



PDF Download  
3705328.3748060.pdf  
03 January 2026  
Total Citations: 3  
Total Downloads: 2190

Latest updates: <https://dl.acm.org/doi/10.1145/3705328.3748060>

RESEARCH-ARTICLE

## Test-Time Alignment with State Space Model for Tracking User Interest Shifts in Sequential Recommendation

CHANGSHUO ZHANG, Renmin University of China, Beijing, China

XIAO ZHANG, Renmin University of China, Beijing, China

TENG SHI, Renmin University of China, Beijing, China

JUN XU, Renmin University of China, Beijing, China

JIRONG WEN, Renmin University of China, Beijing, China

Open Access Support provided by:

Renmin University of China

Published: 22 September 2025

[Citation in BibTeX format](#)

RecSys '25: Nineteenth ACM Conference  
on Recommender Systems  
September 22 - 26, 2025  
Prague, Czech Republic

Conference Sponsors:  
SIGCHI

# Test-Time Alignment with State Space Model for Tracking User Interest Shifts in Sequential Recommendation

Changshuo Zhang  
Gaoling School of Artificial  
Intelligence, Renmin University of  
China  
Beijing, China  
lyingcs@ruc.edu.cn

Xiao Zhang\*  
Gaoling School of Artificial  
Intelligence, Renmin University of  
China  
Beijing, China  
zhangx89@ruc.edu.cn

Teng Shi  
Gaoling School of Artificial  
Intelligence, Renmin University of  
China  
Beijing, China  
shiteng@ruc.edu.cn

Jun Xu  
Gaoling School of Artificial  
Intelligence, Renmin University of  
China  
Beijing, China  
junxu@ruc.edu.cn

Ji-Rong Wen  
Gaoling School of Artificial  
Intelligence, Renmin University of  
China  
Beijing, China  
jrwen@ruc.edu.cn

## Abstract

Sequential recommendation is essential in modern recommender systems, aiming to predict the next item a user may interact with based on their historical behaviors. However, real-world scenarios are often dynamic and subject to shifts in user interests. Conventional sequential recommendation models are typically trained on static historical data, limiting their ability to adapt to such shifts and resulting in significant performance degradation during testing. Recently, Test-Time Training (TTT) has emerged as a promising paradigm, enabling pre-trained models to dynamically adapt to test data by leveraging unlabeled examples during testing. However, applying TTT to effectively track and address user interest shifts in recommender systems remains an open and challenging problem. Key challenges include how to capture temporal information effectively and explicitly identifying shifts in user interests during the testing phase. To address these issues, we propose **T<sup>2</sup>ARec**, a novel model leveraging state space model for TTT by introducing two **Test-Time Alignment** modules tailored for sequential recommendation, effectively capturing the distribution shifts in user interest patterns over time. Specifically, **T<sup>2</sup>ARec** aligns absolute time intervals with model-adaptive learning intervals to capture temporal dynamics and introduce an interest state alignment mechanism to effectively and explicitly identify the user interest shifts with theoretical guarantees. These two alignment modules enable efficient and incremental updates to model parameters in a self-supervised manner during testing, enhancing predictions for online recommendation. Extensive evaluations on three benchmark datasets

demonstrate that **T<sup>2</sup>ARec** achieves state-of-the-art performance and robustly mitigates the challenges posed by user interest shifts.

## CCS Concepts

• **Information systems** → **Recommender systems**.

## Keywords

Test-Time Alignment, Sequential Recommendation, User Interest Shifts, State Space Model

## ACM Reference Format:

Changshuo Zhang, Xiao Zhang, Teng Shi, Jun Xu, and Ji-Rong Wen. 2025. Test-Time Alignment with State Space Model for Tracking User Interest Shifts in Sequential Recommendation. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems (RecSys '25)*, September 22–26, 2025, Prague, Czech Republic. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3705328.3748060>

## 1 Introduction

Sequential recommendations aim to predict the next item a user will interact with by modeling dependencies within their historical interaction data [1, 17, 19, 44, 49]. However, real-world scenarios often present the challenge of distribution shifts, where user behavior patterns and data distributions evolve dynamically over time. For instance, consider a sequential recommendation where a user's interaction history (as shown in Figure 1) during the training phase (weekdays) exhibits a strong preference for study-related items, such as books, reflecting their focus on work or learning. However, during the testing phase (weekends), the user's interest shifts towards sports-related items, such as football or basketball equipment, as they transition to leisure activities. If the recommendation fails to adapt to this shift and continues to recommend study-related items based on the training phase, it will not align with the user's weekend preferences.

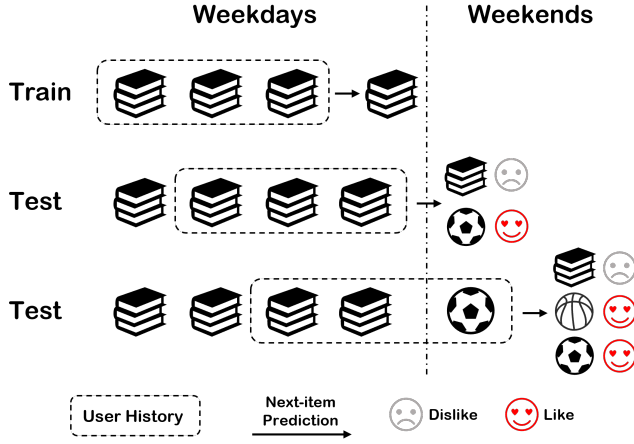
Existing sequential recommendation models, typically trained on static historical data and having their model parameters fixed during online deployment, face challenges in adapting to shifts in user interest patterns. This limitation often leads to significant

\*Xiao Zhang is the corresponding author (e-mail: zhangx89@ruc.edu.cn).

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

RecSys '25, Prague, Czech Republic

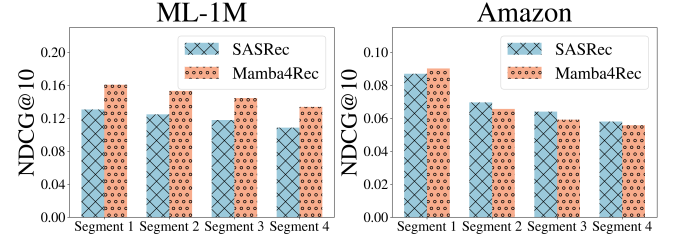
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 979-8-4007-1364-4/25/09  
<https://doi.org/10.1145/3705328.3748060>



**Figure 1: Illustrates of user interest shifts between train and testing phases in sequential recommendation. During the training phase (weekdays), the user’s historical behavior is focused on study-related content (e.g., books), and the model learns to predict the next item based on this pattern. In the testing phase (weekends), user interest shifts from study-related items to leisure activities (e.g., sports). Although the Test input includes the ground truth from the training phase, the model continues to recommend study-related content, failing to adapt to the user’s new preferences. This highlights the importance of modeling temporal contexts and handling user interest shifts effectively in recommendations.**

performance degradation during test time. To validate this phenomenon, we conducted experiments on two datasets (ML-1M [14] and Amazon Prime Pantry [28]) using two models (SASRec [19] and Mamba4Rec [24]) following the implementation settings in Recbole [60]. After training the models on the training set, the testing set was evenly divided into four segments based on timestamps, and NDCG@10 was used as the evaluation metric. As shown in Figure 2, the later the timestamp of the segment, the more significant the user interest shift, resulting in poorer test performance metrics. This illustrates how user behavior evolves dynamically between the train and testing phases, driven by contextual factors such as time availability, necessitating adaptive models to handle such user interest shifts effectively.

Motivated by the recent promising paradigm of Test-Time Training (TTT) [18, 22, 27, 42, 47], which enables pre-trained models to dynamically adapt to test data by leveraging unlabeled examples during inference, we focus on applying TTT to track user interest shifts in sequential recommendation. While TTT facilitates self-adaptation and effectively addresses evolving distribution shifts in an online manner, its application to tracking user interest shifts in recommender systems remains an open and challenging problem. We identify two key challenges associated with this task: first, *capturing temporal information*, as user behavior is often influenced by periodic patterns or trending topics, making it crucial to understand the impact of historical events on predictions at specific future time points; second, *dynamically and efficiently adjusting the representation of user interest patterns*, since even if a user’s



**Figure 2: Validation of user interest shifts during the testing phase. We conducted experiments on two datasets (ML-1M and Amazon Prime Pantry) using two backbone models (SASRec and Mamba4Rec). After training the models on the training set, the testing set was evenly divided into four segments based on timestamps, and NDCG@10 was used as the evaluation metric for analysis and comparison.**

historical behavioral features remain stable, their interest patterns may evolve dynamically. The model needs to identify and adapt to these changes to provide accurate recommendations during testing.

To address the above challenges of tracking user interest shifts in sequential recommendation tasks, we propose  $T^2ARec$ , a sequential model that integrates test-time training.  $T^2ARec$  adopts state space models (SSMs) [4, 9] as the backbone, which effectively models user interest state transitions by handling historical interactions and resolves test-time throughput issues. Then we introduce two alignment-based self-supervised losses to adaptively capture the user interest shifts. The time interval alignment loss computes differences between interaction intervals in the sequence and the target prediction time, aligning these intervals with adaptive time steps to effectively capture temporal dynamics. The interest state alignment loss models the dynamic evolution of user interest patterns by transforming the input sequence into a final state, applying forward and backward state updates to generate a reconstructed state, and aligning it with the original state for precise pattern modeling. During testing,  $T^2ARec$  applies gradient descent on test data to adjust model parameters in real time, enabling accurate next-item predictions under distribution shifts. This design allows  $T^2ARec$  to effectively capture temporal dynamics and adapt to evolving user interest patterns, ensuring robust and efficient performance in sequential recommendation tasks at the test time.

