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 can watch hundreds of micro-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.

1. https://www.tiktok.com/en/
to improve the accuracy of existing recurrent models [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 Fig. 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.
- We present two effective implementations for CmnRec: periodic chunk (PEC) considering the input of each time to be equally important and time-sensitive chunk (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 $I = \{1\}$ and $S = |S|$ as the size of item and sequence sets. The corresponding item embedding vectors are $\{v_1, v_2, v_3, \ldots, v_T\}$.

Fig. 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. 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 $M \in \mathbb{R}^{m \times n}$, where $m$ is the number of memory slots and $n$ is the embedding size of memory slot.

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

$$f_i = \text{softmax}(\Gamma_f(o_i, r_i))$$

$$\hat{x}_i = \text{maxID}(f_i)$$

where $\Gamma_f(\cdot, \cdot)$ is a feedforward operation that performs a linear transformation of the final hidden layer and returns a feature vector $f_i$ as the output. $\text{maxID}(f_i)$ 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 $h_{i-1} \in \mathbb{R}^h$. The memory writing operation $\Gamma_w(\cdot, \cdot)$ and reading operation $\Gamma_r(\cdot, \cdot)$ can be represented as Eqs. (3) and (4):

$$M_i = \Gamma_w(o_i, M_{i-1})$$

$$r_i = \Gamma_r(o_i, M_i)$$

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

$$o_i = \Gamma_o(r_{i-1}, h_{i-1}, v_i)$$

$$h_i = \Gamma_h(r_{i-1}, h_{i-1}, v_i)$$
Normally, $\Gamma_o$ and $\Gamma_h$ are implemented as RNNs. As for $\Gamma_w$ and $\Gamma_r$, 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 $h$ corresponding to these $m$ subsequences will be chunked $m$ times.

In our chunk framework, the controller output $o$ 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 $C \in \mathbb{R}^{l \times h}$ to store $h$ temporarily. During non-chunk time, $C$ caches $h$. For every $h$ that $C$ caches, $l$ increases by 1. When the chunk time arrives, the attention machine converts $C$ to a new controller hidden state $a \in \mathbb{R}^h$ 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:

\[
C = \text{concat}(h_{t_1-1}, h_{t_1-2}, ..., h_{t_1-1}, h_1)
\]

\[
z_{ij} = \tanh(Wc_j + Ur_{t-1})
\]

\[
a_{ij} = \text{softmax}(z_{ij})
\]

\[
a_i = \sum_{j=1}^{l} a_{ij}c_j
\]

where $c_j$ is the $j$-th element of $C$, $a_{ij}$ is the attention score of $c_j$ at $i$-th time step, and $r_{t-1}$ is the read vector at time step $i - 1$. $W \in \mathbb{R}^{l \times h}$, $U \in \mathbb{R}^{l \times h}$ and $w \in \mathbb{R}^h$ are parameters, where $b$ is attention dimension. And Fig. 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 $C$, a memory slot number $m$.

**Output:** The predicted sequential item IDs $x_{T}$

1: Generates chunk time steps set $\text{chunk}$ using Eq.(16);
2: for $i = 1; i < T - 1; i + +$ do
3: $C.add(h_i)$;
4: // The operation of the controller and memory in the chunk moment.
5: if ($i \in \text{chunk}$) then
6: Use Eqs. (8) (9) and (10) to calculate $a_i$;
7: Perform Eq.(5): $o_i = \Gamma_o(r_{i-1}, a_i, v_i)$;
8: Perform Eq.(6): $h_i = \Gamma_h(r_{i-1}, a_i, v_i)$;
9: Perform Eq.(3): $M_i = \Gamma_w(o_i, M_{i-1})$;
10: Perform Eq.(4): $r_i = \Gamma_r(o_i, M_i)$;
11: $C.empty()$
12: // The operation of the controller in the non-chunk moment.
13: else
14: Perform Eq.(5): $o_i = \Gamma_o(r_{i-1}, h_{i-1}, v_i)$;
15: Perform Eq.(6): $h_i = \Gamma_h(r_{i-1}, h_{i-1}, v_i)$;
16: $r_i = r_{i-1}$;
17: // Predict the item ID with the highest probability.
18: Perform Eq.(1): $f_i = \text{softmax}(\Gamma_f(o_i, r_i))$;
19: Perform Eq.(2): $\hat{x}_i = \max ID(f_i)$

