequency of memory operations and improve memory efficiency. Therefore, how to find an appropriate rule of chunk is the critical question.

From a more generic perspective, information units are all converted from discrete item sets. An intuitive method is to chunk the information units based on the position of items. In practice, items are ordered chronologically in the sequence, so the rule of chunk becomes a sequence segmentation problem. That is, how to segment item sequences to minimize information loss while improving memory efficiency?

4.2 Framework

The basic idea of chunk-based memory neural network is formed on a specific sequence partitioning rule (described in section 4.4), where it first divides the close items into different chunks by order, and then writes these chunks into memory. Suppose the memory slot number of the MNR is m, and the length of the sequence is T. The whole sequence $x_{1:T}$ will be divided into m subsequences $x_{1:t_1}$, $x_{t_1:t_2}$,..., $x_{t_{m-1}:t_m}$ ($t_m = T$), where $t_1, t_2, ..., t_m$ are chunk time. The controller hidden states $\boldsymbol{h}$ corresponding to these m subsequences will be chunked m times.

In our chunk framework, the controller output $\boldsymbol{o}$ does not operate memory at every time step, which is the key difference from the standard MNRs [11], [12], [18], [19]. To do this, we create a chunk area $\boldsymbol{C} \in \mathbb{R}^{l*h}$ to store $\boldsymbol{h}$ temporarily. During non-chunk time, $\boldsymbol{C}$ caches $\boldsymbol{h}$. For every $\boldsymbol{h}$ that $\boldsymbol{C}$ caches, l increases by 1. When the chunk

2. We refer interested readers to the original paper for detailed explanation due to limited space.

time arrives, the attention machine converts $\boldsymbol{C}$ to a new controller hidden state $\boldsymbol{a} \in \mathbb{R}^h$ and then replaces the hidden state in the current controller. Finally, the chunk area is emptied ($l = 0$) and memory is manipulated. The formulas of the chunk process are expressed as follows:

$$\boldsymbol{C} = concat(\boldsymbol{h}_{i-l+1}, \boldsymbol{h}_{i-l+2}, ..., \boldsymbol{h}_{i-1}, \boldsymbol{h}_i) \tag{7}$$

$$\boldsymbol{z_{ij}} = \boldsymbol{w} tanh(\boldsymbol{W}\boldsymbol{c}_j + \boldsymbol{U}\boldsymbol{r}_{i-1}) \tag{8}$$

$$a_{ij} = softmax(\boldsymbol{z_{ij}}) \tag{9}$$

$$\boldsymbol{a}_i = \sum_{j=1}^{l} a_{ij}\boldsymbol{c}_j \tag{10}$$

where $\boldsymbol{c}_j$ is the j-th element of $\boldsymbol{C}$, a_{ij} is the attention score of $\boldsymbol{c}_j$ at i-th time step, and r_{i-1} is the read vector at time step $i-1$. $\boldsymbol{W} \in \mathbb{R}^{b*h}$, $\boldsymbol{U} \in \mathbb{R}^{b*h}$ and $\boldsymbol{w} \in \mathbb{R}^b$ are parameters, where b is attention dimension. And Figure 4 shows the architecture. Algorithm 1 summarizes the whole process of chunk-enhanced MNR. The theoretical complexity analysis is attached in Appendix A.

Algorithm 1: CmnRec

Input: a original sequence item IDs $x_{1:T-1}$,
a chunk matrice $\boldsymbol{C}$,
a memory slot number m.
Output: The predicted sequential item IDs $\hat{x}_{2:T}$

Generates chunk time steps set $Ctime$ using Eq.(16);
for $i = 1; i < T-1; i++$ **do**
  $\boldsymbol{C}.add(h_i)$;
  // The operation of the controller and memory in the chunk moment.
  if (i *in* $Ctime$) **then**
    Use Eq.(8) (9) and (10) to calculate $\boldsymbol{a}_i$;
    Perform Eq.(5): $\boldsymbol{o}_i = \Gamma_o(\boldsymbol{r}_{i-1}, \boldsymbol{a}_i, \boldsymbol{v}_i)$;
    Perform Eq.(6): $\boldsymbol{h}_i = \Gamma_h(\boldsymbol{r}_{i-1}, \boldsymbol{a}_i, \boldsymbol{v}_i)$;
    Perform Eq.(3): $\boldsymbol{M}_i = \Gamma_w(\boldsymbol{o}_i, \boldsymbol{M}_{i-1})$;
    Perform Eq.(4): $\boldsymbol{r}_i = \Gamma_r(\boldsymbol{o}_i, \boldsymbol{M}_i)$;
    $\boldsymbol{C}.empty()$
  // The operation of the controller in the non-chunk moment.
  else
    Perform Eq.(5): $\boldsymbol{o}_i = \Gamma_o(\boldsymbol{r}_{i-1}, \boldsymbol{h}_{i-1}, \boldsymbol{v}_i)$;
    Perform Eq.(6): $\boldsymbol{h}_i = \Gamma_h(\boldsymbol{r}_{i-1}, \boldsymbol{h}_{i-1}, \boldsymbol{v}_i)$;
    $\boldsymbol{r}_i = \boldsymbol{r}_{i-1}$;
  // Predict the item ID with the highest probability.
  Perform Eq.(1): $\boldsymbol{f}_i = softmax(\Gamma_f(\boldsymbol{o}_i, \boldsymbol{r}_i))$;
  Perform Eq.(2): $\hat{x}_i = maxID(\boldsymbol{f}_i)$

4.3 Analysis of implementation theory

In this subsection, we introduce a metric to evaluate the memorization ability of RNN and Chunk, which helps guide the establishment of appropriate chunk rules.

Let $\boldsymbol{h}_t = \Psi(\boldsymbol{h}_{t-1}, \boldsymbol{v}_t)$ be the transformation formula for hidden states in RNN. We define the concept of "contribution" to measure the strength of the influence, and use the norm of gradient $\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \right\|$ and $\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{v}_t} \right\|$ to represent the contributions of $\boldsymbol{h}_{t-1}$ and $\boldsymbol{v}_t$ to $\boldsymbol{h}_t$. Since the gradient represents the rate of change of trainable variables, the larger the gradient norm, the greater the contribution it has. As RNN is a cyclic structure, it is also able to measure the contribution of $\boldsymbol{h}_i$ and $\boldsymbol{v}_i$ (to $\boldsymbol{h}_t$) by $\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_i} \right\|$ and $\left\| \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{v}_i} \right\|$. Tersely, let $p_{i,t}$ and $q_{i,t}$ denote these two terms. For standard RNNs, there must be values $\vartheta_p, \vartheta_q \in \mathbb{R}^+$ that satisfy $\vartheta_p p_{i,t} \geq p_{i-1,t}$ and $\vartheta_q q_{i,t} \geq q_{i-1,t}$ (see Appendix B for proof). From past to future, the contribution of $\boldsymbol{v}$ grows when $\vartheta_q < 1$. Correspondingly, we can simply define the total contributions of a sequence with length t in RNN as follows:

$$q_{1,t} + q_{2,t} + ... + q_{t,t} = \sum_{i=0}^{t-1} \vartheta_q^i q_{t,t} \tag{11}$$