Our main contributions are as follows:

- Identification of a key issue in applying Test-Time Training to sequential recommendations: The overlooked shift in the user interest pattern at the test time.
- Introduction of a novel approach: We propose  $T^2ARec$ , a TTT-based sequential recommendation model incorporating two test-time alignment-based losses in a state space model to capture temporal dynamics and evolving user interest patterns, along with real-time parameter adjustment during testing to track user interest shifts and ensure robust performance.
- Extensive experiments: We conduct experiments on three widely used datasets, demonstrating the effectiveness of  $T^2ARec$ . Further ablation studies and analysis explain the superiority of our designed modules.

## 2 Related Work

### 2.1 Sequential Recommendation

Sequential recommendations have evolved from traditional models, like Markov Chains [16], to deep learning approaches [3, 7, 39, 40, 43, 55, 57, 59]. Early RNN-based models (e.g., GRU4Rec [17], HRNN [33]) addressed long-range dependencies, leveraging hidden states to capture dynamic user preferences. Recently, Transformer-based architectures, such as SASRec [19] and BERT4Rec [41], have gained prominence with self-attention mechanisms that model complex interactions and enable efficient parallel computation. Innovations like Mamba4Rec[24] further enhance performance on long interaction sequences, reflecting the ongoing advancements in this field. Recent works also leverage Large Language Models (LLMs) for sequential recommendation [32, 51], relying primarily on the static pre-trained knowledge. Our approach, however, focuses on test-time alignment to explicitly track and adapt to user interest shifts during inference.

### 2.2 Test-Time Training

Test-Time Training (TTT) [18, 27, 36, 47] improves model generalization by enabling partial adaptation during the testing phase to address distribution shifts between training and testing datasets. It leverages self-supervised learning (SSL) [25, 26, 31, 35, 53] tasks to optimize a supervised loss (e.g., cross-entropy) and an auxiliary task loss during training. The auxiliary task (e.g., rotation prediction) allows the model to align test-time features closer to the source domain. Simplified approaches like Tent [46] avoid supervised loss optimization during testing, while advanced methods such as TTT++ [27] and TTT-MAE [15] employ techniques like contrastive learning and masked autoencoding to enhance adaptation. Unsupervised extensions, such as ClusT3 [12], use clustering with mutual information maximization but face limitations due to hyperparameter sensitivity. Test-Time Training (TTT) has been applied to out-of-distribution (OOD) tasks, such as recommendations [37, 48, 52, 54]. DT3OR [52] introduces a model adaptation mechanism during the test phase, specifically designed to adapt to the shifting user and item features. TTT4Rec [54] leverages SSL in an inner-loop, enabling real-time model adaptation for sequential recommendations. Paragon [38] leverages a generative model to produce a portion of the recommendation model's parameters during the test phase, ensuring test-time controllability [36]. LAST [48] constructs simulated feedback through an evaluator during testing, updating model parameters, and achieving online benefits. However, existing work has not effectively captured temporal information or explicitly identified shifts in user interests during the testing phase.

## 3 Preliminaries

### 3.1 Problem Statement

In sequential recommendation, let  $\mathcal{U} = \{u_1, u_2, \dots, u_{|\mathcal{U}|}\}$  denote the user set,  $\mathcal{V} = \{v_1, v_2, \dots, v_{|\mathcal{V}|}\}$  denote the item set, and  $\mathcal{S}_u = [v_1, v_2, \dots, v_{n_u}]$  denote the chronologically ordered interaction sequence for user  $u \in \mathcal{U}$  with the corresponding timestamp sequence  $[t_1, t_2, \dots, t_{n_u}]$ , where  $n_u$  is the length of the sequence,  $v_i$  is the  $i$ -th item interacted with by user  $u$ , and  $t_i$  is the timestamp of the

interaction. Given the interaction history  $\mathcal{S}_u$  and the timestamp of prediction  $t_{n_u+1}$ , the task is to predict the next interacted item  $v_{n_u+1}$ , i.e., the item that the user  $u$  is most likely to interact with at timestamp  $t_{n_u+1}$ . This can be formalized as learning a function:

$$f_{t_{n_u+1}} : \mathcal{S}_u \rightarrow v_{n_u+1}, \quad (1)$$

where  $f_{t_{n_u+1}}$  maps the historical sequence  $\mathcal{S}_u$  at timestamp  $t_{n_u+1}$  to the next likely item  $v_{n_u+1}$  from the item set  $\mathcal{V}$ . In the following sections, we omit the subscript  $u$  in  $n_u$  and  $\mathcal{S}_u$  and directly use  $n$  and  $\mathcal{S}$  for convenience.

### 3.2 State Space Models (SSMs)

SSMs perform well in long-sequence modeling [24], image generation [50], and reinforcement learning [29], providing efficient autoregressive inference like RNN [23] while processing input sequences in parallel like Transformers [45]. This dual functionality enables efficient training and robust performance in applications such as time series analysis [34] and audio generation [8].

The original SSMs originated as continuous-time maps on functions from  $d$ -dimensional input  $\mathbf{x}(t) \in \mathbb{R}^d$  to output  $\mathbf{y}(t) \in \mathbb{R}^d$  at current time  $t$  through a  $d_s$ -dimensional hidden state  $\mathbf{h}(t) \in \mathbb{R}^{d_s}$ . These models leverage the dynamics described below:

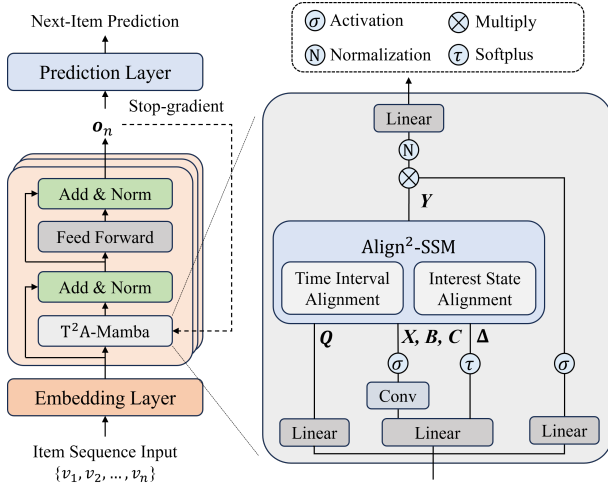
$$\mathbf{h}'(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t), \quad (2a)$$

$$\mathbf{y}(t) = \mathbf{C}^\top \mathbf{h}(t), \quad (2b)$$

where  $\mathbf{A} \in \mathbb{R}^{d_s \times d_s}$  and  $\mathbf{B}, \mathbf{C} \in \mathbb{R}^{d_s \times d}$  are adjustable matrices,  $\mathbf{h}'(t)$  denotes the derivative of  $\mathbf{h}(t)$ . To enable effective representation of discrete data, Structured SSMs [10] employ the Zero-Order Hold (ZOH) [11] method for data discretization from the input sequence  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}$  to output sequence  $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n]^\top \in \mathbb{R}^{n \times d}$  through hidden state  $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n]^\top \in \mathbb{R}^{n \times d_s}$  based on a specified step size  $\Delta \in \mathbb{R}$ . The introduction of the Mamba [9] model significantly enhances SSMs by dynamically adjusting the matrices  $\mathbf{B} \in \mathbb{R}^{n \times d_s}$ ,  $\mathbf{C} \in \mathbb{R}^{n \times d_s}$  and the step size now represented as  $\Delta \in \mathbb{R}^{n \times d}$  that dynamically depends on inputs varying over time. Its latest version, Mamba-2 [4], links structured state space models with attention mechanisms. Mamba-2 refines  $\mathbf{A}$  into a scalar value  $A \in \mathbb{R}$  and sets  $\Delta \in \mathbb{R}^n$ , creating a new state space duality (SSD) framework with multi-head patterns akin to Transformers. This innovation boosts training speed and increases state size, optimizing expressiveness for greater efficiency and scalability, making it a strong contender against Transformers.

## 4 T<sup>2</sup>ARec: The Proposed Method

To track shifts in user interest during the testing phase of sequential recommendation tasks, we propose T<sup>2</sup>ARec, a method that integrates a state space model for test-time training and introduces two alignment modules to adaptively capture these shifts during testing. First, we introduce the base model of the proposed method, which is developed based on the Mamba-2 architecture and can dynamically represent user interest with high test-time throughput. Next, we describe the two alignment modules, designed for time intervals and user interest states, respectively. Finally, we explain how to conduct self-supervised training during the testing phase.



**Figure 3: Overall framework of T<sup>2</sup>ARec:** T<sup>2</sup>ARec processes input sequences through an embedding layer, followed by the T<sup>2</sup>A-Mamba block and Align<sup>2</sup>-SSM block for state updates and output generation. The prediction layer uses output embedding  $o_n$  generated from the feed forward network layer for next-item predictions.  $o_n$  is reintroduced into the T<sup>2</sup>A-Mamba block to compute the alignment losses.