#### 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 $h_i = \Psi(h_{i-1}, v_i)$ 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 $\frac{\partial h_i}{\partial h_{i-1}}$ and $\frac{\partial h_i}{\partial v_i}$ to represent the contributions of $h_{i-1}$ and $v_i$ to $h_i$. 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 $h_i$ and $v_i$ (to $h_j$) by $\frac{\partial h_i}{\partial h_j}$ and $\frac{\partial h_j}{\partial v_i}$. Tersely, let $p_{i,t}$ and $q_{i,t}$ denote these two terms. For standard RNNs, there must be values $\partial_{p_i}h_i, \partial_{v_i}h_i \in \mathbb{R}^+$ that satisfy $\partial_{p_i}h_i \geq p_{i-1,t}$ and $\partial_{q_i}h_i \geq q_{i-1,t}$ (see Appendix B for proof). From past to future, the contribution of $v$ grows when $\partial_{v_i} < 1$. Correspondingly, we can simply define the total contributions of a sequence with length $l$ in RNN as follows:

\[
q_{1,t} + q_{2,t} + ... + q_{l,t} = \sum_{i=0}^{l-1} \partial_{q} q_{l,t}
\]
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 \( l_1, l_2, \ldots, l_m \), 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):

\[
\sum_{i=0}^{l_t-1} \theta_{i}^{T-t_1} \sum_{i=0}^{l_t-1} \theta_{i}^{T-t_2}, \ldots, \sum_{i=0}^{l_t-1} \theta_{i}^{T-t_m}
\]

For brevity, let \( g_r = \sum_{i=0}^{l_t-1} \theta_{i}^{T-t_r} \), where \( r \in \{1,2,\ldots,m\} \). Here, \( g_r \) contains two parts: (1) \( \sum_{i=0}^{l_t-1} \theta_{i}^{T-t_r} \) represents the summation of \( h_i \)-based contributions in a subsequence, which is considered as the instantaneous contribution. (2) \( \theta_{i}^{T-t_r} \) is the proportion between the hidden states \( h_i \) and \( h_{t_r} \), 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 \( g_1, g_2, \ldots, g_m \). 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 \( v \) in the sequence is relatively slow, i.e., the preference transfer of users is not remarkable, we have \( \theta_p \rightarrow 1 \) and \( \theta_q \rightarrow 1 \). 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., \( l_1 = l_2 = \ldots = l_m \)), 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 \( G = \left\lceil \frac{T}{m} \right\rceil \), the chunk time steps are:

\[
T - (m - 1) G, T - (m - 2) G, \ldots, T - 2G, T - G, T
\]

The sequence segmentation results are:

\[
x_{1:T} - (m - 1) G, x_{T} - (m - 1) G / T - (m - 2) G, \ldots, x_{T} - 2G / T - G, x_{T} - G / T
\]

Fig. 6a is a graphical illustration of PEC.

**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.

\[
\text{cosine}(v_i, v_j) = \frac{v_i \cdot v_j}{\|v_i\| \|v_j\|} = \frac{\sum_{r=1}^{k} v_{i_r} v_{j_r}}{\sqrt{\sum_{r=1}^{k} v_{i_r}^2} \sqrt{\sum_{r=1}^{k} v_{j_r}^2}}
\]

We use the item embeddings (trained by the controller) as the input vectors for cosine similarity. Experimental results...
are shown in Fig. 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 Fig. 5a, 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., $l_1 < l_2 < \ldots < l_m$), so as to enhance the impact of latest user interactions. As the input length between each chunk is $1, 2, \ldots, m - 1$ and $m$, the sum of the input length is $\frac{m(m+1)}{2}$, and proportional step is $g = \frac{2T}{m(m+1)}$. Hence, the chunk time steps are given as

$$T - g \frac{2T}{m(m-1)}, T - g \frac{2T}{m(m-2)}, \ldots, T - g \frac{2T}{m chunk time steps}$$

(16)

The sequence segmentation results are

$$x_1, T - g \frac{2T}{m(m-1)}, x_T - g \frac{2T}{m(m-1)}, x_T - g \frac{2T}{m(m-2)}, \ldots, x_T - g T$$

(17)

The example of TSC is shown in Fig. 6b. 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 Fig. 6c, referred to as EXC. As can be seen, in EXC the most attention is paid to the contributions 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. 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 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 with higher capacity to remember longer sequence without increasing the kernel size or stacking more convolutional layers. The network architecture of NextItNet is illustrated in Fig. 7b with kernel size of $1 \times 2$. 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?

