

LazFormer: Scaling Transformers for Industrial Recommendation via Transferable Generative Pre-training

Xiaodong Li, Alin Fan, Mingyang Li, Yan Xiao, Shichao Nie, Junfeng Zhang, Shaochuan Lin,

Zhanming Ou, Tao Luo, Xiaoyi Zeng

Alibaba International Digital Commerce Group, Beijing, China

{limingxi.lxd, alin.fal, lmy398032, yanwei.xy, nsc383441, sichu.zjf, lin.lsc, zhanming.ozm, luotao.lt, yuanhan}@alibaba-inc.com

Abstract

Transformers have shown promising performance in LLMs due to their outstanding scalability, while several studies have investigated the scalability of Transformers for industrial recommendation. They typically rely on a single ranking model to optimize both sparse and dense parameters from scratch, resulting in substantial computational resource consumption and slow convergence. Fortunately, the pre-training models offer an effective solution to the above issues by providing favorable initialization of both sparse and dense parameters for the subsequent ranking. However, the pre-training and ranking paradigm still faces two major limitations: (1) Since the input features used in pre-training and ranking are usually inconsistent, directly transferring dense parameters from pre-training to ranking may lead to negative transfer. (2) Multi-epoch training during the ranking process may result in the overfitting of sparse parameters, while freezing the sparse parameters limits their adaptability to the ranking objectives. To this end, we propose a *Scaling Transformer for Industrial Recommendation via Transferable Generative Pre-training*, termed **LazFormer**. Specifically, we first present a generative pre-training module to autoregressively generate sequential features, providing favorable initialization of both sparse and dense parameters for the subsequent ranking. To solve the negative transfer of dense parameters, we propose a transferable residual adapter that injects additional ranking-specific features into ranking in a residual manner. Moreover, a request-aware ranking module integrates long-sequence compression, hybrid sparse attention, and a request-aware paradigm to efficiently model users' long sequences. Besides, we further propose an asymmetric multi-epoch training strategy that resets sparse parameters while continuously accumulating dense parameters across epochs, alleviating the overfitting of sparse parameters. Extensive offline experiments and online A/B testing demonstrate the effectiveness and scalability of LazFormer. LazFormer has been deployed in an industrial recommendation system, delivering significant gains for the platform.

CCS Concepts

• Information systems → Recommender systems.

Keywords

Recommender Systems, Generative Pre-training, Ranking, Model Scaling, Data Scaling

1 Introduction

The Transformer architecture has demonstrated remarkable success in large language models (LLMs) [1, 21, 30], where model

quality continuously improves by scaling model capacity, training data, and computational resources. The unified token-based architecture and outstanding scalability of Transformers have also inspired a paradigm shift in industrial recommendation. Traditional industrial recommendation models [12, 28, 34] usually follow an explicit feature interaction paradigm, in which high-dimensional sparse embeddings are processed by dedicated sequence modeling and feature interaction modules. More recently, Wukong [32] and RankMixer [17, 36] explore scaling feature interaction networks by increasing model parameters and computational budgets. Nevertheless, industrial recommendation introduces unique challenges absent in LLMs, including billions of sparse features, rapidly evolving user interests, and strict latency constraints. Thus, how to fully unleash the potential of Transformers in industrial recommendation remains largely underexplored.

Recently, several studies [13, 31] have investigated the scalability of Transformer-based recommenders. Specifically, HSTU [31] and MTGR [13] reformulate recommendation as a generative sequential transduction problem and demonstrate an empirical scaling law by increasing model capacity, sequence length, and training compute. More recently, OneTrans [33] employs a unified Transformer backbone to jointly conduct user sequence modeling and feature interaction in a per-token manner, and subsequent studies [15, 16] further extend this direction with heterogeneous feature interaction and long-sequence modeling. These methods demonstrate that Transformer-based recommenders can continuously benefit from scaling depth, width, and sequence length.

Despite their encouraging progress, existing Transformer-based methods typically rely on a single ranking model to optimize both sparse and dense parameters from scratch, leading to substantial computational resource consumption and slow convergence. Fortunately, pre-training models could provide favorable initialization of both sparse and dense parameters for the subsequent ranking process, offering an effective solution to the above issues. However, the pre-training and ranking paradigm still faces two major limitations: (1) **Negative transfer of dense parameters**. The features available during pre-training are usually inconsistent with those required by the ranking process. Directly incorporating these additional ranking-specific features into ranking may disturb the representation space expected by the pre-trained dense parameters, thereby leading to negative transfer. (2) **Overfitting of sparse parameters**. Continuously optimizing billions of sparse parameters for multiple epochs may lead to overfitting [9, 22] and destroy their transferable representations. Conversely, freezing the sparse parameters limits their adaptability to the ranking objectives. Thus, effectively transferring both sparse and dense parameters while

co-scaling model capacity and effective training data consumption remains a critical challenge.

