Linear Recurrent Units for Sequential Recommendation

Zhenrui Yue*
zhenrui3@illinois.edu
University of Illinois
Urbana-Champaign
Champaign, USA

Yueqi Wang*
yueqi@berkeley.edu
University of California, Berkeley
Berkeley, USA

Zhankui He†
zhh004@ucsd.edu
University of California, San Diego
San Diego, USA

Huimin Zeng
huiminz3@illinois.edu
University of Illinois
Urbana-Champaign
Champaign, USA

Julian McAuley
jmcauley@ucsd.edu
University of California, San Diego
San Diego, USA

Dong Wang
dwang24@illinois.edu
University of Illinois
Urbana-Champaign
Champaign, USA

ABSTRACT

State-of-the-art sequential recommendation relies heavily on self-attention-based recommender models. Yet such models are computationally expensive and often too slow for real-time recommendation. Furthermore, the self-attention operation is performed at a sequence-level, thereby making low-cost incremental inference challenging. Inspired by recent advances in efficient language modeling, we propose linear recurrent units for sequential recommendation (LRURec). Similar to recurrent neural networks, LRURec offers rapid inference and can achieve incremental inference on sequential inputs. By decomposing the linear recurrence operation and designing recursive parallelization in our framework, LRURec provides the additional benefits of reduced model size and parallelizable training. Moreover, we optimize the architecture of LRURec by implementing a series of modifications to address the lack of non-linearity and improve training dynamics. To validate the effectiveness of our proposed LRURec, we conduct extensive experiments on multiple real-world datasets and compare its performance against state-of-the-art sequential recommenders. Experimental results demonstrate the effectiveness of LRURec, which consistently outperforms baselines by a significant margin. Results also highlight the efficiency of LRURec with our parallelized training paradigm and fast inference on long sequences, showing its potential to further enhance user experience in sequential recommendation.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

recommender systems, sequential recommendation

[*] Both authors contributed equally to this research.
[†] Correspondence to Zhankui He (zhh004@ucsd.edu).

Conference'17, July 2017, Washington, DC, USA
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

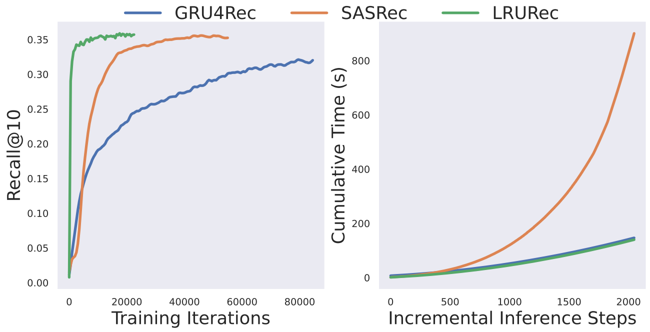

Figure 1: Training and inference efficiency of GRU4Rec, SASRec and LRURec on ML-1M. The proposed LRURec converges significantly faster than SASRec and GRU4Rec, while outperforming both models on Recall@10 scores and achieving $O(1)$ complexity with incremental inference.

ACM Reference Format:
Zhenrui Yue, Yueqi Wang, Zhankui He, Huimin Zeng, Julian McAuley, and Dong Wang. 2023. Linear Recurrent Units for Sequential Recommendation. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Sequential recommender systems play a crucial role in personalization platforms, such as Netflix [29] and Amazon [20], by capturing sequential patterns from user action history (e.g., user clicks) to accurately predict the next action. Over time, these recommenders have undergone significant advances from traditional approaches like Markov chain [9, 28] to deep neural networks like convolutional neural networks (CNNs) [33, 36] and recurrent neural networks (RNNs) [13, 14, 19]. Recently, self-attentive recommenders (SARs), inspired by transformers and their variants [4, 35], have further improved both training efficiency and recommendation accuracy, and thus represent the current state-of-the-art [11, 16, 31].

Despite their superior performance and training efficiency, SARs face criticism from the recommendation community due to their efficiency issues [25]. Specifically, SARs [11, 16, 31] heavily rely on computing item-to-item weights (referred to as "self-attention") at a sequence-level to encode user representations over time. That is, each time a new user action occurs, the system appends the action to the user history, and then recomputes all item-to-item weights for incremental update on user representation. This process incurs significant time and space costs. In contrast, RNNs [13, 14, 19] differ from the following perspectives: (1) in terms of inference, RNNs perform efficiently by storing a fixed-length hidden-state vector as the user representation. RNNs can retrieve and update the vector stepwise whenever a new user action occurs. (2) RNNs face challenges in training efficiency. The recurrent non-linear units in RNNs prevent parallelizable training, leading to the need for cumbersome training sequence augmentation and slow convergence speed. (3) Moreover, empirical studies have shown that existing RNNs fail to match the recommendation performance of SARs [11, 16, 31].

In this work, we propose a novel sequential recommender model, linear reccurent units for sequential recommendation (LRURec), which not only outperforms the recommendation accuracy and training efficiency of SARs with parallelizable training (see Figure 1), but also matches the inference efficiency of RNNs by only maintaining a fixed-length user vector for incremental updates. Our motivation stems from two key observations: (1) the effectiveness of linear models in handling long sequences, as demonstrated in natural language processing (NLP) tasks and models [24, 26, 32]; and (2) our findings that linear RNN recommenders can achieve recommendation performance comparable to vanilla non-linear RNN recommenders. To begin, we introduce LRU [5, 24], a linear version of the traditional recurrent units, which supports parallelizable training with the proposed recursive parallelization. This enables more efficient training compared to its non-linear counterparts. Furthermore, we incorporate a series of sequential modeling techniques from self-attentive architectures, such as layer normalization [1], residual connection [8], and position-wise feed-forward networks [35]. They further improve the recommendation accuracy to outperform existing SARs while retaining RNN-like inference efficiency due to the replacement of self-attention computation with efficient RNN-like recurrence operation.

We summarize the contributions of our work as follows[1]:

- We introduce a linear recurrent unit-based architecture LRURec into sequential recommendation, which addresses the dilemma of training efficiency, inference efficiency and recommendation performance.
- We propose a series of improvements in LRURec to address the lack of non-linearity and improve training dynamics. We further propose recursive parallelization that significantly accelerates both training and inference.
- We empirically demonstrate the effectiveness and efficiency of LRURec in comparison to state-of-the-art methods on multiple real-world datasets, where LRURec consistently outperforms baseline methods by a large margin.
- Our results challenge the necessity of the core self-attention module in existing SARs while highlighting the importance of other techniques in SARs like layer normalization, which provide deeper understanding and new opportunities to the architecture design for sequential recommenders.