For the chunk-enhanced MNR, the contributions of each chunk area can be counted as a separate RNN contribution. Let the lengths of m chunk areas be l_1,l_2,..,l_m, and $T = \sum_{r=1}^{m} l_r$. Each chunk operation integrates l outputs of the controller. Hence, the contributions of m chunk areas can be expressed as follows (see Appendix B for proof).

$$\underbrace{\sum_{i=0}^{l_1-1} \vartheta_q^i \vartheta_p^{T-t_1}, \sum_{i=0}^{l_2-1} \vartheta_q^i \vartheta_p^{T-t_2}, \ ... \ , \sum_{i=0}^{l_m-1} \vartheta_q^i \vartheta_p^{T-t_m}}_{m} \tag{12}$$

For brevity, let $g_r = \sum_{i=0}^{l_r-1} \vartheta_q^i \vartheta_p^{T-t_r}$, where $r \in \{1, 2, .., m\}$. Here g_r contains two parts: (1) $\sum_{i=0}^{l_r-1} \vartheta_q^i$ represents the summation of $\boldsymbol{h}_{l_r}$-based contributions in a subsequence, which is considered as the instantaneous contribution. (2) $\vartheta_p^{T-t_r}$ is the proportion between the hidden states $\boldsymbol{h}_{t_r}$ and $\boldsymbol{h}_{t_m}$, which is long-term dependence. These two parts determine the amount of information stored in the memory slot. In order to maximize the total amount of information stored in all memory slots, we should reduce the gaps among g_1,g_2,... , g_m. In the following, we introduce two segmentation strategies to achieve this goal.

4.4 Chunk Implementation

4.4.1 Periodic Chunk (PEC)

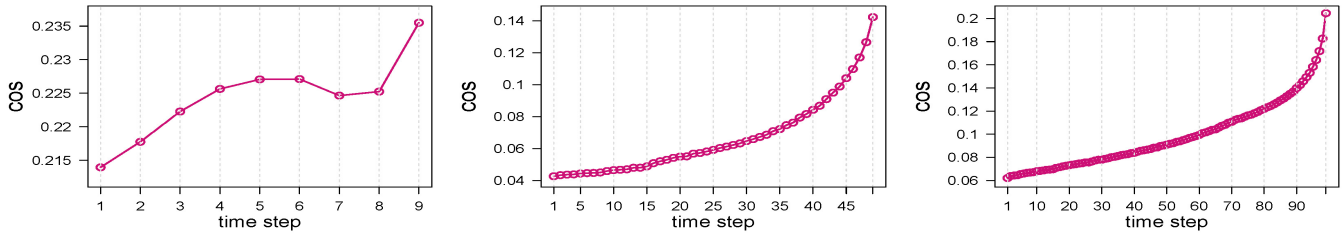
When the change rate of $\boldsymbol{v}$ in the sequence is relatively slow, i.e., the preference transfer of users is not remarkable, we have $\vartheta_p \to 1$ and $\vartheta_q \to 1$. This means the long-term dependencies of each chunk are similar, and the instantaneous contribution increases with the length of the subsequence. Only when all instantaneous contributions are the same (i.e., $l_1 = l_2 = ... = l_m$), the gaps among each chunk contribution are the smallest. Therefore, we propose the periodic chunk (PEC). Given the input sequence $x_{1:T}$ and the chunk cycle $G = \lfloor \frac{T}{m} \rfloor$, the chunk time steps are:

$$\underbrace{T - \ (m-1)\ G, T \ - \ (m-2)\ G, \ ... \ , \ T - 2G, \ T - G, \ T}_{m \ chunk \ time \ steps.} \tag{13}$$

The sequence segmentation results are:

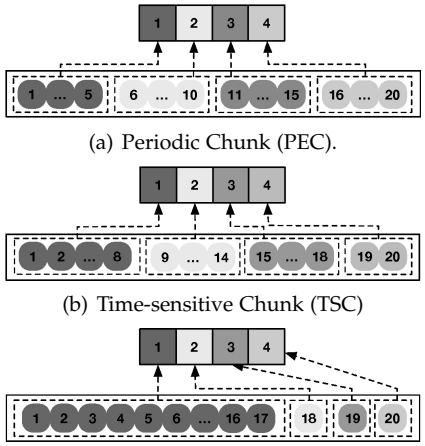
$$\begin{aligned} &x_{1:T \ - \ (m-1)\ G}, \ x_{T \ - \ (m-1)\ G:T \ - \ (m-2)G}, \\ &... \ , \ x_{T \ - \ 2G:T \ - \ G}, \ x_{T \ - \ G:T} \end{aligned} \tag{14}$$

Figure 6 (a) is a graphical illustration of PEC.



(a) weishi (b) ml-10M (c) ml-latest

Fig. 5. Correlation between the target item and other items in the sequence. (a),(b),(c) represent the correlations on three different datasets, the sequence lengths of which are 10, 50, and 100, respectively.



(a) Periodic Chunk (PEC).

(b) Time-sensitive Chunk (TSC)

(c) Extreme Chunk (EXC)

Fig. 6. Chunk rule analysis. Given an input item sequence length of 20, the number of memory slots is 4. (a) shows the periodic chunk with period length of 5 and sequence segmentation $x_{1:5}, x_{6:10}, x_{11:15}, x_{16:20}$, updating memory at time steps 5, 10, 15 and 20. (b) shows time-sensitive chunk, where sequence segmentation and memory update time steps are $x_{1:8}, x_{9:14}, x_{15:18}, x_{19:20}$ and 8, 14, 18, 20 respectively. (c) shows the extreme chunk, where the sequence is divided into $x_{1:17}, x_{18}, x_{19}, x_{20}$, and memory is updated at time steps 17, 18, 19 and 20.