5.1 Datasets

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

- **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 subsequences every L. For those with length less than L, we pad zero in the beginning of the sequence to reach L. For those with length more 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.

Table 1: The Statistics of the Experimental Datasets

| 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|

s: the average length of each sequence. T: the unified sequence length after padding zero.

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 is 64 on all datasets. Our datasets are randomly divided into

![User Memory Network of RUM and Architecture of NextItNet.](image)

**Fig. 7.** User memory network of RUM and architecture of NextItNet.
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 Fig. 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 sensitive 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 larger 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 Fig. 9 and Tables 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 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 Fig. 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, 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

### TABLE 2

| Inference Speedup | 2 | 3 | 4 | 6 | 9 | 12 | Average |
|-------------------|---|---|---|---|---|----|---------|
| m                 |   |   |   |   |   |    |         |
| 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   |

The values denote multiples. m is slot number.

Fig. 8. Training time of each epoch on the three datasets.
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 perfor-
mance in practice. Second, TSC achieves better results than

![Fig. 9. Performance comparisons with respect to top-N values.](image)

TABLE 3

| Dataset | m | weishi | ml-10M | ml-latest |
|---------|---|--------|--------|-----------|
| MRR@5   | 2 | 0.1001 | 0.0739 | 0.0748 |
|         | 3 | 0.0737 | 0.0724 | 0.0738 |
| TSC     | 4 | 0.0751* | 0.0756* | 0.0756 |
| PEC     | 5 | 0.0721* | 0.0733* | 0.0742* |
| EXC     | 6 | 0.0753* | 0.0769* | 0.0755 |
|         | 7 | 0.0708* | 0.0717 | 0.0734* |
|         | 8 | 0.0641 | 0.0673* | 0.0753* |
|         | 9 | 0.0636 | 0.0636 | 0.0662 |
|         | 10| 0.0653 |

**TABLE 4**

| Dataset | m | weishi | ml-10M | ml-latest |
|---------|---|--------|--------|-----------|
| MRR@5   | 2 | 0.0750 | 0.0712 | 0.0721 |
|         | 3 | 0.0733 | 0.0738 | 0.0742 |
| TSC     | 4 | 0.0755* | 0.0755* | 0.0755 |
| PEC     | 5 | 0.0703 | 0.0717 | 0.0734 |
| EXC     | 6 | 0.0753* | 0.0769* | 0.0755 |
|         | 7 | 0.0708* | 0.0717 | 0.0734 |
|         | 8 | 0.0641 | 0.0673* | 0.0753* |
|         | 9 | 0.0636 | 0.0636 | 0.0662 |
|         | 10| 0.0653 |

**TABLE 5**

| Dataset | m | weishi | ml-10M | ml-latest |
|---------|---|--------|--------|-----------|
| MRR@5   | 2 | 0.0750 | 0.0712 | 0.0721 |
|         | 3 | 0.0733 | 0.0738 | 0.0742 |
| TSC     | 4 | 0.0755* | 0.0755* | 0.0755 |
| PEC     | 5 | 0.0703 | 0.0717 | 0.0734 |
| EXC     | 6 | 0.0753* | 0.0769* | 0.0755 |
|         | 7 | 0.0708* | 0.0717 | 0.0734 |
|         | 8 | 0.0641 | 0.0673* | 0.0753* |
|         | 9 | 0.0636 | 0.0636 | 0.0662 |
|         | 10| 0.0653 |

**Bold** means the best result, **m** means the second-best result, **n** is slot number.

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 perfor-
mance in practice. Second, TSC achieves better results than

![TABLE 4](image)

**TABLE 4**

| Dataset | m | weishi | ml-10M | ml-latest |
|---------|---|--------|--------|-----------|
| MRR@5   | 2 | 0.0750 | 0.0712 | 0.0721 |
|         | 3 | 0.0733 | 0.0738 | 0.0742 |
| TSC     | 4 | 0.0755* | 0.0755* | 0.0755 |
| PEC     | 5 | 0.0703 | 0.0717 | 0.0734 |
| EXC     | 6 | 0.0753* | 0.0769* | 0.0755 |
|         | 7 | 0.0708* | 0.0717 | 0.0734 |
|         | 8 | 0.0641 | 0.0673* | 0.0753* |
|         | 9 | 0.0636 | 0.0636 | 0.0662 |
|         | 10| 0.0653 |