Group	Model	Training Efficiency	Inference Efficiency	Rec. Performance
RNNs	GRU4Rec [14]	✗	✓	✓
	NARM [19]	✗	✓	✓
SARs	SASRec [16]	✓	✗	✓✓
	BERT4Rec [31]	✓	✗	✓✓
Others	FPMC [28]	✗	✓	✓
	FMLP-Rec [40]	✓	✗	✓✓
Ours	LRURec	✓	✓	✓✓

Table 1: Comparison among the proposed LRURec and representative sequential recommenders. LRURec is both training-efficient (like SARs [16, 31]) and inference-efficient (like RNNs [14, 19] or FPMC [28]). Additionally, LRURec matches SARs and FMLP-Rec [40] on recommendation performance.

2 RELATED WORK

2.1 Sequential Recommendation

Sequential Recommendation has a primary objective of predicting the user's next item of interest by modeling their past actions in chronological order [14, 16, 28, 33]. The central focus of sequential recommendation lies in the creation of a model architecture that is both effective and efficient. First, CNN- and RNN-based models were proposed to leverage the expressiveness of deep neural networks to model user sequences [14, 19, 33], which outperform linear sequential models such as FMC and FPMC [28]. Nevertheless, RNN-based models faced limitations in training efficiency due to the lack of parallel training. Subsequently, transformer-based models [11, 12, 16, 31] emerged as a solution, offering accelerated training and improved recommendation accuracy through the parallelizable self-attention operation [35]. With efficient training and accurate item-to-item relevance via self-attention, transformer-based models stand as the *state-of-the-art* methods for sequential recommendation. Additionally, MLP-based sequential recommenders [21, 22] like FMLP-Rec [40] exhibit comparable recommendation performance relying solely on the MLP architecture. However, the inference efficiency of transformer- and MLP-based models lag behind RNN-based models [25]. Specifically, whenever a new user action appears, RNN-based models only need to update the latest hidden state, whereas both transformer- and MLP-based models must recompute sequence-level relevance for inference.

In this work, we aim to propose a new model architecture for sequential recommendation based on linear recurrent units [24], which enjoys both the training efficiency of transformer-based models and inference efficiency of RNN-based models.

2.2 Efficient Language Modeling

The difficulty of building an NLP model to achieve (1) training efficiency, (2) inference efficiency and (3) model performance like Table 1 is known as the "impossible triangle" [32]. To achieve these goals, various approaches have been proposed from different aspects. Attention free transformer (AFT) or RWKV [26, 39] simplify token-token attention weights to element-wise operations

[1] Our implementation is available at https://github.com/yueqirex/LRURec.

and moves softmax operations to key vectors, which shows comparable performance to a vanilla transformer model [35] after scaling up [26]. S4 [5] is based on fundamental state space models (SSM) with continuous-time memorization [6], which demonstrates efficiency and effectiveness in long-sequence modeling (e.g. long-range arena benchmark [34]). Linear recurrent units [24] match the long-sequence performance of SSMs with an improved initialization strategy and systematic ablations. Most recently, RetNet [32] proposes attention-like retention that have equivalent recurrent and parallel formulation, RetNet can thus be trained in parallel while achieving low inference costs like RNN models.

In our work, we study the efficient modeling problem in the context of sequential recommendation. The differences are (1) all mentioned methods are primarily studied and evaluated in NLP, where existing solutions (e.g., RWKV) are not specifically designed for recommendation and can lead to overfitting and latency issues; (2) the majority of such methods are optimized for long-range modeling tasks, which may cause performance issues upon recommendation tasks that prioritize short sequences of user actions. Different from existing approaches, we propose a novel sequential recommender based on linear recurrence. With the carefully designed LRU block and recursive parallelization, our model performs well regardless of sequence length and achieves both training and inference efficiency for sequential recommendation.

3 METHODOLOGY

As shown in previous works [5, 24, 32], efficient NLP models are capable of capturing long-term dependencies in a wide range of sequence-to-sequence tasks. Additionally, such models demonstrate significantly improved efficiency thanks to parallelizable training and incremental inference. Motivated by such observations, the proposed linear recurrent units for sequential recommendation (LRURec) incorporates the advantages of both RNN-based and transformer-based models, while requiring significantly reduced computing power with efficient incremental inference and rapid convergence via parallelized training. We introduce the key components and the overall model of LRURec in the following.

3.1 Setup

Data: Let input $x \in \mathcal{X}$ represent the user action history $[x_1, x_2, ..., x_L]$ of length L in chronological order, where the elements are represented in the item space $\mathcal{I}$ (i.e., $x_i \in \mathcal{I}$). The ground truth y is the next user action $x_{L+1} \in \mathcal{I}$ (i.e., $y = x_{L+1}$).

Model: The recommender model is denoted with function f, with H being the hidden dimension in f. Upon input x, f generates prediction scores over $\mathcal{I}$. Ideally, f predicts item y with the highest score for data pair (x, y), namely $y = \arg\max f(x)$.

Learning: The optimization of the recommender model corresponds to the likelihood maximization of ground truth item y upon input x. In other words, the learning objective is to minimize the negative log-likelihood loss $\mathcal{L}$ over distribution $\mathcal{X}$:

$$\min \mathbb{E}_{(x,y)\sim\mathcal{X}} \mathcal{L}(f(x), y). \tag{1}$$

3.2 Linear Recurrent Unit

Decomposing Linear Recurrence. We first introduce a simplified form of linear recurrence [5]. For input x_k at time step k, we represent the hidden representation h_k and output y_k with learnable matrices $A \in \mathbb{R}^{H\times H}$, $B \in \mathbb{R}^{H\times H_{\text{in}}}$, $C \in \mathbb{R}^{H_{\text{out}}\times H}$ and $D \in \mathbb{R}^{H_{\text{out}}\times H_{\text{in}}}$:

$$h_k = Ah_{k-1} + Bx_k, \quad y_k = Ch_k + Dx_k, \tag{2}$$

where the input and output dimensions are denoted with H_{in} and H_{out} (i.e., embedding size), and the hidden dimension size with H. Different from RNN models (i.e., $h_k = \sigma(Ah_{k-1} + Bx_k)$), we discard the non-linearity σ to enable the serialization of h_k:

$$\begin{aligned} h_k &= Ah_{k-1} + Bx_k = A^2 h_{k-2} + ABx_{k-1} + Bx_k \\ &= A^3 h_{k-3} + A^2 Bx_{k-2} + ABx_{k-1} + Bx_k = \dots \\ &= \sum_{i=1}^{k} A^{k-i} Bx_i \quad \text{with} \quad h_1 = Bx_1. \end{aligned} \tag{3}$$

By unrolling the $h_k = Ah_{k-1} + Bx_k$ along the time steps, h_k can be written in closed-form w.r.t. the matrices A, B and the input elements $x_1, x_2, \dots, x_k$. However, the repeated computation of matrix multiplication is inefficient and may lead to numerical issues as k increases (e.g., overflow). To this end, we leverage matrix diagonalization (i.e., eigendecomposition) and introduce eigenvalues and eigenvectors to (1) reduce matrix-level computation and (2) control the numerical stability of h_k. In particular, we perform decomposition on A with $A = P\Lambda P^{-1}$, in which Λ is diagonal and consists of the eignevalues $\lambda_1, \lambda_2, \dots, \lambda_H$, P is an invertible matrix of size $H \times H$. Consequently, the computation of A^n reduces to $P\Lambda^n P^{-1}$, and thereby significantly improving computation efficiency.