#### 4.1 Base Model

As illustrated in Figure 3, we develop the basic framework of T<sup>2</sup>ARec by stacking the embedding layer, T<sup>2</sup>A-Mamba block, and the prediction layer.

**4.1.1 Embedding Layer.** Given a learnable embedding layer  $E = [e_1, e_2, \dots, e_{|\mathcal{V}|}]^T \in \mathbb{R}^{|\mathcal{V}| \times d}$  for all items, where  $d$  is the embedding dimension and  $e_j$  represents the dense vector for item  $v_j$ , this layer transforms the sparse item id sequence  $\mathcal{S} = [v_1, v_2, \dots, v_n]$  into dense vector representations, denoted as  $E_S^{(1)} \in \mathbb{R}^{n \times d}$ .

**4.1.2 T<sup>2</sup>A-Mamba Block.** Though Transformer-based models excel in sequential recommendation [19], their quadratic complexity related to sequence length impedes efficient test-time training and burdens throughput. To address this issue, we develop T<sup>2</sup>A-Mamba based on Mamba-2 [4].

The output representations  $E_S$  of the embedding layer then enter our core T<sup>2</sup>A-Mamba Block. The sequence undergoes a series of transformations to generate inputs for Align<sup>2</sup>-SSM block. Specifically,  $E_S$  first passes through a linear layer:

$$E_S^{(2)}, E_S^{(3)} \leftarrow \text{Linear}_1(E_S^{(1)}), \quad (3)$$

where  $E_S^{(2)} \in \mathbb{R}^{n \times (d+2 \times d_s)}$ ,  $E_S^{(3)} \in \mathbb{R}^n$  and  $\text{Linear}_1$  is a linear parameterized projection from dimension  $d$  to dimension  $(d + 2 \times d_s + 1)$ . Next,  $E_S^{(2)}$  passes through a convolution and a non-linear activation,  $E_S^{(3)}$  passes through a softplus function:

$$X, B, C \leftarrow \sigma(\text{Conv}(E_S^{(2)})), \quad \Delta = \tau(E_S^{(3)}) \quad (4)$$

where  $X = [x_1, x_2, \dots, x_n]^T \in \mathbb{R}^{n \times d}$  is the input of Align<sup>2</sup>-SSM,  $B \in \mathbb{R}^{n \times d_s}$  and  $C \in \mathbb{R}^{n \times d_s}$  are the adjustable matrices,  $\Delta \in \mathbb{R}_+^n$  is the step size ensured to be positive via the softplus function  $\tau$ , Conv is the convolution operation,  $\sigma$  is the non-linear activation.

The overall transformation process of Equation (3) and (4) can be unified into a sequence of mappings from the input embeddings  $E_S$  to the outputs  $X, B, C$ , and the step size  $\Delta$ , denotes as:

$$X, B, C, \Delta \leftarrow \text{Transform}(E_S^{(1)}). \quad (5)$$

Then, we introduce the core ingredient of T<sup>2</sup>A-Mamba, the Align<sup>2</sup>-SSM. Given a learnable scalar value  $A < 0$ , we compute the discretized  $\bar{A} = [\bar{A}_1, \bar{A}_2, \dots, \bar{A}_n]^T \in \mathbb{R}^n$  and  $\bar{B} = [\bar{B}_1, \bar{B}_2, \dots, \bar{B}_n]^T \in \mathbb{R}^{n \times d_s}$  follows the Zero-Order Hold (ZOH) [11] method, formally:

$$\bar{A} = e^{A\Delta}, \quad \bar{B} = \text{diag}(\Delta)B, \quad (6)$$

simplified as  $\bar{A}, \bar{B} \leftarrow \text{discretize}(\Delta, A, B)$ , where  $\text{diag}(\Delta)$  is a  $n \times n$  diagonal matrix, the diagonal elements are the elements of  $\Delta$ .

Finally, given an all-zero matrix  $h_0 \in \mathbb{R}^{d_s \times d}$  as the initial state, we can iteratively compute the outputs and hidden states in Align<sup>2</sup>-SSM:

$$h_t = \bar{A}_t h_{t-1} + \bar{B}_t \otimes x_t, \quad (7a)$$

$$y_t = h_t^T C_t, \quad (7b)$$

where  $\otimes$  represents the outer product, which combines  $\bar{B}_t \in \mathbb{R}^{d_s}$  and  $x_t \in \mathbb{R}^d$  into a matrix of size  $d_s \times d$ ,  $Y = [y_1, y_2, \dots, y_n]^T \in \mathbb{R}^{n \times d}$ ,  $H = [h_1, h_2, \dots, h_n]^T \in \mathbb{R}^{n \times d \times d_s}$  are the outputs and hidden states of Align<sup>2</sup>-SSM block, separately, and  $h_n \in \mathbb{R}^{d_s \times d}$  is referred to as the *final state* that characterizes the user's current preference.

**4.1.3 Feed Forward Network Layer and Prediction Layer.**  $Y$  is then passed through a Feed Forward Network (FFN) layer to adapt the features to the semantic space of the next layer:

$$O = \text{FFN}(Y), \quad (8)$$

where  $O = [o_1, o_2, \dots, o_n]^T \in \mathbb{R}^{n \times d}$  is the output embeddings. In addition, this process involves residual connections, layer normalization, and other transformations, which are not explicitly represented in the equations for the sake of simplicity.

Based on the output embeddings  $O$  generated by the FFN. We use the last element  $o_n$  to predict the next item the user is likely to interact with. The prediction layer computes logits for all items as:

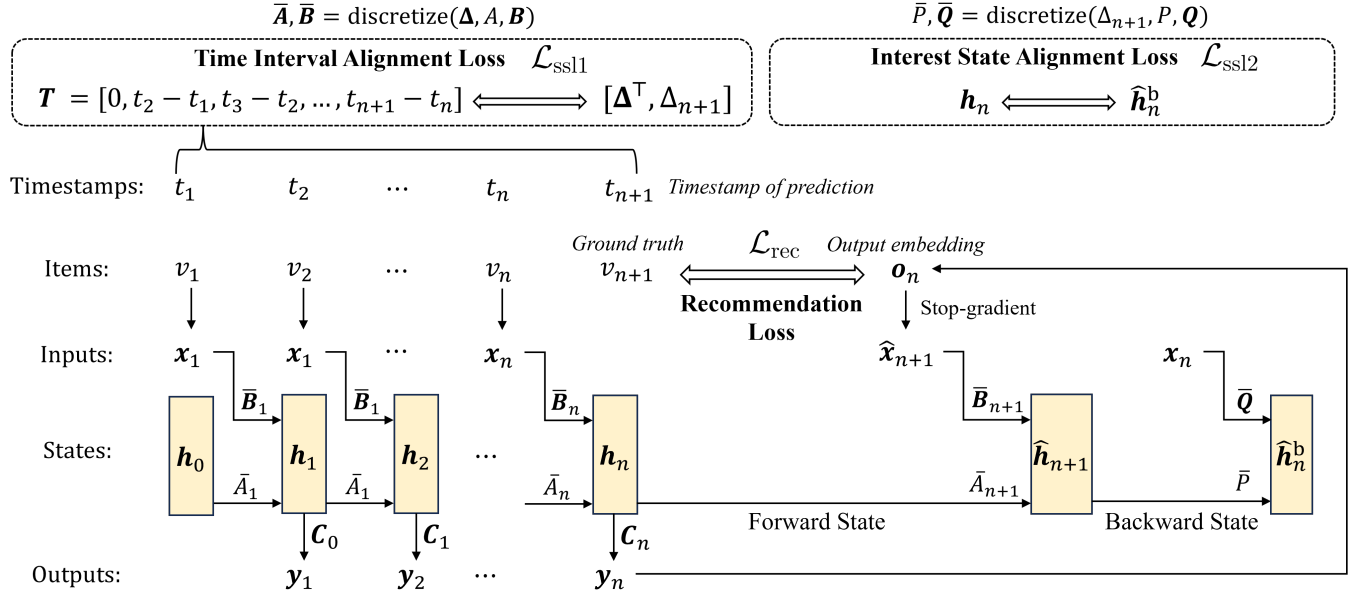
$$\hat{z} = \text{Softmax}(E o_n), \quad (9)$$

where  $\hat{z} \in \mathbb{R}^{|\mathcal{V}|}$ , Softmax is the softmax function and  $E$  is the embedding table of the embedding layer.

**4.1.4 Recommendation Loss.** The recommendation loss is then computed using the cross-entropy loss:

$$\mathcal{L}_{\text{rec}} = -\frac{1}{n} \sum_{i=1}^n z_i \log(\hat{z}_i), \quad (10)$$

where  $z_i$  denotes the ground-truth for item  $i$ ,  $\hat{z}_i$  denotes the predicted logit for item  $i$  in  $\hat{z}$ . This loss encourages the model to maximize the predicted probability of the true interacted item, improving recommendation accuracy.



**Figure 4: The logits and losses computation in Align<sup>2</sup>-SSM:** The left side illustrates the time interval alignment loss ( $\mathcal{L}_{\text{time}}$ ), which aligns the predicted time intervals  $\Delta$  with the ground truth  $T$ . The right side shows the interest state alignment loss ( $\mathcal{L}_{\text{state}}$ ), aligning the final state  $h_n$  with the backward state  $\hat{h}_n^b$ . These two losses jointly enhance the model's robustness and effectiveness in handling user interest shifts. During testing, the model leverages these self-supervised losses to perform gradient descent, adapting to the input data and improving prediction performance.