**TABLE 5**

| Dataset | m | weishi | ml-10M | ml-latest |
|---------|---|--------|--------|-----------|
| MRR@5   | 2 | 0.0750 | 0.0712 | 0.0721 |
|         | 3 | 0.0733 | 0.0738 | 0.0742 |
| TSC     | 4 | 0.0755* | 0.0755* | 0.0755 |
| PEC     | 5 | 0.0703 | 0.0717 | 0.0734 |
| EXC     | 6 | 0.0753* | 0.0769* | 0.0755 |
|         | 7 | 0.0708* | 0.0717 | 0.0734 |
|         | 8 | 0.0641 | 0.0673* | 0.0753* |
|         | 9 | 0.0636 | 0.0636 | 0.0662 |
|         | 10| 0.0653 |

**Bold** means the best result, **m** means the second-best result, **n** is slot number.
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.

Acknowledgments

Shilin Qu and Fajie Yuan have contributed equally to this work.

References

[1] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in Proc. Int. Conf. Learn. Representations, 2016. [Online]. Available: https://dblp.org/rec/journals/corr/HidasiKBT15.html?view=bibtex

[2] M. Quadrana, A. Karatzoglou, B. Hidasi, and P. Cremonesi, "Personalizing session-based recommendations with hierarchical recurrent neural networks," in Proc. 11th ACM Conf. Recommender Syst., 2017, pp. 130–137.

[3] J. Tang and K. Wang, "Personalized top-N sequential recommendation via convolutional sequence embedding," in Proc. 11th ACM Int. Conf. Web Search Data Mining, 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 Proc. 11th ACM Int. Conf. Web Search Data Mining, 2019, pp. 582–590.

[5] C. Ma, P. Kang, and X. Liu, "Hierarchical gating networks for sequential recommendation," in Proc. 25th ACM SIGKDD Conf. Knowl. Discov. Data Mining, 2019, pp. 825–833.

[6] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," IEEE Trans. Neural Netw., vol. 5, no. 2, pp. 157–166, Mar. 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, J. Haochuan, G. Guibing, X. Jian, X. Zhezhao, and X. Yilin, "Future data helps training: Modeling future contexts for session-based recommendation," in Proc. Web Conf., 2020, pp. 303–313.

[9] A. Vaswani et al., "Attention is all you need," in Proc. 31st Int. Conf. Neural Inf. Process. Syst., 2017, pp. 5998–6008.

[10] W.-C. Kang and J. McAuley, "Self-attentive sequential recommendation," in Proc. IEEE Int. Conf. Data Mining, 2018, pp. 197–206.

[11] X. Chen et al., "Sequential recommendation with user memory networks," in Proc. 11th ACM Int. Conf. Web Search Data Mining, 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 Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining, 2018, pp. 2467–2475.

[13] S. Sukhbaatar, A. Szelam, J. Weston, and R. Fergus, "End-to-end memory networks," in Proc. 28th Int. Conf. Neural Inf. Process. Syst., 2015, pp. 2440–2448.

[14] A. Graves et al., "Hybrid computing using a neural network with dynamic external memory," Nature, vol. 538, no. 7626, 2016, Art. no. 471.

[15] J. Johnson et al., "Inferring and executing programs for visual reasoning," in Proc. IEEE Int. Conf. Comput. Vis., 2017, pp. 3008–3017.

[16] M. J. Seo, S. Min, A. Farhadi, and H. Hajishirzi, "Query-reduction networks for question answering," in Proc. 9th Int. Conf. Learn. Representations, 2016. [Online]. Available: https://dblp.org/rec/conf/ICLR/SeoMFH17.html?view=bibtex

[17] J. Cai, R. Shin, and D. Song, "Making neural programming architectures generalize via recursion," in Proc. Int. Conf. Learn. Representations, 2017, pp. 108–116.

[18] T. Ebesu, B. Shen, and Y. Fang, "Collaborative memory network for recommendation systems," in Proc. 41st Int. ACM SIGIR Conf. Res. Devel. Inf. Retrieval, 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 Proc. 41st Int. ACM SIGIR Conf. Res. Devel. Inf. Retrieval, 2018, pp. 505–514.

[20] R. J. Gerrig, Psychology and Life, 20th ed. New Jersey, NJ, USA: Pearson Education, 2013.

[21] J. Johnson, "Fusing similarity models with Markov Chains for sparse sequential recommendation," in Proc. 16th Int. Conf. Data Mining, 2016, pp. 191–200.