Representation in Complex Space. Despite the eigendecomposition of matrix $A = P\Lambda P^{-1}$, the eigenvalues and eigenvectors of A do not necessarily lie in real space $\mathbb{R}$. Therefore, we extend P and Λ to the complex space with $P \in \mathbb{C}^{H\times H}$ and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_H) \in \mathbb{C}^{H\times H}$ [24]. Similarly, we extend h and B, C to the complex space $\mathbb{C}$. As such, the computation of $\sum_{i=1}^{k} A^{k-i} Bx_i$ can now be written as $\sum_{i=1}^{k} P\Lambda^{k-i} P^{-1} Bx_i$. We further write $\bar{h} = P^{-1}h$, $\bar{B} = P^{-1}B$ and $\bar{C} = CP^{-1}$, which simplifies the formulation of $\bar{h}_k$ and output y_k to:

$$\begin{aligned} \bar{h}_k &= \Lambda \bar{h}_{k-1} + \bar{B}x_k = \dots = \sum_{i=1}^{k} \Lambda^{k-i} \bar{B}x_i, \\ y_k &= \Re(\bar{C}\bar{h}_k) + Dx_k, \end{aligned} \tag{4}$$

where $\Re(\bar{C}\bar{h}_k)$ is the real part of complex vector $\bar{C}\bar{h}_k$. To improve multiplication efficiency in the complex space, we represent Λ in polar form with absolute value r and argument θ (i.e., $\Lambda = re^{j\theta} = r(\cos(\theta) + j\sin(\theta))$), j stands for the imaginary unit. Let $r = e^{-\nu}$, we have $\Lambda = re^{j\theta} = e^{-\nu+j\theta}$ where $\nu, \theta \in \mathbb{R}^H$, thus the involved computation is reduced to $(-\nu + j\theta)$. Another advantage of the polar form is that parameters ν and θ can be now optimized in the real space $\mathbb{R}$. As a result, the scale of Λ can be computed with $e^{-\nu}$, while the exponentiation of matrix A in Equation (3) is replaced with the efficient exponential of diagonal matrix Λ.

Parameterization. To avoid numerical instability, a simple condition is to let elements in Λ satisfy $|\lambda_i| < 1, i = 1, 2, \dots, H$, the

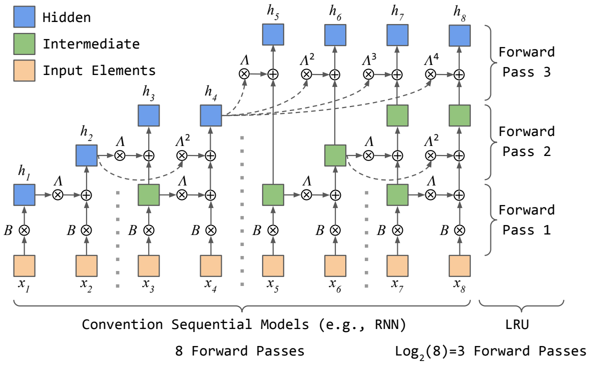

Figure 2: Recursive parallelization for LRURec, we illustrate the recursive split and parallel forward pass.

condition is equivalent to $e^{-\nu_i} < 1$ for the polar form. Therefore, we use another exponential $\lambda_i = \exp(-\exp(\log(\nu_i)) + j\theta_i)$ with $\nu_i > 0$ [5, 24]. More specifically, we use log parameters $\nu^{\log}$, $\theta^{\log} \in \mathbb{R}^H$ and introduce normalization factor $\gamma^{\log} \in \mathbb{R}^H$, with $\nu^{\log} = \log(\nu), \theta^{\log} = \log(\theta)$. $\gamma^{\log}$ normalizes input element-wise and is initialized with $\gamma_i^{\log} \leftarrow \log(\sqrt{1 - |\lambda_i|^2})$. The rest log parameters are initialized following the ring approach with radius between $[0.8, 0.99]$ [24], while the rest matrices are initialized via truncated normal initialization [31]. Using h, B and C to represent $\bar{h}$, $\bar{B}$ and $\bar{C}$ in the complex space $\mathbb{C}$, we summarize the above parameterization and formulate the final linear recurrence unit as follows:

$$\begin{aligned} \Lambda &= \mathrm{diag}(\exp(-\exp(\nu^{\log}) + j\exp(\theta^{\log}))), \\ h_k &= \Lambda h_{k-1} + \exp(\gamma^{\log}) \odot Bx_k, \\ y_k &= \Re(Ch_k) + x_k, \text{ with } h_1 = Bx_1. \end{aligned} \tag{5}$$

In our implementation, we double the size of H in Λ to improve the modeling of recurrence. We also adopt identity matrix $\mathbb{I}$ as D in the output function as $H_{\text{in}} = H_{\text{out}}$. Discarding D additionally reduces learnable parameters and constructs a residual connection between the input and output. In the following, we use LRU to denote the proposed linear recurrence operation in Equation (5), the effectiveness of our design is demonstrated in Section 5.4.

3.3 Recursive Parallelization

Although matrix diagonalization and the exponential parameterization improves the computational efficiency of linear recurrence, the forward pass in Equation (5) is element-wise for each time step, resulting in linear time complexity w.r.t. sequence length. Inspired by the parallel scan algorithms [2, 3, 18, 30], we develop recursive parallelization designed specifically for LRURec. In particular, long input sequences are recursively split into subsequences to enable parallel processing, followed by the scaled addition of hidden features from each subsequence, and thereby improving the forward pass of LRURec to logarithmic time complexity.

For simplicity, we formulate the recursive parallelization based on $h_t = \sum_{i=1}^{k} \Lambda^{k-i} Bx_i$ in Equation (4). Given input sequence x of length t and time step k with $k < t$, the final output h_t at time step

```
def recursive_parallelization(x, B, La):
    # 1. sequence padding
    L_log2 = int(math.ceil(math.log2(x.shape[1])))
    x = F.pad(x, (0, 0, 2**L_log2-x.shape[1], 0, 0, 0))
    N, L, D = x.shape  # left padded sequence shape
    # 2. recursive split
    h = torch.matmul(x, B)
    for i in range(1, L_log2+1):
        l = 2 ** i  # length of subsequences
        h = h.reshape(N*L//l, l, D)
        # 3. parallel forward pass
        h1, h2 = h[:, :l//2], h[:, l//2:]
        if i > 1:  # compute [La, La^2, ..., La^l]
            La = torch.cat((La, La * La[-1]), 0)
        h2 = h2 + La * h1[:, -1:]  # linear recurrence
        h = torch.cat([h1, h2], 1)
    return h
```

Algorithm 1: LRURec Recursive Parallelization, with x, B and La representing input x, parameters B and Λ.