To address the above limitations, we propose **LazFormer**, a *Scaling Transformer for Industrial Recommendation via Transferable Generative Pre-training*. Specifically, LazFormer first performs generative pre-training over users' historical interaction sequences via autoregressive next-item prediction, jointly learning sparse and dense parameters to initialize subsequent ranking. To mitigate the negative transfer of dense parameters, we introduce a transferable residual adapter that injects additional ranking-specific features into ranking in a residual manner. Moreover, we propose a request-aware ranking module for long-sequence modeling, integrating long-sequence compression, hybrid sparse attention, and a request-aware paradigm. Besides, we introduce an asymmetric multi-epoch training strategy that resets sparse parameters while continuously accumulating dense parameters across epochs, enabling the Transformer backbone to absorb more ranking supervision while avoiding sparse overfitting. Ultimately, these components enable the effective transfer of both sparse and dense parameters, as well as the co-scaling of model capacity and training data in LazFormer.

Our main contributions can be summarized as follows:

- We propose LazFormer, a scaling transformer for industrial recommendation via transferable generative pre-training, which jointly transfers sparse and dense parameters from generative pre-training to downstream ranking while alleviating dense negative transfer and sparse overfitting.
- We introduce a transferable residual adapter for stable sparse and dense transfer, a request-aware ranking module for efficient model scaling, and an asymmetric multi-epoch training strategy for effective training data scaling.
- We conduct extensive offline experiments and online A/B testing to validate the effectiveness of LazFormer. LazFormer has been deployed in a real-world industrial recommendation system, delivering significant gains for the platform.

2 Problem Definition

In industrial recommendation, we focus on predicting multiple probabilities that reflect a user's preference for a target (candidate) item, including: the click-through rate (*i.e.*, **CTR**) and the post-click conversion rate (*i.e.*, **CVR**).

Formally, we use $\mathcal{D}$ to denote the user-item interaction data, which can be formulated as $\mathcal{D} = \{\mathcal{U}, \mathcal{V}, \mathcal{Y}\}$. Specifically, $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{Y}$ represent the user set, item set and several true labels (*i.e.*, $y^{ctr} \in \{0, 1\}$, and $y^{cvr} \in \{0, 1\}$), respectively. For instance, $y_{ij}^{ctr} \in \{0, 1\}$ denotes whether user u_i clicks item v_j . Moreover, the historical interaction sequence of user u_i can be denoted as $\mathcal{H}_i = \{v_j | v_j \in \mathcal{V}\}_{j=1}^{|\mathcal{H}_i|}$, where each v_j is associated with multiple distinct categories of features, including ID-based features and Side Info-based features:

- **ID-based** features refer to the item ID of user's (*i.e.*, u_i) historical interaction sequence, *i.e.*, e_j^{id} .
- **Side Info-based** features contain item's attribute information and the contextual features. The former includes category, shop, brand, decay, stay time, and so on (*i.e.*, e_j^{cate} , e_j^{shop} , e_j^{brand} , e_j^{decay} , e_j^{stay} , etc.), where e_j^{decay} represents the time interval between the

user's interacted item and the current request. The latter mainly describes the interaction scene (*i.e.*, e_j^{scene}).

Thus, given samples consisting of users' ID-based features, Side Info-based features, and several true labels, our model aims to predict the probabilities of **CTR** and **CVR** for a user u_i with respect to the target (candidate) items $\mathcal{T}_i = \{v_j | v_j \in \mathcal{V}\}_{j=1}^{|\mathcal{T}_i|}$ as follows:

$$\hat{y}^{ctr}, \hat{y}^{cvr} = f(u_i, \mathcal{H}_i, \mathcal{T}_i; \Theta), \quad (1)$$

where $\hat{y}^{ctr}, \hat{y}^{cvr}$ are predicted probabilities, Θ are trainable parameters, f is a Transformer-based [15, 16, 33] model.

Ultimately, the Transformer-based model is optimized by minimizing the standard negative log-likelihood function as follows:

$$\mathcal{L}_{XTRs} = - \sum_{xtr}^{ctr, cvr} [y^{xtr} \log(\hat{y}^{xtr}) + (1 - y^{xtr}) \log(1 - \hat{y}^{xtr})], \quad (2)$$

where y^{xtr} denotes user's true labels, and $\hat{y}^{xtr}$ represents the predicted probabilities of XTRs.