Analysis. In the long run, user preferences always shift, and users usually have different degrees of preference to different items, which means there may be a distribution of user preferences in the sequence. To demonstrate the preference distribution, we investigate the importance of items to the target item in a sequence. In a given sequence, the last item is treated as target item. We calculated the item importance as the correlation between the current item in the sequence and the target item. Specifically, we adopt cosine similarity as the correlation indicator, which is given as follows.

$$cosine(\boldsymbol{v}_i, \boldsymbol{v}_j) = \frac{\boldsymbol{v}_i \cdot \boldsymbol{v}_j}{\|\boldsymbol{v}_i\| \, \|\boldsymbol{v}_j\|} = \frac{\sum_{r=1}^{k} v_{ir} v_{jr}}{\sqrt{\sum_{r=1}^{k} v_{ir}^2} \sqrt{\sum_{r=1}^{k} v_{jr}^2}} \tag{15}$$

We use the item embeddings (trained by the controller) as the input vectors for cosine similarity. Experimental results are shown in Figure 5, the horizontal axis is the position of items in a sequence, and the vertical axis is the correlation between the target item and the current item. As shown, although there are some small fluctuations in Figure 5 (a), the overall trend of the correlation in (b) (c) and (d) is stable and upward. These figures generally indicate that the correlations between a target user interaction and previous interactions increase as the their interval time decrease. That is, the newer the interaction is, the more it reflects future changes of preference in the sequence.

4.4.2 Time-sensitive Chunk (TSC)

Inspired by the above analysis, it is reasonable to propose a time-sensitive chunk strategy where the writing interval is larger at the beginning of the interaction sequence but will be reduced over time (i.e., $l_1 < l_2 < ... < l_m$), so as to enhance the impact of latest user interactions. As the input length between each chunk is $1, 2, .., m-1$ and m, the sum of the input length is $\frac{m(m+1)}{2}$, and proportional step is $g = \left\lfloor \frac{2T}{m(m+1)} \right\rfloor$. Hence, the chunk time steps are given as:

$$\underbrace{T - g\frac{2T}{m(m-1)}, T - g\frac{2T}{(m-1)(m-2)}, ..., T-g, T}_{m \text{ chunk time steps}} \tag{16}$$

The sequence segmentation results are:

$$x_{1:T-g\frac{2T}{m(m-1)}}, x_{T-g\frac{2T}{m(m-1)}:T-g\frac{2T}{(m-1)(m-2)}}, ..., x_{T-g,T} \tag{17}$$

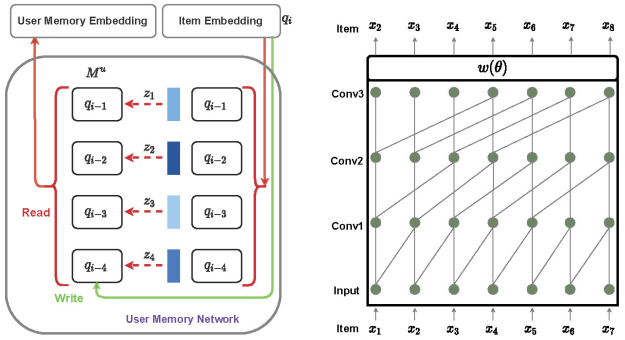
The example of TSC is shown in Figure 6 (b). To obtain the goal of "the newer the interaction is, the greater importance it has for the next prediction", we may need to investigate an extreme case, as shown in Figure 6 (c), referred to as EXC. As can be seen, in EXC the most attention is paid to the contribution of latest interaction, i.e., the first $T-m+1$ items form a large chunk, whereas each of the remaining $m-1$ items is treated as a separate chunk.

4.5 Model Discussion

We conduct model analysis to identify the difference between CmnRec and two related models RUM [11] and NextItNet [4].

4.5.1 Relation with RUM

RUM is considered as the first work that adapts external memory network (EMN) for the recommendation task. By introducing an EMN, its ability to model long sequences has been greatly improved. Figure 7(a) illustrates the structure of EMN. As can be seen, both RUM and CmnRec rely on the reading and writing operations when using and updating memory. However, RUM needs to perform these operations at every time step. Such frequent and continuous



(a) RUM (b) NextItNet

Fig. 7. User memory network of RUM and architecture of NextItNet.

memory operations largely increase the computation costs and cannot robustly process noisy information. By contrast, CmnRec only needs to operate at specific time steps. By caching the input information during non-memory update time and using the attention machine to extract the valuable information during memory update, CmnRec achieves efficient and high-quality recommendation results.

4.5.2 Relation with NextItNet

NextItNet is a classic application of dilated CNNs for the sequential recommendation task. By puncturing regular holes on the convolutional kernel, NextItNet is endowed higher capacity to remember longer sequence without increasing the kernel size or stacking more convolutional layers. The network architecture of NextItNet is illustrated in Figure 7(b) with kernel size of $1*2$. From the memory saving perspective, the idea of CmnRec and NextItNet is similar in some ways since both models perform connection skipping to reduce memory usage. The main difference lies in that NextItNet performs convolutions and memorization with fixed time intervals, while CmnRec performs memorization according to the chunk rules and has no such a restriction as mentioned before. In addition, both the controller and external memory of CmnRec have the memorization capacity, while NextItNet does not have a specific memory container for memorization.

5 Experiments

In this section, we conduct extensive experiments to investigate the efficacy of the chunk-accelerated MNR. Specifically, we aim to answer the following research questions (RQs).

1) **RQ1:** Does chunk speed up MNR significantly? What impacts does the sequence length have on model acceleration?
2) **RQ2:** Does the chunk-accelerated MNR perform comparably with the typical memory-based neural recommendation models in terms of recommendation accuracy?
3) **RQ3:** How does chunk-accelerated MNR perform with TSC, EXC and PEC? Which setting performs best?

TABLE 1
The statistics of the experimental datasets. s: the average length of each sequence. T: the unified sequence length after padding zero.

Dataset	# Interactions	# Sequence	# Item	s	T
weishi	9,986,953	1,048,575	65,997	9.5243	10
ml-10M	7,256,224	178,768	10,670	40.5902	50
ml-latest	25,240,741	300,624	18,226	83.9612	100

TABLE 2
Inference speedup. The values denote multiples. m is slot number.

m	2	3	4	6	9	12	Average
weishi	1.51	1.52	–	–	–	–	1.515
ml-10M	6.28	–	6.71	6.17	4.68	–	5.96
ml-latest	11.35	–	12.16	10.51	8.28	8.03	10.07

5.1 Datasets

We conduct experiments on three real-world recommendation datasets: ml-latest, ml-10M[3] and weishi[4].

ml-latest [36] is a widely used public dataset for both general and sequential recommendations [8], [10], [37], [38]. The original dataset contains 27,753,444 interactions, 283,228 users and 58,098 video clips with timestamps. To reduce the impact of cold items, we filter out videos that appear less than 20 times, and generate a number of sequences, each of which belongs to one user in chronological order. Then, we split the sequence into subsequence every L movies. If the length of the subsequence is less than L, we pad zero in the beginning of the sequence to reach L. For those with length less than l, we simply remove them in our experiments. In our experiments, we set $L = 100$ with $l = 20$.