t can be formulated as:

$$\begin{aligned} h_t = \sum_{i=1}^{t} \Lambda^{t-i} Bx_i &= \sum_{i=1}^{k} \Lambda^{t-i} Bx_i + \sum_{i=k+1}^{t} \Lambda^{t-i} Bx_i \\ &= \Lambda^{t-k} \sum_{i=1}^{k} \Lambda^{k-i} Bx_i + \sum_{i=k+1}^{t} \Lambda^{t-i} Bx_i \\ &= \Lambda^{t-k} h_k + \sum_{i=k+1}^{t} \Lambda^{t-i} Bx_i. \end{aligned} \tag{6}$$

The results suggest that it is possible to split the sequence $x = [x_1, x_2, ..., x_t]$ into $[x_1, x_2, ..., x_k]$ and $[x_{k+1}, x_2, ..., x_t]$ for parallel processing on the subsequences. Then, h_t can be obtained by summing the output from the subsequences, where the last-step hidden feature from the first subsequence (i.e., h_k) is additionally multiplied with Λ^{t-k} to correct the time steps.

Based on such observations, we propose recursive parallelization to accelerate the forward feeding of sequential input. The recursive parallelization process is illustrated in Figure 2. Specifically, we perform the the following steps in recursive parallelization:

1. Sequence Padding: Given input sequence x of length t, we first pad the sequence length to the power of two (i.e., $2^2, 2^3, \ldots$). The reason for such padding is to enable recursive split of the input sequences until reaching the shortest length of two (i.e., $[x_1, x_2], [x_3, x_4], [x_5, x_6], \ldots$), which maximizes the performance of our recursive parallelization algorithm.

2. Recursive Split: With the padded sequence of length 2^k, we multiply x with B and perform k forward passes. In the i-th step, the input sequence is split into 2^{k-i} subsequences, each of length 2^i (i.e., $[x_1, x_2, \ldots, x_{2^i}], [x_{2^i+1}, x_{2^i+2}, \ldots, x_{2*2^i}], \ldots$). The subsequences are processed in parallel to perform linear recurrence (see next step), followed by restoring the original sequence. Consequently, the time complexity reduces with no additional space required.

3. Parallel Forward Pass: For each input $x = [x_1, x_2, \ldots, x_{2l}]$ of length $2l$ ($l \geq 1$), we further split the input into two equal subsequences of length l, namely $[x_1, x_2, \ldots, x_l]$ and $[x_{l+1}, x_{l+2}, \ldots, x_{2l}]$. The last element of the first subsequence (i.e., x_l) is multiplied with

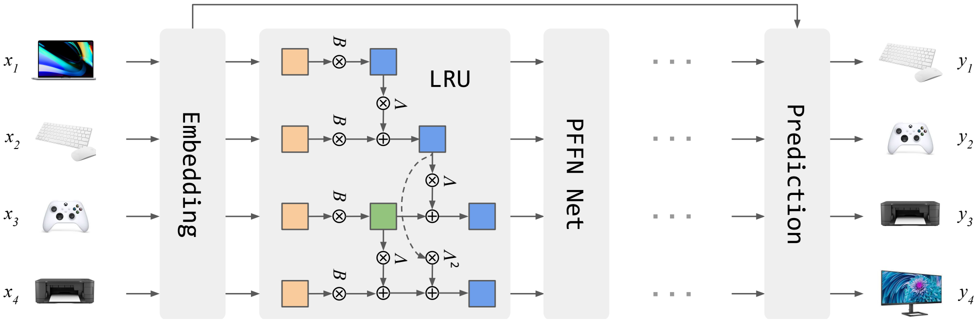

Figure 3: The overall architecture of the proposed LRURec.

$[\Lambda, \Lambda^2, \ldots, \Lambda^l]$ respectively, and element-wise added to the second subsequence. In other words, the second subsequence becomes $[x_{l+1}+\Lambda x_l, x_{l+2}+\Lambda^2 x_l, \ldots, x_{2l}+\Lambda^l x_l]$. Then, we restore the original sequence shape: $[x_1, x_2, \ldots, x_l, x_{l+1}+\Lambda x_l, x_{l+2}+\Lambda^2 x_l, \ldots, x_{2l}+\Lambda^l x_l]$. Such forward passes are performed in parallel for all subsequences in each of the k forward passes in the recursive split step.

Recursive parallelization can be perform regardless of input length and hidden dimension, and is designed for full-length training and inference. We provide an implementation of recursive parallelization with PyTorch-like pseudocode in Algorithm 1, with `x`, `B` and `La` representing input x, parameters B and Λ. The above steps follow the divide-and-conquer principle to break down long sequences into subsequences recursively, such that linear recurrence can be computed on the subsequences in parallel. By applying recursive parallelization, we reduce the number of forward passes to $\log_2(L)$ for input sequence of length L (e.g., three passes for an eight-element sequence, see Figure 2), which significantly improves the time efficiency for both training and inference in LRURec.

3.4 Overall LRURec Model

In this section, we introduce the overall architecture of the proposed LRURec. The proposed model comprises of: (1) embedding module; (2) LRU block with position-wise feed-forward network (PFFN) and (3) prediction layer. The overall model is illustrated in Figure 3, we describe the details of each module in the following.

Embedding Module. Similar to existing methods, we use a learnable matrix $E \in \mathbb{R}^{|\mathcal{I}| \times H_{\text{in}}}$ to transform the discrete item IDs in $\mathcal{I}$ from the input sequence to the high-dimensional embedding space $\mathbb{R}^{H_{\text{in}}}$. For sequences of different lengths, we perform left padding to the power of two for parallelization, as demonstrated in Section 3.3. Layer normalization (`LayerNorm`) is performed after the embedding retrieval. Hence, for input sequence $x = [x_1, x_2, ..., x_t]$, we denote the embedding module with `Embed`:

$$\begin{aligned} \texttt{Embed}(E, x) &= \texttt{LayerNorm}([E_{x_1}, E_{x_2}, \ldots, E_{x_t}]), \\ \texttt{LayerNorm}(x) &= \alpha * \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \end{aligned} \tag{7}$$

where α and β are learnable rescaling factors, μ and σ represents the mean and standard deviation of the input, ϵ is added to the denominator in `LayerNorm` for numerical stability.