## 4.2 Time Interval Alignment

To more accurately capture the temporal information in distribution shifts, we introduce a time interval alignment loss in T<sup>2</sup>ARec. In this process, we not only consider relative time but also incorporate absolute timestamps to better reflect the dynamics of temporal information. As shown in the left half of Figure 4, we illustrate the computation method for the time interval alignment loss. In the Align<sup>2</sup>-SSM block, the discretization process (Equation (20)) involves the use of time steps  $\Delta = [\Delta_1, \Delta_2, \dots, \Delta_n]^T \in \mathbb{R}_+^n$ , where  $\Delta_i \in \mathbb{R}_+$ . Ensuring the correctness of the time steps is crucial for alleviating user interest shifts, as accurate time steps can better capture the temporal information embedded in distribution shifts.

As discussed in Section 4.1.2, the time steps  $\Delta$  in the T<sup>2</sup>A-Mamba block are not fixed but are dynamically generated based on the input sequence  $S$  through the embedding layer output  $E_S$ . This dynamic adjustment enables the model to adapt to variations in input features, resulting in a more flexible temporal representation. However,  $\Delta$  can only adaptively learn the temporal information in timestamp sequence  $[t_1, t_2, \dots, t_n]$ , while ignoring the prediction timestamp  $t_{n+1}$ . To address this, we aim to obtain an adaptive test-time temporal representation  $\Delta_{n+1} \in \mathbb{R}_+$  in Align<sup>2</sup>-SSM.

To achieve this, although we cannot directly access the ground truth item  $v_{n+1}$  during prediction, we consider the output embedding of the feed forward network layer  $\mathbf{o}_n \in \mathbb{R}^d$ . This embedding is used in the prediction layer to compute similarity scores with all item embeddings via a dot product (as detailed in Section 4.1.3). During training,  $\mathbf{o}_n$  is expected to progressively converge toward the embedding representation of the next ground truth item  $v_{n+1}$ .

Based on this observation, we re-feed  $\mathbf{o}_n$  into the T<sup>2</sup>A-Mamba block to estimate the adaptive time step  $\Delta_{n+1}$  using Equation (5) and apply stop-gradient to it to prevent unstable training:

$$\hat{\mathbf{x}}_{n+1}, B_{n+1}, C_{n+1}, \Delta_{n+1} \leftarrow \text{Transform}(\mathbf{o}_n)_{\text{stop-gradient}}. \quad (11)$$

After obtaining the adaptive time steps  $\Delta$  and  $\Delta_{n+1}$ , we demonstrate how to align them with the timestamps  $t_1, t_2, \dots, t_{n+1}$ . First, we compute the time interval sequence based on these timestamps and pad it with 0 at the beginning (since the timestamps of interactions prior to the input sequence are unavailable):

$$T = [0, t_2 - t_1, t_3 - t_2, \dots, t_{n+1} - t_n], \quad (12)$$

where  $T \in \mathbb{R}^{n+1}$  serves as the ground truth for  $\Delta$  and  $\Delta_{n+1}$ .

Then we propose using a pairwise loss to align  $\Delta$  and  $\Delta_{n+1}$  with the ground truth time interval  $T$ , excluding the padded 0. Specifically, we compute the pairwise self-supervised loss as follows:

$$\mathcal{L}_{\text{pairwise}} = \sum_{2 \leq i < j \leq n+1} \max \left\{ 0, 1 - (\Delta_i - \Delta_j) \cdot \left( \frac{T_i - T_j}{\lambda} \right) \right\}, \quad (13)$$

where  $(\Delta_i - \Delta_j)$  represents the predicted pairwise differences,  $(T_i - T_j)$  represents the ground truth pairwise differences,  $\lambda$  is constant value for scaling. The loss penalizes mismatches between the relative signs of predicted and ground truth time differences, preserving the relative relationships between the learned time intervals instead of relying on their absolute values.

However, directly applying this loss introduces certain challenges. When handling long sequences (e.g., 50–200 items), the computational complexity of pairwise difference calculations increases to  $O(n^2)$ . To address this issue, we adopt a block-based

computation approach to simplify the process. Specifically, the sequence is divided into smaller, non-overlapping blocks of size  $b$ , and pairwise losses are computed independently within each block. For each block, we first calculate the pairwise differences for predicted values  $(\Delta_i - \Delta_j)$  and ground truth values  $(T_i - T_j)$ . Next, we apply a mask to exclude invalid pairs, accommodating sequences of varying lengths. Finally, the *time interval alignment loss* for each block is computed using the hinge loss formulation and normalized by the total number of valid pairs across all blocks:

$$\mathcal{L}_{\text{time}} = \frac{\sum_{\text{block}} \sum_{2 \leq i < j \leq n+1} \max \left\{ 0, 1 - (\Delta_i - \Delta_j) \cdot \left( \frac{T_i - T_j}{\lambda} \right) \right\}}{\text{Total Valid Pairs}}. \quad (14)$$

This approach reduces both computational and memory complexity from  $O(n^2)$  to  $O(b^2 \times \lceil n/b \rceil)$ , significantly enhancing scalability for long sequences.

### 4.3 User Interest State Alignment

To ensure the model accurately captures and represents the evolving patterns of user interests over time, we introduce a interest state alignment loss in T<sup>2</sup>ARec. As shown in Figure 1, user interest shifts during testing often occur toward the end of the input sequence. Therefore, it is crucial for recommendation models to better understand and align with the user's interests at the tail end. To achieve this, we align the final states of T<sup>2</sup>A-Mamba.

As illustrated on the right side of Figure 4, we present the computation process for the interest state alignment loss. First, we introduce a backward state update function, which is then applied to align the final state of the input user history sequence. Ensuring the correctness of the final states generated by the model is critical for alleviating shifts in user interests.

**4.3.1 Backward State Update Function.** Given the forward state update function  $\mathbf{h}'(t) = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t)$ , similar to Equation (2a), multiplying both sides by  $\mathbf{A}^{-1}$  yields its backward form:

$$\mathbf{h}(t) = \mathbf{A}^{-1}\mathbf{h}'(t) + (-\mathbf{A}^{-1}\mathbf{B}\mathbf{x}(t)), \quad (15)$$

where  $\mathbf{h}'(t)$  denotes the derivative of  $\mathbf{h}(t)$ . To simplify the notation, we define:

$$\mathbf{P}^b = \mathbf{A}^{-1}, \quad \mathbf{Q}^b = -\mathbf{A}^{-1}\mathbf{B}. \quad (16)$$

Substituting Equation (16) into Equation (15) yields

$$\mathbf{h}(t) = \mathbf{P}^b\mathbf{h}'(t) + \mathbf{Q}^b\mathbf{x}(t). \quad (17)$$

For the discrete-time system, we adopt the Zero-Order Hold [11] method for the discretize process with a scalar step size  $\Delta$  as follows:

$$\bar{\mathbf{P}}^b = e^{\Delta\mathbf{P}^b}, \quad \bar{\mathbf{Q}}^b = \Delta\mathbf{Q}^b. \quad (18)$$

This discretize process can be simplified as  $\bar{\mathbf{P}}, \bar{\mathbf{Q}} = \text{discretize}(\Delta, \mathbf{P}, \mathbf{Q})$ . Using this parameterization, Equation (17) can be expressed as a *backward discrete state update function* as follows:

$$\mathbf{h}_t^b = \bar{\mathbf{P}}^b\mathbf{h}_{t+1} + \bar{\mathbf{Q}}^b \otimes \mathbf{x}_t. \quad (19)$$

**4.3.2 Interest State Alignment Loss.** In the Align<sup>2</sup>-SSM block explained in Section 4.1.2, as described in Equation (7a) and (7b), Align<sup>2</sup>-SSM block generates the hidden states  $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_n]^\top$ . We take its final state  $\mathbf{h}_n \in \mathbb{R}^{d_s \times d}$  and subsequently perform the

interest state alignment operation on it. This is because  $\mathbf{h}_n$  encapsulates the entirety of the user's historical information while retaining the most recent behavioral context with maximal fidelity.