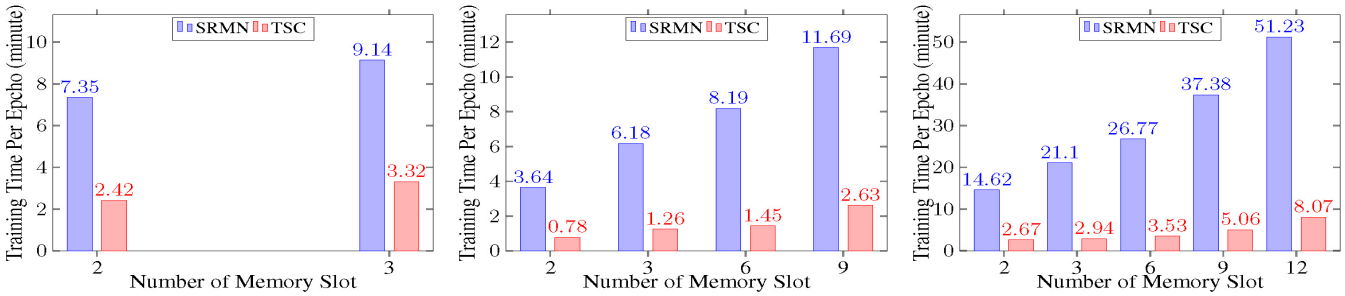
ml-10M contains 10,000,054 interactions, 10,681 movies and 71,567 users. We perform similar pre-processing as ml-latest by setting L to 50 and l to 5.

weishi is a micro-video recommendation dataset collected by the Weishi Group of Tencent. Since both cold users and items have already been trimmed by the official provider, we do not need to perform pre-processing for the cold-start problem. Each user sequence contains 10 items at maximum. The statistics of our datasets after above preprocessing are shown in Table 1.

5.2 Comparative Methods & Evaluation Metrics

GRU4Rec [1]: It is a seminal work that applies the Gated Recurrent Unit (GRU) for sequential recommendation. For a fair comparison, we use the cross-entropy loss function for all neural network models. **LSTM4Rec**: It simply replaces GRU with LSTM since we observe that LSTM generally performs better than GRU for the item recommendation task. **SRMN** [11]: It is a recently proposed sequential recommendation model with external memory network architecture. For comparison purpose, we report results by using LSTM as the controller. In addition, we also compare with two CNN-based sequential recommendation methods: **Caser** [3] and **NextItNet** [4]. As for our proposed methods, we report results with the three chunk variants, i.e., TSC, PEC and EXC. Note that following the original paper of GRU4Rec

3. https://grouplens.org/datasets/movielens/
4. https://www.weishi.com/



(a) weishi (b) ml-10M (c) ml-latest

Fig. 8. Training time of each epoch on the three datasets.

and NextItNet, we conduct all our experiments without learning an explicit user embedding.[5]

Following [3], [4], we use three popular top-N metrics to evaluate the performance of these sequential recommendation models, namely, MRR@N (Mean Reciprocal Rank) [27], HR@N (Hit Ratio) [12] and NDCG@N (Normalized Discounted Cumulative Gain) [39].

5.3 Experiment Setup

To ensure the fairness of the experiment, the dimensions of item embeddings are set to 128 for all neural network models, similar to [3], [4]. We first tune baseline GRU4Rec and LSTM4Rec to optimal performance. Specifically, we set the number of layers of GRU4Rec and LSTM4Rec to 1 and the hidden dimension to 256, which performs better than two hidden layers or a larger hidden dimension. We empirically find that all models except NextItNet benefit from a larger batch size. To make full use of GPU, we set batch size to 1024 for these models. As for NextItNet, we empirically find that it performs best when batch size is between 64 and 256 for all these datasets. We report the results with its best-performing batch size. For SRMN, we set the embedding size of memory slot as 256. The attention dimension b of chunk is 64 on all datasets. Our datasets are randomly divided into training (80%), validation (2%) and testing (18%) sets. All methods are implemented using Tensorflow with Adam [40] as the optimizer. Results are reported when models are converged on the validation test. Our implementation code will be released later.

5.4 Experimental Result and Analysis

5.4.1 Run time (RQ1).

As analyzed before, the chunk framework is theoretically more efficient than SRMN by reducing the number of memory access. To confirm this, we plot the results of the running time of the two methods in Figure 8. It can be seen that the training time of SRMN is several times slower than that of TSC, and the speedup with the maximum memory slot (best accuracy for both SRMN and TSC) on the three datasets are 2.75, 4.44, and 6.34 respectively. We find that the relative improvements are much larger on ml-10M and ml-latest than on weishi. The larger improvements should be attributed to the lengths of the item sequence since for longer sequences, the interval distance between two memory accesses is also larger. Taking the weishi and ml-latest as an example. By setting the number of memory slots as 2, the average interval distance to perform memory access on weishi is 5, while it is 50 on ml-latest. It is also worth noting that the relation between the number of memory slots and the running time is not linear. Increasing the number of memory slots will lead to a decrease of the chunk area, which helps to reduce the computing time of the attention machine. Therefore, the optimal slot number depends on the specific dataset. We also demonstrate the speedup for item generating in Table 2. As shown, similar conclusions also hold to the inference phase.

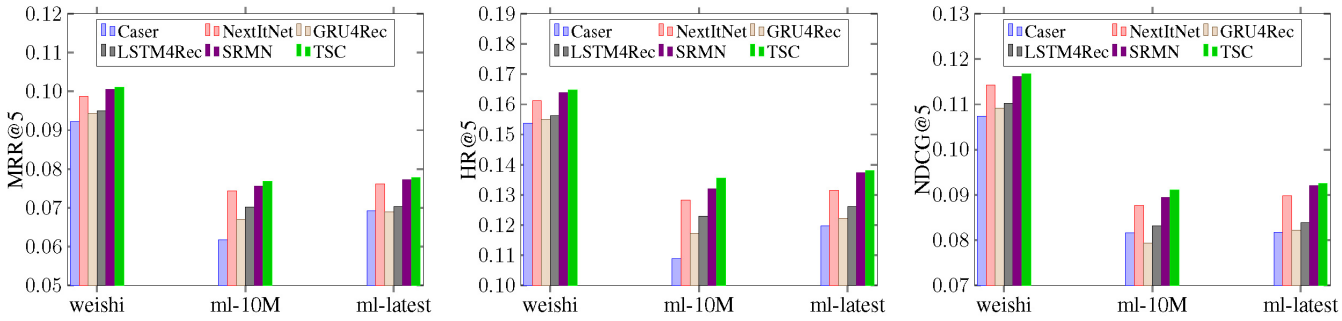
5.4.2 Performance comparison with original SRMN (RQ2)

To verify the effectiveness of the proposed chunk framework, we focus on comparing it with the standard SRMN. We report the recommendation accuracy on Table 3, and all models in the same column share the same hyperparameter Settings. The observations are (1) TSC achieves comparable results with SRMN on all datasets by applying for a relatively large slot number. Both SRMN and our method are senstive to the number of slots — better accuracy is obtained with larger slot number. Particularly, TSC and PEC with 9 memory slots had an even 1.72% performance improvement over SRMN on ml-10M in term of MRR@5. (2) In general, the performance of all chunk-based methods will keep growing by increasing the number of memory slots in the beginning. It then keeps relatively stable once the number of memory slots has been large enough. The optimal number can be achieved by hyperparameter tuning. Empirically, for a sequential dataset with session length longer than 50, we can set the default number to 10, which is a favorable trade-off between the performance and computational cost.