LRU Block. As in Section 3.2, we use `LRU` to describe the linear recurrence operation in LRURec. We additionally apply layer normalization on the output hidden features of `LRU` and denote the process with `LRUNorm`. Nevertheless, `LRU` sacrifices non-linearity in the recurrence operation for improved performance and efficiency. To compensate for the absence of non-linearity, we leverage position-wise feed-forward network (PFFN) to improve the modeling of user actions in the hidden dimension. In particular, we use `PFFN` to describe the two-layer MLP network:

$$\texttt{PFFN}(x) = \texttt{GELU}(W^{(2)}\texttt{GELU}(W^{(1)}x + b^{(1)}) + b^{(2)}), \tag{8}$$

where $W^{(1)} \in \mathbb{R}^{4H \times H}, W^{(2)} \in \mathbb{R}^{H \times 4H}, b^{(1)} \in \mathbb{R}^{4H}, b^{(2)} \in \mathbb{R}^{H}$ are the parameters for the two-layer MLP, and `GELU` refers to the GELU activation. We additionally introduce sublayer connection (`SubLayer`) with both layer normalization and residual connection on the PFFN net to improve the recommendation performance and training dynamics. In short, we leverage linear recurrence unit to efficiently process sequential input. PFFN, layer normalization and residual connections are additionally introduced to improve the modeling of non-linear transition patterns in LRURec:

$$\begin{aligned} \texttt{LRUNorm}(x) &= \texttt{LayerNorm}(\texttt{LRU}(x)), \\ \texttt{SubLayer}(\texttt{PFFN}, x) &= \texttt{LayerNorm}(\texttt{PFFN}(x) + x). \end{aligned} \tag{9}$$

In our experiments, we stack two LRU blocks following [16, 31].

Prediction Layer. Given hidden features h_t at time step t from the previous LRU block, we compute the scores over $\mathcal{I}$ for next-item prediction via the `Pred` function:

$$\texttt{Pred}(h_t) = Eh_t + b^o, \tag{10}$$

where E is the embedding matrix from the embedding module and $b^o \in \mathbb{R}^{|\mathcal{I}|}$ is an additional bias term. Thanks to the non-linearity introduced in the LRU block, dot product between embedding features and h_t can capture non-trivial patterns despite utilizing shared item features E. Additionally, the shared E significantly reduces the model size, while effectively alleviates overfitting in LRURec.

4 DISCUSSION

Why does linear recurrence perform well in sequential recommendation? We have two explanations for the improvements of LRURec: (1) Linear recurrence by design assigns higher weights to recent items, thus it is effective for modeling recommendation

Datasets	Users	Items	Interact.	Lenth	Density
ML-1M	6,040	3,416	1M	165.5	5e-2
Beauty	52,204	57,289	398K	7.6	1e-4
Video	31,013	23,715	287K	9.3	4e-4
Sports	83,970	83,728	596K	7.1	8e-5
Steam	334,730	13,047	3.7M	11.0	8e-4
XLong	69,691	2,122,932	66.8M	958.8	5e-4

Table 2: Dataset statistics after preprocessing.

data that emphasizes recent interactions. To examine this, we inspect the average $|\lambda|$ values (i.e., $e^{-\nu}$) on different sequence lengths. Notice that higher $|\lambda|$ values suggest the inclusion of more history information. With ~200 sequence length (i.e., ML-1M), we observe ~0.4 for both LRU blocks. The values reduce to 0.25 and 0.31 for ~10 sequence length in Beauty (see Section 5.3). As $|\lambda|$ is initialized close to 1, the low values indicate high emphasis on recent items, thus providing a solid justification for the recurrence design in LRURec. (2) PFFN and hierarchical LRU blocks further improve the modeling of non-trivial transition patterns in long-range dependencies and dense datasets. For example, PFFN and stacking LRU block causes up to 11.24% and 3.43% performance improvements on long sequences (ML-1M), while contributing less than 1% improvements on the sparse Beauty data (see Section 5.4). Hence, the combination of additional non-linearity and hierarchical linear recurrence plays a vital role in the overall performance of LRURec.

How does the complexity of LRURec compare to other models? We use H and L to represent the model hidden dimension and input sequence length, $\mathcal{I}$ and $\mathcal{U}$ to represent the items and users. The quantity of the learnable parameters in the proposed LRURec is comparable to RNN-based sequential recommenders, which have $O(|\mathcal{I}|H + H^2)$ space complexity. In comparison, factorization models have $O((|\mathcal{I}| + |\mathcal{U}|)H)$ and transformer models have $O((L + |\mathcal{I}|)H + H^2)$ space complexity. Given the low hidden dimension size ($H = 64$ in our experiments), the space complexity of LRURec is smaller in size and does not grow with increasing user numbers. Moreover, we use identity matrix as D and decompose $A = P\Lambda P^{-1}$ (P is integrated in B and C matrices), which leads to additional parameter reduction in LRURec. For time complexity, LRURec achieves $O(\log(L)H^2)$ for full-length input sequence as a result of our recursive parallelization design. In contrast, typical RNN-based recommenders require $O(LH^2)$ in time and the original transformer-based recommenders require $O(L^2H + LH^2)$. Like RNNs, LRURec additionally support incremental inference with only $O(H^2)$ complexity. We demonstrate the advantages of LRURec on both performance and efficiency in Section 5.

5 EXPERIMENTS

5.1 Experimental Setup

Datasets. Our model is evaluated on the following datasets:

- **MovieLens**: A benchmark dataset for movie recommendation, we select the widely used ML-1M here [7].
- **Amazon**: A series of datasets with product reviews from Amazon, here we select Beauty, Video and Sports [10, 23].
- **Steam**: A video game review dataset crawled from Steam, a large online video game distribution platform [16].
- **XLong**: XLong is an Alibaba dataset known for long interaction histories to evaluate lifelong sequential models [27].

For preprocessing, we follow [11, 16, 37, 38] to construct input sequences and exclude users and items with fewer than 5 interactions. For maximum sequence length, we adopt 200 for ML-1M, 1000 for XLong and 50 for the rest datasets. We follow the leave-last-out strategy on dataset splitting and use the most recent item as the test set, the second most recent item as the validation set, and the rest items in the sequences as the training set. During testing, we include both training and validation items as input. We report the dataset statistics after preprocessing in Table 2.

Baseline Methods. We compare our LRURec against multiple baseline methods, which include classic factorization and Markov chain-based methods (e.g. MF, FISM, FPMC), RNN-based models (e.g. GRU4Rec, NARM) as well as transformer- and MLP-based state-of-the-art models (e.g., SASRec, BERT4Rec and FMLP-Rec):

- **MF**: A vanilla factorization model that learns user and item latent representations for next-item prediction [17].
- **FISM**: FISM does not explicitly model user preference and predicts next interaction via item-to-item similarity [15].
- **FPMC**: FPMC is a matrix factorization model and uses Markov chains to capture user transition patterns [28].
- **GRU4Rec**: A classic sequential recommender that models user-item interactions using a GRU-based model [13, 14].
- **NARM**: Also a RNN-based sequential recommender with local and global user modeling for next-item prediction [19].
- **SASRec**: The first unidirectional transformer-based sequential recommender, SASRec leverages unidirectional self-attention to capture user-item transition patterns [16].
- **BERT4Rec**: A bidirectional transformer encoder architecture for sequential recommendation. BERT4Rec is learnt via predicting a random proportion of masked items [31].
- **FMLP-Rec**: A filter-based all-MLP model that learns in the complex domain using fast Fourier transformation [40].