[22] R. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized Markov Chains for next-basket recommendation," in Proc. 19th Int. Conf. World Wide Web, 2010, pp. 811–820.

[23] F. Yuan et al., "IBGD: Learning embeddings from positive unlabeled data with BGD," in Proc. Conf. Uncertainty Artif. Intell., 2018, pp. 198–207. [Online]. Available: https://dblp.org/rec/conf/ICLR/YuanX0GZCJ18.html?view=bibtex

[24] Y. Fajie, L. Chen, J. Yu, and W. Zhang, "LambdaFM: Learning optimal ranking with factorization machines using lambda surrogates," in Proc. 25th ACM Int. Conf. Inf. Knowl. Manage., 2016, pp. 227–236.

[25] B. Hidasi and A. Karatzoglou, "Recurrent neural networks with top-k gains for session-based recommendations," in Proc. 27th ACM Int. Conf. Inf. Knowl. Manage., 2018, pp. 843–852.

[26] Y. K. Tan, X. Xu, and Y. Liu, "Improved recurrent neural networks for session-based recommendations," in Proc. 1st Workshop Deep Learn. Recommender Syst., 2016, pp. 17–22.

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.
Y. Gu, T. Lei, R. Barzilay, and T. Jaakkola, “Learning to refine text based recommendations,” in Proc. Conf. Empir. Methods Natural Lang. Process., 2016, pp. 2103–2108.

E. Smirnova and F. Vasile, “Contextual sequence modeling for recommendation with recurrent neural networks,” in Proc. 2nd Workshop Deep Learn. Recommender Syst., 2017, pp. 2–9.

E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom, “Learning to transduce with unbounded memory,” in Proc. 28th Int. Conf. Neural Inf. Process. Syst., 2015, pp. 1828–1836.

A. Miller, A. Fisch, J. Dodge, A.-H. Karimi, A. Bordes, and J. Weston, “Key-value memory networks for directly reading documents,” in Proc. Conf. Empir. Methods Natural Lang. Process., 2016, pp. 1400–1409.

J. Zhang, X. Shi, I. King, and D.-Y. Yeung, “Dynamic key-value memory networks for knowledge tracing,” in Proc. Int. Conf. World Wide Web, 2017, pp. 765–774.

H. Le, T. Tran, and S. Venkatesh, “Learning to remember more with less memorization,” in Proc. Int. Conf. Learn. Representations, 2019. [Online]. Available: https://dblp.org/rec/journals/corr/abs-1901-01347.html?view=bibtex

D. Lian, H. Wang, Z. Liu, J. Lian, E. Chen, and X. Xie, “LightRec: A memory and search-efficient recommender system,” in Proc. Int. Conf. World Wide Web, 2020, pp. 695–705.

F. M. Harper and J. A. Konstan, “The MovieLens datasets: History and context,” ACM Trans. Interactive Intell. Syst., vol. 5, no. 4, pp. 19:1–19:19, 2015.

J. Tang et al., “Towards neural mixture recommender for long range dependent user sequences,” in Proc. Int. Conf. World Wide Web, 2019, pp. 1782–1793.

J. Wang, Q. Liu, Z. Liu, and S. Wu, “Towards accurate and interpretable sequential prediction: A CNN & attention-based feature extractor,” in Proc. 28th ACM Int. Conf. Inf. Knowl. Manage., 2019, pp. 1703–1712.

W. Guo, S. Wu, L. Wang, and T. Tan, “Personalized ranking with pairwise factorization machines,” Neurocomputing, vol. 214, no. C, pp. 191–200, 2016.

D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in Proc. Int. Conf. Learn. Representations, 2015. [Online]. Available: https://dblp.org/rec/journals/corr/KingmaB14.html?view=bibtex

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

Fajie Yuan received the PhD degree from the University of Glasgow, Glasgow, Scotland. He 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. His main research interests include deep learning and transfer learning and their applications in recommender systems.

Guibing Guo received the PhD degree in computer science from Nanyang Technological University, Singapore, in 2015. He is currently a professor with Software College, Northeastern University, Shenyang, China. 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 PhD degree from the Huazhong University of Science and Technology, Wuhan, China, in 2012. He is currently an associate professor with the School of Computer Science and Technology and the director of Cognitive Computing and Intelligent Information Processing (CCIIIP) Laboratory in the 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.

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