5.4.3 Performance comparison against baselines.

We report the results of all methodologies in Figure 9 and Table 4, 5, and make the following observations. First, the CNN-based model Caser performs worse than GRU4Rec and LSTM4Rec. By contrast, the state-of-the-art temporal CNN model NextItNet yields obviously better results than these baselines. Our findings here are consistent with those in previous works [4], [37]. Third, SRMN and

5. We empirically found that concating a user embedding vector (e.g., in Caser) for sequential recommendation models do not yield any better results. This is probably because the user embedding has already been well represented by the embedding of his interaction sequences, as well illustrated in [10].



(a) MRR@5 (b) HR@5 (c) NDCG@5

Fig. 9. Performance comparisons with respect to top-N values.

TABLE 3
Performance comparison between SRMN and the proposed methods. Bold means the best result, $*$ means the second-best result. m is slot number.

Dataset		weishi		ml-10M				ml-latest				
	m	2	3	2	4	6	9	2	4	6	9	12
MRR@5	SRMN	**0.1001**	0.1005	**0.0739**	**0.0748**	0.0751*	0.0756	**0.0741**	**0.0749**	**0.0755**	**0.0764**	0.0773*
	TSC	0.0978	**0.1010**	0.0727*	0.0724	0.0738	**0.0769**	0.0721*	0.0733	0.0742	0.0755*	**0.0778**
	PEC	0.0985*	0.1006*	0.0721	0.0731*	**0.0753**	0.0768*	0.0717	0.0734*	0.0753*	0.0750	0.0750
	EXC	0.0958	0.0954	0.0651	0.0633	0.0641	0.0673	0.0636	0.0593	0.0622	0.0633	0.0653
HR@5	SRMN	**0.1636**	0.1638	**0.1302**	**0.1316**	0.1318*	0.1320	**0.1317**	**0.1335**	**0.1340**	**0.1359**	0.1374*
	TSC	0.1599	**0.1648**	0.1285*	0.1287	0.1302	**0.1356**	0.1289	0.1318	0.1329	0.1348*	**0.1381**
	PEC	0.1607*	0.1640*	0.1280	0.1305*	**0.1336**	0.1350*	0.1300*	0.1321*	0.1345*	0.1346	0.1347
	EXC	0.1573	0.1567	0.1157	0.1125	0.1137	0.1188	0.1149	0.1088	0.1132	0.1161	0.1190
NDCG@5	SRMN	**0.1158**	0.1161	**0.0878**	**0.0888**	0.0890*	0.0895	**0.0883**	**0.0893**	**0.0899**	**0.0911**	0.0921*
	TSC	0.1131	**0.1168**	0.0864*	0.0863	0.0876	**0.0911**	0.0860	0.0875	0.0882	0.0901*	**0.0925**
	PEC	0.1139*	0.1162*	0.0857	0.0871*	**0.0895**	0.0909*	0.0860*	0.0878*	0.0897*	0.0897	0.0896
	EXC	0.1110	0.1105	0.0774	0.0754	0.0763	0.0800	0.0758	0.0713	0.0746	0.0762	0.0784

TABLE 4
Performance comparisons against baseline with respect to top-5 values. ↑ means the percentage of TSC's performance over baseline methods

Dataset		weishi	ml-10M	ml-latest
MRR@5	Caser	0.0922(↑ 8.71%)	0.0618(↑ 19.63%)	0.0602(↑ 22.62%)
	NextItNet	0.0987(↑ 2.28%)	0.0743(↑ 3.34%)	0.0761(↑ 2.19%)
	GRU4Rec	0.0943(↑ 6.63%)	0.0670(↑ 12.87%)	0.0690(↑ 11.31%)
	LSTM4Rec	0.0950(↑ 5.94%)	0.0702(↑ 8.71%)	0.0703(↑ 9.64%)
	SRMN	0.1005(↑ 0.50%)	0.0756(↑ 1.69%)	0.0773(↑ 0.64%)
	TSC	**0.1010**	**0.0769**	**0.0778**
Hit@5	Caser	0.1538(↑ 6.67%)	0.1089(↑ 19.69%)	0.1198(↑ 13.25%)
	NextItNet	0.1613(↑ 2.12%)	0.1283(↑ 5.38%)	0.1314(↑ 4.85%)
	GRU4Rec	0.1550(↑ 5.59%)	0.1172(↑ 13.57%)	0.1223(↑ 11.44%)
	LSTM4Rec	0.1562(↑ 5.22%)	0.1230(↑ 9.29%)	0.1261(↑ 8.69%)
	SRMN	0.1638(↑ 0.61%)	0.1320(↑ 2.65%)	0.1374(↑ 0.51%)
	TSC	**0.1648**	**0.1356**	**0.1381**
NDCG@5	Caser	0.1074(↑ 8.05%)	0.0816(↑ 10.43%)	0.0817(↑ 11.68%)
	NextItNet	0.1142(↑ 2.23%)	0.0877(↑ 3.73%)	0.0898(↑ 2.92%)
	GRU4Rec	0.1092(↑ 6.51%)	0.0793(↑ 12.95%)	0.0822(↑ 11.14%)
	LSTM4Rec	0.1102(↑ 5.65%)	0.0832(↑ 8.67%)	0.0839(↑ 9.30%)
	SRMN	0.1161(↑ 0.60%)	0.0895(↑ 1.67%)	0.0921(↑ 0.43%)
	TSC	**0.1168**	**0.0911**	**0.0925**

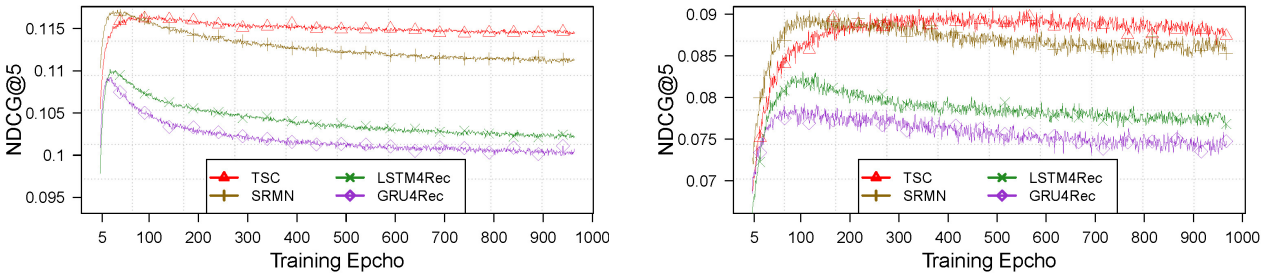
TABLE 5
Performance comparisons against baseline with respect to top-20 values.