Evaluation. For evaluation results, we select models with the best validation Recall@10 scores in training to perform prediction on the test sets. The models are evaluated using Recall@k and NDCG@k metrics, and with $k \in \{10, 20\}$. The predicted items are ranked against all items in the dataset to compute the final scores.

Implementation Details. For the baseline methods, we refer to the original papers for implementation. We train all models using cross entropy loss with AdamW optimizer using the static learning rate of 1e-3. During training, we use a batch size of 128 (32 for XLong) and set the maximum epoch to be 500, validation is performed every 500 to 2000 iterations depending on the data size. Early stopping is triggered if validation Recall@10 does not improve in 10 consecutive validation rounds. We perform grid search for hyperparameters, with weight decay from [0, 1e-6, 1e-4, 1e-2] and dropout rate from [0.2, 0.4, 0.6, 0.8]. Hyperparameters not mentioned above were used as reported in the original implementation[2].

[2] Notice that our preprocessing differs from the k-core method and leads to more realistic and sparser distributions. As such, the reported numbers should not be directly compared with those found in works that employ k-core preprocessing.

Dataset	Metric	MF	FISM	FPMC	GRU4Rec	NARM	SASRec	BERT4Rec	FMLP-Rec	LRURec	Improv.
ML-1M	NDCG@10 ↑	0.02835	0.03327	0.12030	0.15943	0.15310	0.18293	0.16377	0.15295	**0.19065**	4.22%
	NDCG@20 ↑	0.03991	0.04655	0.14370	0.18735	0.17774	0.21232	0.19118	0.17912	**0.21767**	2.52%
	Recall@10 ↑	0.06290	0.07368	0.23030	0.28317	0.27290	0.31441	0.30894	0.29073	**0.32402**	3.06%
	Recall@20 ↑	0.10910	0.12520	0.31510	0.39378	0.37066	0.43084	0.41738	0.39454	**0.43118**	0.08%
Beauty	NDCG@10 ↑	0.01193	0.01281	0.02091	0.01997	0.01887	0.02917	0.02422	0.02515	**0.03088**	5.86%
	NDCG@20 ↑	0.01563	0.01621	0.02371	0.02343	0.02239	0.03403	0.02937	0.02945	**0.03560**	4.61%
	Recall@10 ↑	0.02596	0.02614	0.03586	0.03379	0.03312	0.05043	0.04679	0.04623	**0.05284**	4.78%
	Recall@20 ↑	0.03987	0.04008	0.04699	0.04751	0.04708	0.06965	0.06727	0.06336	**0.07161**	2.81%
Video	NDCG@10 ↑	0.01337	0.02813	0.04318	0.05081	0.05329	0.05829	0.05600	0.05294	**0.06198**	6.33%
	NDCG@20 ↑	0.01722	0.03692	0.05267	0.06200	0.06512	0.07111	0.06958	0.06576	**0.07547**	6.13%
	Recall@10 ↑	0.02634	0.05815	0.08110	0.09512	0.09777	0.10910	0.11252	0.10885	**0.11337**	0.76%
	Recall@20 ↑	0.04052	0.09281	0.11830	0.13959	0.14488	0.15989	0.16640	0.15991	**0.16705**	0.39%
Sports	NDCG@10 ↑	0.00266	0.00810	0.01070	0.01002	0.01117	0.01236	0.01457	0.01546	**0.01815**	17.43%
	NDCG@20 ↑	0.00362	0.01036	0.01257	0.01243	0.01414	0.01520	0.01813	0.01871	**0.02149**	14.86%
	Recall@10 ↑	0.00413	0.01614	0.01936	0.01882	0.02087	0.02294	0.02879	0.02987	**0.03148**	5.39%
	Recall@20 ↑	0.00861	0.02644	0.02643	0.02841	0.03265	0.03428	0.04295	0.04276	**0.04477**	4.24%
Steam	NDCG@10 ↑	0.06480	0.04209	0.04377	0.06512	0.06480	0.06759	0.06433	0.06081	**0.06934**	2.59%
	NDCG@20 ↑	0.07996	0.05439	0.05582	0.08050	0.07996	0.08298	0.08117	0.07521	**0.08508**	2.53%
	Recall@10 ↑	0.12166	0.08561	0.08773	0.12204	0.12166	0.12597	0.12188	0.11987	**0.12831**	1.86%
	Recall@20 ↑	0.18199	0.13450	0.13570	0.18319	0.18199	0.18716	0.18245	0.17719	**0.19082**	1.96%

Table 3: Main performance results, best results are marked in bold, second best results underlined.

Methods	XLong			
	NDCG@10 ↑	Recall@10 ↑	NDCG@20 ↑	Recall@20 ↑
SASRec	0.30949	0.49348	0.33531	0.59527
BERT4Rec	0.23426	0.41536	0.27293	0.56860
LRURec	**0.34867**	**0.52688**	**0.37174**	**0.61800**

Table 4: Long-range modeling performance results, best results are marked in bold, second best results underlined.

	ML-1M	Beauty	Video	Sports	Steam	XLong
$\vert\lambda\vert$ **Block 1**	0.3927	0.2540	0.2501	0.3675	0.3382	0.7720
$\vert\lambda\vert$ **Block 2**	0.4216	0.3099	0.3047	0.4287	0.3687	0.8045

Table 5: Average $|\lambda|$ values of each LRU block, higher $|\lambda|$ indicates the incorporation of increased history information.

5.2 Overall Performance

Our main performance results are reported in Table 3. Here, rows represent the dataset and metric, and the columns represent each of the methods, we mark the best results in bold and underline the second best results. We also compute the relative improvement of LRURec compared to the best-performing baseline method (i.e., Improv.). We observe: (1) LRURec consistently outperforms baseline methods across all metrics and datasets, with an average performance improvement of 4.39% compared to the second best method. Despite the efficient and light-weight design, the performance gains of LRURec can go up to over 10% depending on the data distribution (e.g., 17.43% on NDCG@10 in Sports). (2) The performance gains of LRURec are more pronounced on sparse datasets, while being comparatively modest on dense datasets. For example, LRURec achieves 1.32% average improvements on the relatively dense ML-1M. The improvement is much more significant on the sparse Sports dataset with average gains of 10.48%, suggesting the substantial benefits of LRURec on sparse data. (3) In contrast to recall, LRURec demonstrates better ranking performance. For instance, there is a noteworthy increase of 6.73% in the average NDCG@10 scores with LRURec, while the relative improvement on Recall@10 is slightly lower at 3.08%. Overall, we find LRURec performs particularly well on sparse data and shows significantly improved ranking performance compared to the baseline methods. LRURec also achieves consistent performance improvements in all scenarios, suggesting the effectiveness of LRURec regardless of data domains.