To begin, we estimate the forward state for the next step at  $t_{n+1}$  (abbreviated as *forward state*), denoted as  $\hat{\mathbf{h}}_{n+1} \in \mathbb{R}^{d_s \times d}$ , after  $\mathbf{h}_n$ . Following the approach in Equation (11), the feed-forward network layer output  $\mathbf{o}_n$  is transformed to derive  $\hat{\mathbf{x}}_{n+1} \in \mathbb{R}^d$ ,  $\mathbf{B}_{n+1} \in \mathbb{R}^{d_s}$ ,  $\mathbf{C}_{n+1} \in \mathbb{R}^{d_s}$ , and  $\Delta_{n+1} \in \mathbb{R}_+$ . These terms are subsequently used to compute  $\hat{\mathbf{h}}_{n+1}$ . Specifically, the discretized terms  $\bar{\mathbf{A}}_{n+1} \in \mathbb{R}$  and  $\bar{\mathbf{B}}_{n+1} \in \mathbb{R}^{d_s \times d}$  are computed as follows:

$$\bar{\mathbf{A}}_{n+1} = e^{\Delta_{n+1}\mathbf{A}}, \quad \bar{\mathbf{B}}_{n+1} = \Delta_{n+1}\mathbf{B}_{n+1}, \quad (20)$$

Using these, along with  $\mathbf{h}_n$  and  $\hat{\mathbf{x}}_{n+1}$ , we update the forward state:

$$\hat{\mathbf{h}}_{n+1} = \bar{\mathbf{A}}_{n+1}\mathbf{h}_n + \bar{\mathbf{B}}_{n+1} \otimes \hat{\mathbf{x}}_{n+1}. \quad (21)$$

Next, we introduce a *backward state*, denoted as  $\hat{\mathbf{h}}_n^b$ , and evaluate its alignment with the original state  $\mathbf{h}_n$  in Equation (7a) using an additional loss termed *interest state alignment loss*. In simple terms, this involves substituting  $\hat{\mathbf{h}}_{n+1}$  from Equation (21) into the backward discrete state update function in Equation (19), estimate  $\bar{\mathbf{P}}^b$  and  $\bar{\mathbf{Q}}^b$  using neural network, for computing the backward state. More specifically, the process begins by taking  $\mathbf{x}_n \in \mathbb{R}^d$  in Equation (4) as input and passing it through a linear layer to produce the estimated matrix  $\mathbf{Q} \in \mathbb{R}^{d_s}$ :

$$\mathbf{Q} \leftarrow \text{Linear}_2(\mathbf{x}_n). \quad (22)$$

where  $\text{Linear}_2$  is a linear projection from dimension  $d$  to dimension  $d_s$ . Then, for estimating  $\bar{\mathbf{P}}^b$  and  $\bar{\mathbf{Q}}^b$  in Equation (18) with  $\Delta_{n+1}$ , we introduce a scalar value  $P$ , as specified in Theorem 4.1, and compute the discretized  $\bar{\mathbf{P}}$  and  $\bar{\mathbf{Q}}$ :

$$\bar{\mathbf{P}} = e^{\Delta_{n+1}P}, \quad \bar{\mathbf{Q}} = \Delta_{n+1}\mathbf{Q}. \quad (23)$$

Subsequently, substituting Equation (23) into the backward discrete state update function in Equation (19), we derive the following *backward state*:

$$\hat{\mathbf{h}}_n^b = \bar{\mathbf{P}}\hat{\mathbf{h}}_{n+1} + \bar{\mathbf{Q}} \otimes \mathbf{x}_n. \quad (24)$$

Finally, we define the *interest state alignment loss* as follows:

$$\mathcal{L}_{\text{state}} = \frac{\|\mathbf{h}_n - \hat{\mathbf{h}}_n^b\|_2}{\Delta_{n+1}^2}, \quad (25)$$

where the coefficient  $\frac{1}{\Delta_{n+1}^2}$  serves as a dilution term as analyzed in Theorem 4.1. Intuitively, the interest state alignment loss ensures that the model's internal representation of the user's current interest aligns with the expected state derived from the sequence. Specifically, it reconstructs the user's interest state at the end of their interaction sequence and aligns it with a backwardly updated version of the same state, enabling the model to fine-tune its understanding during the testing phase. Theoretically, we provide an upper bound of  $\mathcal{L}_{\text{state}}$  to analyze its effect.

**THEOREM 4.1.** Denote  $\epsilon_n = \mathbf{Q} - \mathbf{Q}_{n+1}^b$  and  $\mathbf{Q}_{n+1}^b = -\mathbf{A}^{-1}\mathbf{B}_{n+1}$ , and let  $P = \frac{\ln(-\mathbf{A}^{-1})}{\Delta_{n+1}}$ , the following upper bound for  $\mathcal{L}_{\text{state}}$  holds:

$$\mathcal{L}_{\text{state}} \leq \Delta_{n+1}^{-2} \|\mathbf{h}_n\|_2 + \Delta_{n+1}^{-1} \|\mathbf{x}_n\|_2 \|\epsilon_n\|_2 + \mathbf{A}^{-1} \|\mathbf{B}_{n+1}\|_2 \left\| \frac{\mathbf{x}_n - \hat{\mathbf{x}}_{n+1}}{\Delta_{n+1}} \right\|_2.$$

**Algorithm 1** Test-Time Alignment for T<sup>2</sup>ARec on a Single Batch

**Input:** Batch of test sequences, denoted as  $\{S_i\}_{i=1}^m$ , where each represented as  $S_i = [v_1, v_2, \dots, v_n]$  with the corresponding timestamp sequence  $[t_1, t_2, \dots, t_n]$  and timestamp of prediction  $t_{n+1}$ , well-trained model  $g_\theta(\cdot)$ , number of training steps  $M$ , learning rate  $\alpha$ , and weight parameters  $\mu_1^{\text{train}}$  and  $\mu_2^{\text{train}}$

**Output:** Final prediction  $\mathbf{o}_n$

- 1: Record the original model parameters:  $\theta_o \leftarrow \theta$
- 2: **for** step = 1, 2,  $\dots$ ,  $M$  **do**
- 3: Forward  $g_\theta(\{S_i\}_{i=1}^m)$  and get the learned step size  $\Delta$ , final state  $\mathbf{h}_n$  and output  $\mathbf{o}_n$
- 4: Compute the time interval sequence  $T$  using Equation (12)
- 5: Calculate  $\mathcal{L}_{\text{time}}$  from Equation (14) using  $\Delta$  and  $T$
- 6: Compute the forward state  $\hat{\mathbf{h}}_{n+1}$  using Equation (21)
- 7: Compute the backward state  $\hat{\mathbf{h}}_n^b$  using Equation (24)
- 8: Calculate  $\mathcal{L}_{\text{state}}$  from Equation (25) using  $\mathbf{h}_n$  and  $\hat{\mathbf{h}}_n^b$
- 9: Update model parameters:

$$\theta \leftarrow \theta - \alpha \nabla_\theta (\mu_1^{\text{test}} \mathcal{L}_{\text{time}} + \mu_2^{\text{test}} \mathcal{L}_{\text{state}})$$

- 10: **end for**
- 11: Compute final prediction  $\mathbf{o}_n$  using the updated  $g_\theta(\{S_i\}_{i=1}^m)$
- 12: Restore the model parameters:  $\theta \leftarrow \theta_o$  for the next batch

**Table 1: Dataset statistics.**

Dataset	#Users	#Items	#Interactions	Avg. Length	Sparsity
ML-1M	6,034	3,706	1,000,209	138.3	95.57%
Amazon	3,834	6,350	471,615	17.04	98.78%
Zhihu-1M	7,974	81,563	999,970	29.03	99.48%

In Theorem 4.1,  $\|\epsilon_n\|_2$  is considered an instance-dependent term that depends on the input  $\mathbf{x}_n$ , and it tends to increase for unseen user interaction  $\mathbf{x}_n$ . The term  $\|(\mathbf{x}_n - \hat{\mathbf{x}}_{n+1})/\Delta_{n+1}\|_2$  aims to ensure consistency between the model and the most recent interaction under dynamic behavior changes, especially when the time interval  $\Delta_{n+1}$  between the user's testing time and their most recent behavior is very short. Similar to how humans continuously update expectations based on new experiences, this alignment mechanism enables the model to dynamically adapt to changes in user behavior, thereby improving recommendation accuracy.

## 4.4 Training and Testing Process

**4.4.1 Training Process.** After designing the two self-supervised losses, the total loss of the model combines the primary recommendation loss  $\mathcal{L}_{\text{rec}}$  in Section 4.1.4 and the two self-supervised losses  $\mathcal{L}_{\text{time}}$  and  $\mathcal{L}_{\text{state}}$ , weighted by their respective hyperparameters:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + \mu_1^{\text{train}} \mathcal{L}_{\text{time}} + \mu_2^{\text{train}} \mathcal{L}_{\text{state}}, \quad (26)$$

where  $\mu_1^{\text{train}}$  and  $\mu_2^{\text{train}}$  are the weights for the self-supervised losses. This combined loss optimizes the model by integrating both the primary task and self-supervised learning objectives.

**4.4.2 Testing Process.** During testing, the pre-trained model  $g_\theta(\cdot)$ 's parameters  $\theta$  are adaptively fine-tuned on all testing examples. For each batch of test data  $\{S_i\}_{i=1}^m$ , optimization is performed using the

self-supervised losses  $\mathcal{L}_{\text{time}}$  and  $\mathcal{L}_{\text{state}}$  as defined in Equation (14) and Equation (25). The trained model is then used to make the final item predictions. After completing the predictions, the adjusted parameters on  $\theta$  are discarded, and the model reverts to its original parameters to process the next batch of testing examples. The pseudo-code for processing a single batch of testing examples is illustrated in Algorithm 1.

## 5 Experiments

To verify the effectiveness of T<sup>2</sup>ARec, we conduct extensive experiments and report detailed analysis results.

### 5.1 Experimental Settings

**5.1.1 Datasets.** We evaluate the performance of the proposed model through experiments conducted on three public datasets:

- **MovieLens-1M** (referred to as **ML-1M**) [14]: A dataset collected from the MovieLens platform, containing approximately 1 million user ratings of movies.
- **Amazon Prime Pantry** (referred to as **Amazon**) [28]: A dataset of user reviews in the grocery category collected from the Amazon platform up to 2018.
- **Zhihu-1M** [13]: A dataset sourced from a large knowledge-sharing platform (Zhihu), consisting of raw data, including information on questions, answers, and user profiles.