Dataset		ml-10M	ml-latest
MRR@20	Caser	0.0786(↑ 16.03%)	0.0755(↑ 23.44%)
	NextItNet	0.0879(↑ 3.75%)	0.0898(↑ 3.79%)
	GRU4Rec	0.0799(↑ 14.14%)	0.083(↑ 12.29%)
	LSTM4Rec	0.0842(↑ 8.31%)	0.0845(↑ 10.29%)
	SRMN	0.0905(↑ 0.77%)	0.0925(↑ 0.75%)
	TSC	**0.0912**	**0.0932**
Hit@20	Caser	0.2373(↑ 22.37%)	0.2436(↑ 22.53%)
	NextItNet	0.2713(↑ 7.04%)	0.275(↑ 8.54%)
	GRU4Rec	0.2559(↑ 13.48%)	0.2722(↑ 9.66%)
	LSTM4Rec	0.272(↑ 6.76%)	0.278(↑ 7.37%)
	SRMN	0.2829(↑ 2.65%)	0.2945(↑ 1.35%)
	TSC	**0.2904**	**0.2985**
NDCG@20	Caser	0.1102(↑ 21.76%)	0.1127(↑ 22.11%)
	NextItNet	0.1279(↑ 4.92%)	0.1301(↑ 5.79%)
	GRU4Rec	0.1179(↑ 13.81%)	0.1238(↑11.21%)
	LSTM4Rec	0.1247(↑ 7.63%)	0.1261(↑ 9.13%)
	SRMN	0.1303(↑ 2.98%)	0.1363(↑ 1.02%)
	TSC	**0.1342**	**0.1377**

TSE outperform all other baselines, which demonstrates the effectiveness of memory-based neural networks. Finally, TSC outperforms SRMN in all data sets, which proves the rationality of chunk policy.

5.4.4 Denoising

We plot the convergence behaviors of GRU4Rec, LSTM4Rec, SRMN and TSC in Figure 10. As shown, memory-based recommendation models (i.e., SRMN and TSC) have apparent advantages over the RNN models in terms of both accuracy and robustness. We believe the external memory network can enhance the storage and the capacity of information processing of RNN so as to improve accuracy. In addition, abnormal input data or noise usually leads to overfitting after convergence. However, since the external storage network maintains more information than the recurrent unit,



(a) weishi (b) ml-10M

Fig. 10. Convergence behaviors in terms of NDCG@5. The number of memory slots of TSC and SRMN are 3 and 9 respectively on the two datasets.

the impacts of abnormal data from a small number of instances can be restricted to a certain extent. Furthermore, we observe that TSC is even more robust than SRMN, and the results on both weishi and ml-10M imply that TSC can effectively prevent the overfitting problem. We argue that the memory update mechanism in TSC makes it insensitive to noise since it takes into account data obtained from previous timesteps rather than that from only the current timestep.

5.4.5 Performance comparison of TSC, PEC and EXC (RQ3)

Since we have introduced three chunk variants, i.e., TSC, PEC and EXC, we report their results on Table 3 for a clear comparison. First, we observe that PEC and TSC perform much better than EXC on all datasets. In fact, EXC performs even worse than the baseline models. We suspect that this is because EXC mainly focuses on modeling the most recent interactions, ignoring earlier interactions which however make up the vast majority of the interaction sequence. That is, the extreme partitioning cannot offer satisfied performance in practice. Second, TSC achieves better results than PEC in terms of all evaluation metrics when setting a large slot number. This implies that the time-sensitive chunk strategy is better suited to balance long short-term sequential relations than the periodic setup.

6 Conclusion

In this paper, we have introduced a novel sequential recommendation framework by combining the Chunk and External Memory Network (EMN). The motivation is that the way of memory access operations in the existing EMN introduces redundant computation, which results in very high time complexity when modeling long-range user session data. A Chunk-accelerated memory network is proposed with two practical implementations: periodic chunk (PEC) and time-sensitive chunk (TSC). We demonstrate that our proposed chunk framework significantly reduces the computation time of memory-based sequential recommendation models but achieves competitive recommendation results.

7 Conclusion

The conclusion goes here.

Appendix A
A Rough Complexity Analysis

MNR consists of a controller and an external memory network (EMN). For easier illustration, the controller and EMN are often realized by RNNs and DNC. And the time complexity of them is $\mathcal{O}(h^2 + kh)$ and $\mathcal{O}(4h^2)$ [14], respectively (k is item embedding size, and h is hidden state size). Based on the time complexity of the controller and EMN, the total time consumption of MNR and CmnRec is $(5h^2+kh)*T$ and $(h^2+kh)*T+4mh^2$, respectively. When $h = 2k$, the time consumption ratio of MNR and CmnRec is $1 \leq \frac{8}{3+8\frac{m}{T}} \leq \frac{8}{3}$. In practice, a larger acceleration can be obtained due to there are many other complex operations when EMN processes memory operations, such as memory addressing operations [14].

Appendix B
Find Upper Bound

The update equation of standard RNN hidden state is:

$$\boldsymbol{h}_t = tanh(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{v}_t + \boldsymbol{b}) \tag{18}$$

Starting from the derivative of $\boldsymbol{v}_t$, we make use of the result $\vartheta_q = B\|\boldsymbol{U}\|$ which is shown in [34], where B is the bound of $\left\|diag(tanh^{'}(\boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{W}\boldsymbol{v}_t + \boldsymbol{b}))\right\|$. Then we take the derivative of $\boldsymbol{h}_{t-1}$. In Eq.(18), $\boldsymbol{U}\boldsymbol{h}_{t-1}$ and $\boldsymbol{W}\boldsymbol{v}_t$ are interchangeable. Referring to the solution of ϑ_q, we replace $\boldsymbol{W}$ with $\boldsymbol{U}$, and $\boldsymbol{v}_t$ with $\boldsymbol{h}_{t-1}$. As a result, we have $\vartheta_p = B\|\boldsymbol{W}\|$.