5.3 Long-Range Modeling Performance

To examine the performance of LRURec on long-range dependencies, we additionally experiment on XLong: a large-scale dataset with ~1k sequence length. Due to scalability issues, we only experiment on selected state-of-the-art baselines (SASRec and BERT4Rec) along with LRURec[3]. We also reduce the hyperparameter search to [0, 1e-2] for weight decay and [0.2, 0.4] for dropout rate. We improve the training efficiency on XLong by randomly sampling 100 negative items to compute loss and update model in training. For evaluation, we randomly sample 10k negative items compute the metrics, the experiment results are reported in Table 4. Analogous to the main results, LRURec outperforms baseline methods by a considerable margin, indicating the effectiveness of LRURec even for long-range dependencies. For example, LRURec achieves 8.53%

[3] The demand for intricate sequence processing and augmentation renders the training of XLong computationally infeasible for RNN-based models and FMLP-Rec.

Variants	Metric	ML-1M	Beauty	Video	Sports	Steam
		NDCG ↑ / Recall ↑	NDCG ↑ / Recall ↑	NDCG ↑ / Recall ↑	NDCG ↑ / Recall ↑	NDCG ↑ / Recall ↑
LRURec	@10	**0.19298** / 0.32636	0.03088 / **0.05284**	**0.06198** / **0.11337**	**0.01815** / **0.03148**	0.06934 / 0.12831
	@20	**0.21990** / **0.43313**	0.03560 / **0.07161**	**0.07547** / **0.16705**	**0.02149** / **0.04477**	0.08508 / 0.19082
(1) **LRURec** w/o `LayerNorm`	@10	0.19242 / **0.32661**	0.01562 / 0.02904	0.05222 / 0.09435	0.01072 / 0.02008	0.06793 / 0.12691
	@20	0.21751 / 0.42584	0.01929 / 0.04364	0.06430 / 0.14232	0.01348 / 0.03106	0.08360 / 0.18920
(2) **LRURec** w/o `Residual`	@10	0.18038 / 0.31185	0.00932 / 0.01924	0.02827 / 0.05503	0.00580 / 0.01205	0.06138 / 0.11639
	@20	0.20606 / 0.41369	0.01244 / 0.03171	0.03552 / 0.08394	0.00763 / 0.01934	0.07648 / 0.17647
(3) **LRURec** w/o `PFFN`	@10	0.17319 / 0.29318	0.03075 / 0.05249	0.06019 / 0.11177	0.01769 / 0.03026	0.06565 / 0.12251
	@20	0.19768 / 0.39024	0.03556 / 0.07157	0.07356 / 0.16496	0.02109 / 0.04380	0.08098 / 0.18348
(4) **Backbone**: LRU (1 Layer)	@10	0.18777 / 0.31489	**0.03120** / 0.05276	0.06017 / 0.11102	0.01742 / 0.03012	0.06784 / 0.12563
	@20	0.21330 / 0.41569	**0.03580** / 0.07104	0.07350 / 0.16400	0.02062 / 0.04279	0.08323 / 0.18684
(6) **Backbone**: RWKV	@10	0.15942 / 0.28411	0.02620 / 0.04653	0.04913 / 0.09400	0.01663 / 0.03071	0.04744 / 0.09081
	@20	0.18549 / 0.39112	0.03089 / 0.06511	0.05983 / 0.13643	0.01962 / 0.04255	0.05915 / 0.13737
(5) **Backbone**: S4	@10	0.16460 / 0.28158	0.02260 / 0.03671	0.05039 / 0.09203	0.01120 / 0.01907	**0.06945** / **0.13044**
	@20	0.18972 / 0.38122	0.02584 / 0.04956	0.06212 / 0.13869	0.01315 / 0.02680	**0.08573** / **0.19515**

Table 6: Ablation results, best results are marked in bold, second best results underlined.

average improvements across all metrics compared to the best-performing baseline SASRec. In addition, we evaluate the weights of history information in LRURec by computing the average $|\lambda|$ in each of the LRU blocks. The $|\lambda|$ values for all datasets are reported in Table 5. As expected, we observe high $|\lambda|$ values for long sequences (e.g., ~0.8 for XLong), whereas for short sequences, the $|\lambda|$ values are significantly lower (e.g., ~0.3 for Beauty and Video). Interestingly, we observe relatively high $|\lambda|$ on Sports despite its short sequence length, which may explain for the significant performance improvement of LRURec (up to 17.43%) in the main results.

5.4 Ablation Studies

We perform a series of ablations to demonstrate the effectiveness of the proposed components in LRURec. In particular, we remove the layer normalization, residual connection and PFFN net respectively and evaluate the performance changes. We additionally reduce the LRU blocks in LRURec and adopt two efficient sequence modeling methods, S4 and RWKV, to replace LRU [5, 26]. The ablation results are reported in Table 6, with rows representing the ablation variant and columns representing the datasets. We observe the following on the ablation of LRURec components: (1) All components contribute to the overall performance of LRURec. For example, removing PFFN results in an average performance drop of 11.24% on ML-1M. (2) The components contribute differently depending on the datasets. For instance, layer normalization and residual connection contribute significantly to the performance on sparse datasets (e.g., with 82.07% and over 100% average gains on Beauty). In contrast, PFFN improves the modeling of non-linear transition patterns, and thereby further enhances the performance on dense datasets like ML-1M. By additionally switching the backbone of our LRURec, we notice: (1) The overall best performing variant is still our reduced one-layer LRURec, demonstrating the effectiveness of the proposed architecture. On average, the one-layer LRURec outperforms the second-best backbone variant by 7.70% on Recall@10. (2) Surprisingly, the S4 variant performs the best on the Steam dataset, which may be attributed to the combination of the linear design of S4 and the imbalanced item popularity in Steam. Overall, the ablation results suggest that all proposed components and the carefully designed architecture in LRURec are effective for sequential recommendation across various data scenarios.

5.5 Model Efficiency

To further demonstrate the advantages of our design, we study the efficiency of LRURec against two representative baselines: RNN-based GRU4Rec and transformer-based SASRec. We illustrate validation Recall@10 curves during training in Figure 1 (left), with the horizontal axis representing training steps and the vertical axis representing Recall@10 scores. Owning to the linear recurrence design and improved training dynamics, LRURec converges significantly faster and triggers early stopping at ~23k training steps, compared to over 50k of SASRec and over 80k of GRU4Rec. Aside from training, LRURec also demonstrates substantial advantages with incremental inference that drastically reduces latency for real-time recommendation, as illustrated in Figure 1 (right). Here, we perform batched inference and keep extending input length after each prediction step, we visualize the cumulative prediction time for a total of 2048 steps. Given hidden states and current input elements, we observe almost linear correlation between the cumulative computing time and increasing input steps on both GRU4Rec and LRURec. Moreover, the throughput capacity of LRURec is independent of the the sequence length in incremental inference. For example, LRURec can achieve ×7.3 increased input examples compared to SASRec with maximum sequence length of 50. In summary, the results suggest that LRURec has the benefits in training parallelization and performance like transformer models, while retaining the advantage of incremental inference from RNNs.