For each user, we sort their interaction records by timestamp to generate an interaction sequence. We retain only users and items associated with at least ten interaction records. We follow the leave-one-out policy [19] for training-validation-testing partition. The statistical details of these datasets are presented in Table 1.

**5.1.2 Baselines.** To demonstrate the effectiveness of our proposed method, we conduct comparisons with several representative sequential recommendation baseline models: Foundational models such as **Caser** [44], a Convolutional Neural Network (CNN)-based approach; **GRU4Rec** [17], which leverages Gated Recurrent Units (GRU) from Recurrent Neural Networks (RNN); **BERT4Rec** [41], adopting the bidirectional attention mechanism of BERT [5]; **SASRec** [19], the first to introduce the Transformer architecture to this field; and innovative models like **Mamba4Rec** [24], which applies the Mamba architecture. Time-aware models such as **Ti-SASRec** [21], which enhances SASRec with temporal information, and **TiM4Rec** [6], which improves low-dimensional performance of SSD while maintaining efficiency, integrate temporal dynamics. **TTT4Rec** [54] uses Test-Time Training to dynamically adapt model parameters during inference for sequential recommendation.

**5.1.3 Evaluation Metrics.** To evaluate the performance of top- $K$  recommendation, we adapt the metrics Recall@ $K$ , MRR@ $K$ , and NDCG@ $K$ , which are widely used in recommendation research to evaluate model performance [2, 56, 58]. In this context, we set  $K = 10$  and present the average scores on the test dataset.

**5.1.4 Implementation Details.** Our evaluation is based on implementations using PyTorch [30] and RecBole [60]. We set all model dimensions  $d$  to 64, the learning rate to 0.001, and the batch size to 4096. Additionally, we set the state dimension  $d_s$  of the T<sup>2</sup>A-Mamba to 32, the number of blocks to 1, the training steps  $M$  during testing

**Table 2: Performance comparison of different sequential recommendation models. The best result is bolded and the runner-up is underlined. \* means improvements over the second-best methods are significant ( $t$ -test,  $p$ -value < 0.05).**

Model	ML-1M			Amazon			Zhihu-1M		
	Recall@10	MRR@10	NDCG@10	Recall@10	MRR@10	NDCG@10	Recall@10	MRR@10	NDCG@10
Caser	0.1954	0.0703	0.0994	0.0594	0.0204	0.0294	0.0288	0.0089	0.0134
GRU4Rec	0.2732	0.1147	0.1518	0.0939	0.0480	0.0586	0.0283	0.0092	0.0142
BERT4Rec	0.2770	0.1093	0.1482	0.0675	0.0372	0.0443	0.0289	0.0098	0.0142
SASRec	0.2471	0.0911	0.1273	0.1025	0.0527	0.0644	0.0364	0.0098	0.0159
Mamba4Rec	0.2813	0.1201	0.1578	0.1003	0.0522	0.0635	0.0355	0.0112	0.0167
TiSASRec	0.2713	0.1191	0.1478	<u>0.1030</u>	0.0540	0.0651	<u>0.0390</u>	0.0117	0.0175
TiM4Rec	0.2873	<u>0.1211</u>	0.1596	0.1016	<u>0.0567</u>	<u>0.0665</u>	0.0384	<u>0.0118</u>	<u>0.0179</u>
TTT4Rec	<u>0.2887</u>	<u>0.1208</u>	0.1599	0.1020	0.0560	0.0655	0.0370	0.0115	0.0174
T <sup>2</sup> ARec (ours)	<b>0.2932*</b>	<b>0.1262*</b>	<b>0.1648*</b>	<b>0.1102*</b>	<b>0.0580*</b>	<b>0.0705*</b>	<b>0.0402*</b>	<b>0.0122*</b>	<b>0.0187*</b>

$M$  to 1 and the learning rate during testing  $\alpha$  to 0.005. Furthermore, the search space for the weights of the two self-supervised losses during training,  $\lambda_1^{\text{train}}$  and  $\lambda_2^{\text{train}}$ , is  $\{0.01, 0.1, 1, 10\}$ , and for testing, the search space for  $\lambda_1^{\text{test}}$  and  $\lambda_2^{\text{test}}$  is  $\{1e-3, 1e-2, 1e-1, 1\}$ . To adapt to the characteristics of the baselines and datasets, we set the fixed sequence length to 200 for ML-1M and 50 for other datasets (Amazon and Zhihu-1M). For a fair comparison, we ensured consistency of key hyperparameters across different models while using default hyperparameters for baseline methods as recommended in their respective papers. Finally, we utilized the Adam optimizer [20] in a mini-batch training manner.

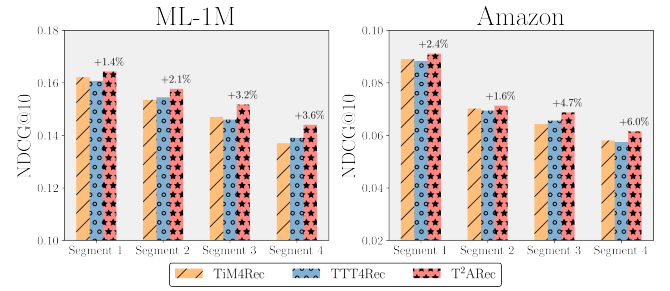
## 5.2 Overall Performance

Table 2 presents the performance comparison of the proposed T<sup>2</sup>ARec and other baseline methods on three datasets. From the table, we observe several key insights:

- (1) General sequential methods, such as SASRec and Mamba4Rec, show varying strengths depending on the dataset characteristics. Mamba4Rec performs better on ML-1M, a dataset with longer sequences, due to its ability to model complex long-term dependencies, whereas SASRec achieves slightly higher performance on shorter sequence datasets like Amazon and Zhihu-1M, where simpler sequence modeling suffices.
- (2) TiSASRec, TiM4Rec and TTT4Rec outperform these general sequential methods by alleviating specific challenges in recommendation. TiSASRec and TiM4Rec effectively incorporates temporal information, enhancing their ability to model time-sensitive dynamics, while TTT4Rec leverages test-time training to alleviate distribution shifts, leading to superior performance.
- (3) Finally, T<sup>2</sup>ARec outperforms all baselines across datasets, achieving the best results by introducing time interval alignment loss and interest state alignment loss. These innovations enable T<sup>2</sup>ARec to capture temporal and sequential patterns more effectively and adapt dynamically to test data through gradient descent during inference, further alleviating distributional shifts and enhancing overall performance.

## 5.3 Ablation Studies

We conduct ablation experiments in T<sup>2</sup>ARec to validate the effectiveness of the time interval alignment loss and the interest state

**Figure 5: Effectiveness of T<sup>2</sup>ARec on user interest shifts.**

alignment loss. The experimental configurations include ' $\mathcal{L}_{\text{both}}$ ', where both losses are removed during training and testing, as well as ' $\mathcal{L}_{\text{time}}$ ' and ' $\mathcal{L}_{\text{state}}$ ', where only one of the losses is removed. During testing, we design configurations such as ' $\mathcal{L}_{\text{both}}^{\text{test}}$ ', ' $\mathcal{L}_{\text{time}}^{\text{test}}$ ', and ' $\mathcal{L}_{\text{state}}^{\text{test}}$ ', which disable all or part of the losses during the testing phase. As shown in Table 3, removing any loss significantly degrades model performance, highlighting the importance of adapting to use interest shifts. The time interval alignment loss captures temporal dependencies, while the interest state alignment loss captures the current user's interest state. Both are crucial for the robustness and adaptability of sequential recommendations.

## 5.4 Further Analysis

**5.4.1 Effectiveness of T<sup>2</sup>ARec on User Interest Shifts.** To validate the effectiveness of T<sup>2</sup>ARec on user interest shifts, we followed the setup in Figure 2 and compared T<sup>2</sup>ARec with TiM4Rec and TTT4Rec on both the ML-1M and Amazon datasets. As shown in Figure 5, the later the timestamp of the segment, indicating a greater user interest shift, the performance of TiM4Rec and TTT4Rec shows a declining trend. On the basis of these two baselines, our model demonstrates higher improvements in Segment 3 and Segment 4 compared to Segment 1 and Segment 2. This indicates that T<sup>2</sup>ARec performs better on test data with more significant user interest shifts, highlighting the effectiveness of our two alignment losses during Test-Time Alignment.