For LSTM, the update equation is:

$$\begin{aligned}\boldsymbol{c}_t &= \sigma(\boldsymbol{U}_f\boldsymbol{h}_{t-1} + \boldsymbol{W}_f\boldsymbol{v}_t + \boldsymbol{b}_f) \odot \boldsymbol{c}_{t-1} \\ &\quad + \sigma(\boldsymbol{U}_i\boldsymbol{h}_{t-1} + \boldsymbol{W}_i\boldsymbol{v}_t + \boldsymbol{b}_i) \odot tanh(\boldsymbol{U}_z\boldsymbol{h}_{t-1} + \boldsymbol{W}_z\boldsymbol{v}_t + \boldsymbol{b}_z) \\ \boldsymbol{h}_t &= tanh(\boldsymbol{U}_o\boldsymbol{h}_{t-1} + \boldsymbol{W}_o\boldsymbol{v}_t + \boldsymbol{b}_o) \odot tanh(\boldsymbol{c}_t)\end{aligned} \tag{19}$$

Following the same way, we start from the derivative of $\boldsymbol{v}_t$. According to the results in [34], $\vartheta_q > 0$ is equivalent to $\vartheta_q q_{i,t} \geq q_{i-1,t}$. To solve the problem of ϑ_p, we define:

$$\Phi_\varrho(\boldsymbol{h}_{t-1}, \boldsymbol{v}_t, \boldsymbol{U}_\varrho, \boldsymbol{W}_\varrho, \boldsymbol{b}_\varrho) = \begin{cases} \boldsymbol{U}_f\boldsymbol{h}_{t-1} + \boldsymbol{W}_f\boldsymbol{v}_t + \boldsymbol{b}_f, \\ \boldsymbol{U}_i\boldsymbol{h}_{t-1} + \boldsymbol{W}_i\boldsymbol{v}_t + \boldsymbol{b}_i, \\ \boldsymbol{U}_z\boldsymbol{h}_{t-1} + \boldsymbol{W}_z\boldsymbol{v}_t + \boldsymbol{b}_z, \\ \boldsymbol{U}_o\boldsymbol{h}_{t-1} + \boldsymbol{W}_o\boldsymbol{v}_t + \boldsymbol{b}_o. \end{cases}$$

where $\varrho \in \{f, i, o, z\}$. In Eq.(19), we swap $\boldsymbol{U}_\varrho\boldsymbol{h}_{t-1}$ and $\boldsymbol{W}_\varrho\boldsymbol{v}_t$. As a result, $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$ remain unchanged, which

means $\boldsymbol{h}_{t}, \boldsymbol{U}_{\varrho}\boldsymbol{h}_{t-1}$ and $\boldsymbol{W}_{\varrho}\boldsymbol{v}_{t}$ are equivalent. Following the solution of ϑ_q in standard RNN, we replace $\boldsymbol{W}_{\varrho}$ with $\boldsymbol{U}_{\varrho}$, and $\boldsymbol{v}_t$ with $\boldsymbol{h}_{t-1}$. As a result, we have $\vartheta_p > 0$ to guarantee $\vartheta_p p_{i,t} \geq p_{i-1,t}$.

APPENDIX C
THE CONTRIBUTION OF MEMORY SLOTS

Accroding to Eq.(11) , the contribution of each slot can be measured as $\sum_{i=0}^{l_r-1} \vartheta_q^i q_{t_r,t_r}, (r = \{1, 2, 3...m\})$, i.e.,

$$\begin{aligned} q_{t_r,t_m} &= \left\| \frac{\partial \boldsymbol{h}_{t_m}}{\partial \boldsymbol{v}_{t_r}} \right\| = \left\| \frac{\partial \boldsymbol{h}_{t_m}}{\partial \boldsymbol{h}_{t_r}} \frac{\partial \boldsymbol{h}_{t_r}}{\partial \boldsymbol{v}_{t_r}} \right\| \\ &= p_{t_r,t_m} q_{t_r,t_r} \leq \vartheta_p^{t_m - t_r} p_{t_m,t_m} q_{t_r,t_r} \\ &= \vartheta_p^{T-t_r} p_{T,T} q_{t_r,t_r} \qquad (t_m = T) \end{aligned} \tag{20}$$

For the sake of analysis, assuming the contribution of $\boldsymbol{v}$ and $\boldsymbol{h}_t$ at time step t is constant, i.e., $q_{t_1,t_1} = q_{t_2,t_2} = ... = q_{t_m,t_m}$. Then we can get $\sum_{i=0}^{l_r-1} \vartheta_q^i q_{t_r,t_r} = \sum_{i=0}^{l_m-1} \vartheta_q^i \vartheta_p^{T-t_m} p_{T,T} q_{t_r,t_r}$. Considering that $p_{T,T} q_{t_r,t_r}$ is same for every r, $\sum_{i=0}^{l_m-1} \vartheta_q^i \vartheta_p^{T-t_m} p_{T,T} q_{t_r,t_r}$ can be simplified to $\sum_{i=0}^{l_m-1} \vartheta_q^i \vartheta_p^{T-t_m}$.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China under Grant (No. 62032013, 61972078, 61602197, L1924068), Hebei Natural Science Foundation No.G2021203010 and CCF-AFSG Research Fund under Grant No.RF20210005.

REFERENCES

[1] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *ICLR*. San Juan, Puerto Rico: OpenReview.net, 2016.

[2] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi, "Personalizing session-based recommendations with hierarchical recurrent neural networks," in *RecSys*. Como, Italy: ACM, 2017, pp. 130–137.

[3] J. Tang and K. Wang, "Personalized top-n sequential recommendation via convolutional sequence embedding," in *WSDM*. New York, NY, USA: ACM, 2018, pp. 565–573.

[4] F. Yuan, A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He, "A simple convolutional generative network for next item recommendation," in *WSDM*. New York, NY, USA: ACM, 2019, pp. 582–590.

[5] C. Ma, P. Kang, and X. Liu, "Hierarchical gating networks for sequential recommendation," in *KDD*. New York, NY, USA: ACM, 2019, pp. 825—-833.

[6] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] Y. Fajie, H. Xiangnan, G. Guibing, X. Zhezhao, X. Jian, and H. Xiuqiang, "Modeling the past and future contexts for session-based recommendation," 2019.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*. Long Beach, CA, USA: ACM, 2017, pp. 5998–6008.

[10] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in *ICDM*. Singapore, Singapore: IEEE, 2018, pp. 197–206.

[11] X. Chen, H. Xu, Y. Zhang, J. Tang, Y. Cao, Z. Qin, and H. Zha, "Sequential recommendation with user memory networks," in *WSDM*. New York, NY, USA: ACM, 2018, pp. 108–116.