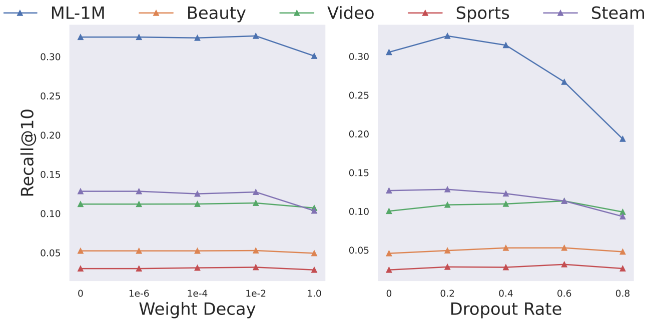

Figure 4: Hyperparameter sensitivity of LRURec.

5.6 Hyperparameter Sensitivity

We evaluate the hyperparameter sensitivity of LRURec. In particular, we vary weight decay and dropout values to evaluate the trained models with the best validation performance. Figure 4 compares the performance with different hyperparameters, with the x-axis stands for the varying values and y-axis stands for Recall@10 scores. For weight decay, we observe minor changes with increasing penalty strength, the performance remains robust until increasing weight decay to 1. For varying dropout rates, we observe different performance changes depending on the datasets. For dense datasets (e.g., ML-1M), the best performance is achieved at 0.2 and then consistently reduces with increasing dropout rates. Unlike dense datasets, Recall@10 performance peaks at 0.6 dropout rate on sparse datasets like Video and Beauty. Overall, the performance of LRURec is quite robust with varying weight decay values, while dropout rates should be carefully selected for optimal performance.

6 CONCLUSION

In this paper, we propose a novel sequential recommender called LRURec. The proposed model introduces: (1) linear recurrence with matrix diagonalization to efficiently capture user transition patterns; and (2) a recursive parallelization framework that significantly accelerates training and inference. Moreover, LRURec is designed with a series of improvements for optimal recommendation performance and inference efficiency. We demonstrate the effectiveness and efficiency of LRURec by performing extensive experiments on multiple real-world datasets, where LRURec consistently achieves superior results over state-of-the-art baselines.

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer Normalization. (2016).

[2] Guy E Blelloch. 1989. Scans as primitive parallel operations. *IEEE Transactions on computers* 38, 11 (1989), 1526–1538.

[3] Guy E Blelloch. 1990. Prefix sums and their applications. (1990).

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota.

[5] Albert Gu, Karan Goel, and Christopher Re. 2021. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*.

[6] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. 2021. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems* 34 (2021), 572–585.

[7] F Maxwell Harper and Joseph A Konstan. 2015. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)* 5, 4 (2015), 1–19.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[9] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 191–200.

[10] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.

[11] Zhankui He, Handong Zhao, Zhe Lin, Zhaowen Wang, Ajinkya Kale, and Julian McAuley. 2021. Locker: Locally constrained self-attentive sequential recommendation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 3088–3092.

[12] Zhankui He, Handong Zhao, Zhaowen Wang, Zhe Lin, Ajinkya Kale, and Julian Mcauley. 2022. Query-Aware Sequential Recommendation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 4019–4023.

[13] Balázs Hidasi and Alexandros Karatzoglou. 2018. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM international conference on information and knowledge management*. 843–852.

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

[15] Santosh Kabbur, Xia Ning, and George Karypis. 2013. Fism: factored item similarity models for top-n recommender systems. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 659–667.

[16] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.

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

[18] Richard E Ladner and Michael J Fischer. 1980. Parallel prefix computation. *Journal of the ACM (JACM)* 27, 4 (1980), 831–838.

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

[20] Jiacheng Li, Tong Zhao, Jin Li, Jim Chan, Christos Faloutsos, George Karypis, Soo-Min Pantel, and Julian McAuley. 2022. Coarse-to-fine sparse sequential recommendation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2082–2086.

[21] Muyang Li, Zijian Zhang, Xiangyu Zhao, Wanyu Wang, Minghao Zhao, Runze Wu, and Ruocheng Guo. 2023. AutoMLP: Automated MLP for Sequential Recommendations. In *Proceedings of the ACM Web Conference 2023*. 1190–1198.

[22] Muyang Li, Xiangyu Zhao, Chuan Lyu, Minghao Zhao, Runze Wu, and Ruocheng Guo. 2022. MLP4Rec: A Pure MLP Architecture for Sequential Recommendations. In *31st International Joint Conference on Artificial Intelligence and the 25th European Conference on Artificial Intelligence (IJCAI-ECAI 2022)*. International Joint Conferences on Artificial Intelligence, 2138–2144.

[23] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*. 43–52.

[24] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. 2023. Resurrecting recurrent neural networks

for long sequences. *arXiv preprint arXiv:2303.06349* (2023).

[25] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. 2022. PinnerFormer: Sequence Modeling for User Representation at Pinterest. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3702–3712.

[26] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. 2023. RWKV: Reinventing RNNs for the Transformer Era. *arXiv preprint arXiv:2305.13048* (2023).

[27] Kan Ren, Jiarui Qin, Yuchen Fang, Weinan Zhang, Lei Zheng, Weijie Bian, Guorui Zhou, Jian Xu, Yong Yu, Xiaoqiang Zhu, et al. 2019. Lifelong sequential modeling with personalized memorization for user response prediction. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 565–574.

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

[29] Harald Steck, Linas Baltrunas, Ehtsham Elahi, Dawen Liang, Yves Raimond, and Justin Basilico. 2021. Deep learning for recommender systems: A Netflix case study. *AI Magazine* 42, 3 (2021), 7–18.

[30] Jianlin Su. 2023. *Google's new work tries to "resurrect" RNN: Can RNN regain attention?* https://spaces.ac.cn/archives/9554

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

[32] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive Network: A Successor to Transformer for Large Language Models. *arXiv preprint arXiv:2307.08621* (2023).

[33] Jiaxi Tang and Ke Wang. 2018. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 565–573.

[34] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020. Long Range Arena: A Benchmark for Efficient Transformers. In *International Conference on Learning Representations*.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[36] An Yan, Shuo Cheng, Wang-Cheng Kang, Mengting Wan, and Julian McAuley. 2019. CosRec: 2D convolutional neural networks for sequential recommendation. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 2173–2176.

[37] Zhenrui Yue, Zhankui He, Huimin Zeng, and Julian McAuley. 2021. Black-box attacks on sequential recommenders via data-free model extraction. In *Proceedings of the 15th ACM Conference on Recommender Systems*. 44–54.

[38] Zhenrui Yue, Huimin Zeng, Ziyi Kou, Lanyu Shang, and Dong Wang. 2022. Defending substitution-based profile pollution attacks on sequential recommenders. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 59–70.

[39] Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. 2021. An attention free transformer. *arXiv preprint arXiv:2105.14103* (2021).

[40] Kun Zhou, Hui Yu, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Filter-enhanced MLP is all you need for sequential recommendation. In *Proceedings of the ACM web conference 2022*. 2388–2399.