**Table 3: Ablation studies.**

Model	ML-1M			Amazon			Zhihu-1M		
	Recall@10	MRR@10	NDCG@10	Recall@10	MRR@10	NDCG@10	Recall@10	MRR@10	NDCG@10
T <sup>2</sup> ARec	0.2932	0.1262	0.1648	0.1102	0.0580	0.0705	0.0402	0.0122	0.0187
w/o $\mathcal{L}_{\text{both}}$	0.2813	0.1207	0.1578	0.1003	0.0522	0.0635	0.0355	0.0112	0.0167
w/o $\mathcal{L}_{\text{time}}$	0.2891	0.1225	0.1618	0.1026	0.0531	0.0649	0.0389	0.0110	0.0180
w/o $\mathcal{L}_{\text{state}}$	0.2876	0.1214	0.1590	0.1010	0.0545	0.0652	0.0365	0.0114	0.0175
w/o $\mathcal{L}_{\text{both}}^{\text{test}}$	0.2899	0.1239	0.1626	0.1033	0.0568	0.0677	0.0389	0.0113	0.0177
w/o $\mathcal{L}_{\text{time}}^{\text{test}}$	0.2912	0.1252	0.1645	0.1095	0.0570	0.0690	0.0390	0.0120	0.0180
w/o $\mathcal{L}_{\text{state}}^{\text{test}}$	0.2921	0.1241	0.1635	0.1067	0.0558	0.0686	0.0393	0.0120	0.0183

**Table 4: Analysis of test-time throughput, defined as the number of iterations per second, with each iteration processing a batch of 4096 testing examples.**

Model	Test-Time Throughput (# of iterations / second)		
	ML-1M	Amazon	Zhihu-1M
SASRec	1.56	2.26	1.96
Mamba4Rec	2.82	3.11	3.01
TiSASRec	1.50	2.52	1.71
TiM4Rec	2.16	2.64	2.56
TTT4Rec	0.96	1.74	1.19
T <sup>2</sup> ARec (ours)	1.02	2.05	1.36

**5.4.2 Analysis of Test-time Throughput.** Noticed that training during testing requires gradient descent, which could introduce additional load on test-time (inference) throughput. We conducted offline experiments to analyze the impact of training during testing. The results are shown in Table 4, where we compare T<sup>2</sup>ARec with several typical baselines and quantify the test-time throughput in terms of it/s (iterations per second). From the results in the table, it is evident that the test-time throughput of the method that trains during testing is approximately half of that of Mamba4Rec that does not train during testing. Nevertheless, with increasing computational power, the additional load from training during testing may not have a significant impact, especially since previous work has shown that TTT has yielded online benefits [48]. We present these findings and provide optimization opportunities for future work applying TTT in recommendations.

## 6 Conclusion

To address dynamic user interest shifts in real-world sequential recommendation, we propose T<sup>2</sup>ARec, which integrates Test-Time Training via two self-supervised losses. Existing static methods struggle to adapt to such shifts, leading to performance drops. Our approach aligns absolute time intervals with model-adaptive learning intervals to capture temporal dynamics and introduces an interest state alignment mechanism to explicitly identify user interest shifts with theoretical guarantees. These two modules enable efficient, incremental parameter updates during testing. Experiments show T<sup>2</sup>ARec outperforms state-of-the-art methods, demonstrating TTT's potential for enhancing test-time adaptability in tracking evolving user preferences.

## Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (No. 62376275, 62472426). Work partially done at Beijing Key Laboratory of Research on Large Models and Intelligent Governance, and Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE. Supported by fund for building world-class universities (disciplines) of Renmin University of China.

## A Proof of Theorem 4.1

**PROOF OF THEOREM 4.1.** Based on the definitions of  $\mathcal{L}_{\text{state}}$ ,  $\hat{\mathbf{h}}_n^b$ , and  $\hat{\mathbf{h}}_{n+1}$ , we can derive:

$$\begin{aligned}
\mathcal{L}_{\text{state}} &= \left\| \mathbf{h}_n - \hat{\mathbf{h}}_n^b \right\|_2 \Delta_{n+1}^{-2} \\
&= \left\| \mathbf{h}_n - \left( e^{\Delta_{n+1}P} \hat{\mathbf{h}}_{n+1} + \Delta_{n+1} \mathbf{Q} \otimes \mathbf{x}_n \right) \right\|_2 \Delta_{n+1}^{-2} \\
&= \left\| \mathbf{h}_n - \left( e^{\Delta_{n+1}P} \left( e^{\Delta_{n+1}A} \mathbf{h}_n + \Delta_{n+1} \mathbf{B}_{n+1} \otimes \hat{\mathbf{x}}_{n+1} \right) + \Delta_{n+1} \mathbf{Q} \mathbf{x}_n \right) \right\|_2 \Delta_{n+1}^{-2} \\
&= \left\| \left( 1 - e^{\Delta_{n+1}(P+A)} \right) \mathbf{h}_n + \Delta_{n+1} \mathbf{Q} \otimes \mathbf{x}_n - \Delta_{n+1} e^{\Delta_{n+1}P} \mathbf{B}_{n+1} \otimes \hat{\mathbf{x}}_{n+1} \right\|_2 \Delta_{n+1}^{-2} \\
&= \left\| \left( 1 - e^{\Delta_{n+1}(P+A)} \right) \mathbf{h}_n + \Delta_{n+1} \left( \mathbf{Q} - e^{\Delta_{n+1}P} \mathbf{B}_{n+1} \right) \otimes \mathbf{x}_n + \right. \\
&\quad \left. \Delta_{n+1} e^{\Delta_{n+1}P} \mathbf{B}_{n+1} \otimes (\mathbf{x}_n - \hat{\mathbf{x}}_{n+1}) \right\|_2 \Delta_{n+1}^{-2},
\end{aligned}$$

yielding that

$$\begin{aligned}
\mathcal{L}_{\text{state}} &\leq \left\| \left( 1 - e^{\Delta_{n+1}(P+A)} \right) \mathbf{h}_n \right\|_2 \Delta_{n+1}^{-2} + \left\| \Delta_{n+1} \left( \mathbf{Q} - \mathbf{B}_{n+1} e^{\Delta_{n+1}P} \right) \otimes \mathbf{x}_n \right\|_2 \Delta_{n+1}^{-2} + \\
&\quad \left\| \Delta_{n+1} \mathbf{B}_{n+1} e^{\Delta_{n+1}P} \otimes (\mathbf{x}_n - \hat{\mathbf{x}}_{n+1}) \right\|_2 \Delta_{n+1}^{-2} \quad (\text{triangle inequality}) \\
&\leq \Delta_{n+1}^{-2} \|\mathbf{h}_n\|_2 + \left\| \Delta_{n+1}^{-1} \left( -A^{-1} \mathbf{B}_{n+1} + \epsilon_n - e^{\Delta_{n+1}P} \mathbf{B}_{n+1} \right) \otimes \mathbf{x}_n \right\|_2 + \\
&\quad \left\| \Delta_{n+1}^{-1} e^{\Delta_{n+1}P} \mathbf{B}_{n+1} \otimes (\mathbf{x}_n - \hat{\mathbf{x}}_{n+1}) \right\|_2 \quad (\text{Def. of } \mathbf{Q}; e^x \leq 1, \forall x \leq 0) \\
&= \Delta_{n+1}^{-2} \|\mathbf{h}_n\|_2 + \left\| \Delta_{n+1}^{-1} \left( - \left( A^{-1} + e^{\Delta_{n+1}P} \right) \mathbf{B}_{n+1} + \epsilon_n \right) \otimes \mathbf{x}_n \right\|_2 + \\
&\quad \left\| \Delta_{n+1}^{-1} A^{-1} \mathbf{B}_{n+1} \otimes (\mathbf{x}_n - \hat{\mathbf{x}}_{n+1}) \right\|_2 \quad (P = \ln(-A^{-1})/\Delta_{n+1}) \\
&= \Delta_{n+1}^{-2} \|\mathbf{h}_n\|_2 + \left\| \Delta_{n+1}^{-1} \epsilon_n \otimes \mathbf{x}_n \right\|_2 + \left\| \Delta_{n+1}^{-1} A^{-1} \mathbf{B}_{n+1} \otimes (\mathbf{x}_n - \hat{\mathbf{x}}_{n+1}) \right\|_2 \\
&= \Delta_{n+1}^{-2} \|\mathbf{h}_n\|_2 + \Delta_{n+1}^{-1} \|\mathbf{x}_n\|_2 \|\epsilon_n\|_2 + A^{-1} \|\mathbf{B}_{n+1}\|_2 \left\| \frac{\mathbf{x}_n - \hat{\mathbf{x}}_{n+1}}{\Delta_{n+1}} \right\|_2.
\end{aligned}$$