[12] Q. Wang, H. Yin, Z. Hu, D. Lian, H. Wang, and Z. Huang, "Neural memory streaming recommender networks with adversarial training," in *KDD*. London, UK: ACM, 2018, pp. 2467–2475.

[13] S. Sukhbaatar, a. szlam, J. Weston, and R. Fergus, "End-to-end memory networks," in *NIPS*. Montreal, Quebec, Canada: Curran Associates, Inc., 2015, pp. 2440–2448.

[14] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, p. 471, 2016.

[15] J. Johnson, B. Hariharan, L. Maaten, J. Hoffman, L. Fei-Fei, C. Zitnick, and R. Girshick, "Inferring and executing programs for visual reasoning," in *ICCV*. Los Alamitos, CA, USA: IEEE Computer Society, 2017, pp. 3008–3017.

[16] M. J. Seo, S. Min, A. Farhadi, and H. Hajishirzi, "Query-reduction networks for question answering," in *ICLR*. San Juan, Puerto Rico: OpenReview.net, 2016.

[17] J. Cai, R. Shin, and D. Song, "Making neural programming architectures generalize via recursion," in *ICLR*. New Orleans, Louisiana, USA: OpenReview.net, 2017, pp. 108–116.

[18] T. Ebesu, B. Shen, and Y. Fang, "Collaborative memory network for recommendation systems," in *SIGIR*. New York, NY, USA: ACM, 2018, pp. 515–524.

[19] J. Huang, W. X. Zhao, H. Dou, J.-R. Wen, and E. Y. Chang, "Improving sequential recommendation with knowledge-enhanced memory networks," in *SIGIR*. New York, NY, USA: ACM, 2018, pp. 505–514.

[20] R. J. Gerrig, *Psychology and life*, 20th ed. One Lake Street, Upper Saddle River, NJ, USA: Pearson Education, 2013, (book).

[21] S. Wang, L. Cao, and Y. Wang, "A survey on session-based recommender systems," *CoRR*, vol. abs/1902.04864, 2019.

[22] G. Shani, D. Heckerman, and R. I. Brafman, "An mdp-based recommender system," *Journal of Machine Learning Research*, vol. 6, no. Sep, pp. 1265–1295, 2005.

[23] R. He and J. McAuley, "Fusing similarity models with markov chains for sparse sequential recommendation," in *ICDM*. Barcelona, Spain: IEEE, 2016, pp. 191–200.

[24] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *WWW*. Raleigh, North Carolina: ACM, 2010, pp. 811–820.

[25] F. Yuan, X. Xin, X. He, G. Guo, W. Zhang, C. Tat-Seng, and J. M. Jose, "fbgd: Learning embeddings from positive unlabeled data with bgd," in *UAI*. Monterey, California, USA: AUAI Press, 2018.

[26] F. Yuan, G. Guo, J. M. Jose, L. Chen, H. Yu, and W. Zhang, "Lambdafm: learning optimal ranking with factorization machines using lambda surrogates," in *CIKM*. New York, NY, USA: ACM, 2016, pp. 227–236.

[27] B. Hidasi and A. Karatzoglou, "Recurrent neural networks with top-k gains for session-based recommendations," in *CIKM*. New York, NY, USA: ACM, 2018, p. 843–852.

[28] Y. K. Tan, X. Xu, and Y. Liu, "Improved recurrent neural networks for session-based recommendations," in *DLRS*. Boston, MA, USA: ACM, 2016, pp. 17–22.

[29] Y. Gu, T. Lei, R. Barzilay, and T. Jaakkola, "Learning to refine text based recommendations," in *EMNLP*. Austin, Texas: ACL, 2016, pp. 2103–2108.

[30] E. Smirnova and F. Vasile, "Contextual sequence modeling for recommendation with recurrent neural networks," in *DLRS*. Como, Italy: ACM, 2017, pp. 2–9.

[31] E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom, "Learning to transduce with unbounded memory," in *NIPS*. Cambridge, MA, USA: MIT Press, 2015, pp. 1828–1836.

[32] A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston, "Key-value memory networks for directly reading documents," in *EMNLP*. Austin, Texas: ACL, 2016, pp. 1400–1409.

[33] J. Zhang, X. Shi, I. King, and D.-Y. Yeung, "Dynamic key-value memory networks for knowledge tracing," in *WWW*. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 765–774.

[34] H. Le, T. Tran, and S. Venkatesh, "Learning to remember more with less memorization," in *ICLR*. New Orleans, LA, USA: OpenReview.net, 2019.

[35] D. Lian, H. Wang, Z. Liu, J. Lian, E. Chen, and X. Xie, "Lightrec: A memory and search-efficient recommender system," in *WWW '20*. Taipei, Taiwan: ACM, 2020, p. 695–705.

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

[37] J. Tang, F. Belletti, S. Jain, M. Chen, A. Beutel, C. Xu, and E. H Chi, "Towards neural mixture recommender for long range dependent user sequences," in *WWW*. San Francisco, CA, USA: ACM, 2019, pp. 1782–1793.
[38] J. Wang, Q. Liu, Z. Liu, and S. Wu, "Towards accurate and interpretable sequential prediction: A cnn & attention-based feature extractor," in *CIKM*. Beijing, China: ACM, 2019, pp. 1703–1712.
[39] W. Guo, S. Wu, L. Wang, and T. Tan, "Personalized ranking with pairwise factorization machines," *Neurocomput*, vol. 214, no. C, pp. 191–200, 2016.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*. San Diego, CA, USA: OpenReview.net, 2015.

Shilin Qu received the BS and MS degrees in software engineering from Northeastern University, China, in 2017 and 2020 respectively. He is currently working toward a PhD degree in the Faculty of Information Technology, Monash University, Australia. His research interests are in the area of graph representation learning and recommender system.

Fajie Yuan is currently an assistant professor at Westlake University. Prior to that, he was a senior AI researcher at Tencent working on recommender systems and machine learning. He received his Ph.D. degree from the University of Glasgow. His main research interests include deep learning and transfer learning and their applications in recommender systems.

Guibing Guo is currently a Professor with the Software College, Northeastern University, Shenyang, China. He received the Ph.D. degree in computer science from Nanyang Technological University, Singapore, in 2015. His research interests include recommender systems, deep learning, natural language processing, and data mining.

Liguang Zhang is a technician at Tencent. His mainly responsibility is business recommendation of KANDIAN.

Wei Wei received the Ph.D. degree from the Huazhong University of Science and Technology, China, in 2012. He is currently an Associate Professor with School of Computer Science and Technology and the Director of Cognitive Computing and Intelligent Information Processing (CCIIP) Laboratory in Huazhong University of Science and Technology, China. His major research interests include information retrieval, natural language processing, artificial intelligence, data mining (text mining), statistics machine learning, social media analysis and mining recommender system.