□

## References

- [1] Jianxin Chang, Chen Gao, Yu Zheng, Yiqun Hui, Yanan Niu, Yang Song, Depeng Jin, and Yong Li. 2021. Sequential recommendation with graph neural networks. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*. 378–387.
- [2] Sirui Chen, Yuan Wang, Zijing Wen, Zhiyu Li, Changshuo Zhang, Xiao Zhang, Quan Lin, Cheng Zhu, and Jun Xu. 2023. Controllable multi-objective re-ranking with policy hypernetworks. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*. 3855–3864.
- [3] Sunhao Dai, Changle Qu, Sirui Chen, Xiao Zhang, and Jun Xu. 2024. Recode: Modeling repeat consumption with neural ode. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2599–2603.
- [4] Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060* (2024).
- [5] Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [6] Hao Fan, Mengyi Zhu, Yanrong Hu, Hailin Feng, Zhijie He, Hongjiu Liu, and Qingyang Liu. 2024. TiM4Rec: An Efficient Sequential Recommendation Model Based on Time-Aware Structured State Space Duality Model. *arXiv preprint arXiv:2409.16182* (2024).
- [7] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. 2020. Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations. *ACM Transactions on Information Systems (TOIS)* 39, 1 (2020), 1–42.
- [8] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. 2022. It's raw! audio generation with state-space models. In *International Conference on Machine Learning*. PMLR, 7616–7633.
- [9] Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752* (2023).
- [10] Albert Gu, Karan Goel, and Christopher Ré. 2021. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396* (2021).
- [11] 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.
- [12] Gustavo A Vargas Hakim, David Osowiechi, Mehrdad Noori, Milad Cheraghali, Ali Bahri, Ismail Ben Ayed, and Christian Desrosiers. 2023. Clust3: Information invariant test-time training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 6136–6145.
- [13] Bin Hao, Min Zhang, Weizhi Ma, Shaoyun Shi, Xinxing Yu, Houzhi Shan, Yiqun Liu, and Shaoping Ma. 2021. A large-scale rich context query and recommendation dataset in online knowledge-sharing. *arXiv preprint arXiv:2106.06467* (2021).
- [14] 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.
- [15] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. 2022. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16000–16009.
- [16] 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.
- [17] B Hidasi. 2015. Session-based Recommendations with Recurrent Neural Networks. *arXiv preprint arXiv:1511.06939* (2015).
- [18] Wei Jin, Tong Zhao, Jiayuan Ding, Yozon Liu, Jiliang Tang, and Neil Shah. 2022. Empowering graph representation learning with test-time graph transformation. *arXiv preprint arXiv:2210.03561* (2022).
- [19] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.
- [20] Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [21] Jiacheng Li, Yujie Wang, and Julian McAuley. 2020. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th international conference on web search and data mining*. 322–330.
- [22] Jian Liang, Ran He, and Tieniu Tan. 2024. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision* (2024), 1–34.
- [23] Zachary Chase Lipton. 2015. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv Preprint, CoRR, abs/1506.00019* (2015).
- [24] Chengkai Liu, Jianghao Lin, Jinling Wang, Hanzhou Liu, and James Caverlee. 2024. Mamba4rec: Towards efficient sequential recommendation with selective state space models. *arXiv preprint arXiv:2403.03900* (2024).
- [25] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2021. Self-supervised learning: Generative or contrastive. *IEEE transactions on knowledge and data engineering* 35, 1 (2021), 857–876.
- [26] Yixin Liu, Ming Jin, Shirui Pan, Chuan Zhou, Yu Zheng, Feng Xia, and S Yu Philip. 2022. Graph self-supervised learning: A survey. *IEEE transactions on knowledge and data engineering* 35, 6 (2022), 5879–5900.
- [27] Yuejiang Liu, Parth Kothari, Bastien Van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. 2021. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems* 34 (2021), 21808–21820.
- [28] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 188–197.
- [29] Toshihiro Ota. 2024. Decision mamba: Reinforcement learning via sequence modeling with selective state spaces. *arXiv preprint arXiv:2403.19925* (2024).
- [30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).
- [31] Weicong Qin, Zelin Cao, Weijie Yu, Zihua Si, Sirui Chen, and Jun Xu. 2024. Explicitly Integrating Judgment Prediction with Legal Document Retrieval: A Law-Guided Generative Approach. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2210–2220.
- [32] Weicong Qin, Yi Xu, Weijie Yu, Chenglei Shen, Xiao Zhang, Ming He, Jianping Fan, and Jun Xu. 2024. Enhancing Sequential Recommendations through Multi-Perspective Reflections and Iteration. *arXiv preprint arXiv:2409.06377* (2024).
- [33] Massimo Quadrana, Alexandros Karatzoglou, Balázs Hidasi, and Paolo Cremonesi. 2017. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *proceedings of the Eleventh ACM Conference on Recommender Systems*. 130–137.
- [34] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. 2018. Deep state space models for time series forecasting. *Advances in neural information processing systems* 31 (2018).
- [35] Madeline C Schiappa, Yogesh S Rawat, and Mubarak Shah. 2023. Self-supervised learning for videos: A survey. *Comput. Surveys* 55, 13s (2023), 1–37.
- [36] Chenglei Shen, Xiao Zhang, Teng Shi, Changshuo Zhang, Guofu Xie, and Jun Xu. 2024. A survey of controllable learning: Methods and applications in information retrieval. *arXiv preprint arXiv:2407.06083* (2024).
- [37] Chenglei Shen, Xiao Zhang, Wei Wei, and Jun Xu. 2023. Hyperbandit: Contextual bandit with hypernetwork for time-varying user preferences in streaming recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*. 2239–2248.
- [38] Chenglei Shen, Jiahao Zhao, Xiao Zhang, Weijie Yu, Ming He, and Jianping Fan. 2024. Generating Model Parameters for Controlling: Parameter Diffusion for Controllable Multi-Task Recommendation. *arXiv preprint arXiv:2410.10639* (2024).
- [39] Teng Shi, Zihua Si, Jun Xu, Xiao Zhang, Xiaoxue Zang, Kai Zheng, Dewei Leng, Yanan Niu, and Yang Song. 2024. UniSAR: Modeling User Transition Behaviors between Search and Recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1029–1039.
- [40] Teng Shi, Jun Xu, Xiao Zhang, Xiaoxue Zang, Kai Zheng, Yang Song, and Enyun Yu. 2025. Unified Generative Search and Recommendation. *arXiv preprint arXiv:2504.05730* (2025).
- [41] 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.
- [42] Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. 2020. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*. PMLR, 9229–9248.
- [43] Jiakai Tang, Sunhao Dai, Teng Shi, Jun Xu, Xu Chen, Wen Chen, Wu Jian, and Yuning Jiang. 2025. Think before recommend: Unleashing the latent reasoning power for sequential recommendation. *arXiv preprint arXiv:2503.22675* (2025).
- [44] Jiayi 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.
- [45] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
- [46] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. 2020. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726* (2020).
- [47] Yiqi Wang, Chaozhou Li, Wei Jin, Rui Li, Jianan Zhao, Jiliang Tang, and Xing Xie. 2022. Test-time training for graph neural networks. *arXiv preprint arXiv:2210.08813* (2022).
- [48] Yuan Wang, Zhiyu Li, Changshuo Zhang, Sirui Chen, Xiao Zhang, Jun Xu, and Quan Lin. 2024. Do Not Wait: Learning Re-Ranking Model Without User Feedback

- At Serving Time in E-Commerce. In *Proceedings of the 18th ACM Conference on Recommender Systems*. 896–901.
- [49] Xu Xie, Fei Sun, Zhaoyang Liu, Shiwen Wu, Jinyang Gao, Jiandong Zhang, Bolin Ding, and Bin Cui. 2022. Contrastive learning for sequential recommendation. In *2022 IEEE 38th international conference on data engineering (ICDE)*. IEEE, 1259–1273.
- [50] Rui Xu, Shu Yang, Yihui Wang, Bo Du, and Hao Chen. 2024. A survey on vision mamba: Models, applications and challenges. *arXiv preprint arXiv:2404.18861* (2024).
- [51] Yi Xu, Weicong Qin, Weijie Yu, Ming He, Jianping Fan, and Jun Xu. 2025. Decoding Recommendation Behaviors of In-Context Learning LLMs Through Gradient Descent. *arXiv preprint arXiv:2504.04386* (2025).
- [52] Xihong Yang, Yiqi Wang, Jin Chen, Wenqi Fan, Xiangyu Zhao, En Zhu, Xinwang Liu, and Defu Lian. 2024. Dual test-time training for out-of-distribution recommender system. *arXiv preprint arXiv:2407.15620* (2024).
- [53] Xihong Yang, Yiqi Wang, Yue Liu, Yi Wen, Lingyuan Meng, Sihang Zhou, Xinwang Liu, and En Zhu. 2024. Mixed graph contrastive network for semi-supervised node classification. *ACM Transactions on Knowledge Discovery from Data* (2024).
- [54] Zhaoqi Yang, Yanan Wang, and Yong Ge. 2024. TTT4Rec: A Test-Time Training Approach for Rapid Adaption in Sequential Recommendation. *arXiv preprint arXiv:2409.19142* (2024).
- [55] Changshuo Zhang, Sirui Chen, Xiao Zhang, Sunhao Dai, Weijie Yu, and Jun Xu. 2024. Reinforcing Long-Term Performance in Recommender Systems with User-Oriented Exploration Policy. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1850–1860.
- [56] Changshuo Zhang, Zihan Lin, Shukai Liu, Yongqi Liu, and Han Li. 2025. Comment Staytime Prediction with LLM-enhanced Comment Understanding. In *Companion Proceedings of the ACM on Web Conference 2025*. 586–595.
- [57] Changshuo Zhang, Teng Shi, Xiao Zhang, Qi Liu, Ruobing Xie, Jun Xu, and Ji-Rong Wen. 2024. Modeling Domain and Feedback Transitions for Cross-Domain Sequential Recommendation. *arXiv preprint arXiv:2408.08209* (2024).
- [58] Changshuo Zhang, Teng Shi, Xiao Zhang, Yanping Zheng, Ruobing Xie, Qi Liu, Jun Xu, and Ji-Rong Wen. 2024. QAGCF: Graph Collaborative Filtering for Q&A Recommendation. *arXiv preprint arXiv:2406.04828* (2024).
- [59] Kepu Zhang, Teng Shi, Sunhao Dai, Xiao Zhang, Yinfeng Li, Jing Lu, Xiaoxue Zang, Yang Song, and Jun Xu. 2024. SAQRec: Aligning Recommender Systems to User Satisfaction via Questionnaire Feedback. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 3165–3175.
- [60] Wayne Xin Zhao, Shanlei Mu, Yupeng Hou, Zihan Lin, Yushuo Chen, Xingyu Pan, Kaiyuan Li, Yujie Lu, Hui Wang, Changxin Tian, et al. 2021. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. In *proceedings of the 30th acm international conference on information & knowledge management*. 4653–4664.