3 Methodology

This section introduces our proposed LazFormer, which implements a scaling Transformer framework via transferable generative pre-training, as described in Figure 1. Specifically, LazFormer mainly consists of three parts: 1) the **generative pre-training** aims to generate sequence features autoregressively, thus providing favorable initialization of sparse and dense parameters for the subsequent ranking process, 2) the **transferable residual adapter** draws inspiration from LoRA [14] to inject additional ranking-specific features into the ranking process in a residual manner, thereby solving the negative transfer of dense parameters, and 3) the **request-aware ranking** applies a combination of long-sequence compression in a coarse-to-fine manner, hybrid sparse attention, progressive token pruning, and a request-aware paradigm to achieve model scaling alongside pre-training. Besides, we further propose an **asymmetric multi-epoch training strategy**, which continuously accumulates dense parameters across epochs for data scaling, and resets sparse parameters to alleviate overfitting. Details of each component are demonstrated in the following parts.

3.1 Generative Pre-training

We utilize a Transformer architecture to autoregressively generate sequence features during the generative pre-training, providing favorable initialization of sparse and dense parameters for subsequent ranking. Unlike previous methods [26, 27] that freeze pre-trained parameters, our model enables further updates and optimizations of the pre-trained parameters during ranking.

3.1.1 Tokenization. Given the historical interaction sequence $\mathcal{H}_i = \{v_j\}_{j=1}^{|\mathcal{H}_i|}$ of user u_i , we first transform each interacted item v_j and its associated ID-based and Side Info-based features into a unified token representation. Specifically, for each item v_j , we retrieve the corresponding feature embeddings by performing embedding lookup, which can be formulated as follows:

$$\mathbf{x}_j = [e_j^{id} \| e_j^{cate} \| e_j^{shop} \| e_j^{brand} \| e_j^{decay} \| e_j^{stay}], \quad (3)$$

where $\|$ denotes the concatenation operation.

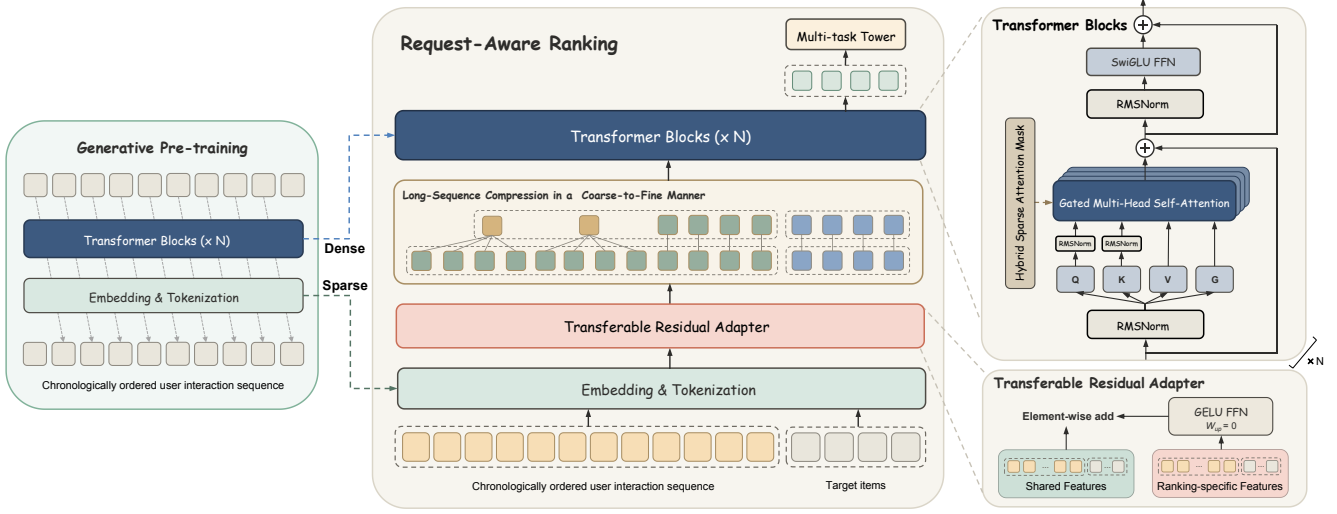


Figure 1: The overall framework of LazFormer.

Since different feature groups have distinct representation spaces, we employ linear projection and normalization to map them into the same dimension. The final representation of the j -th interaction token is formulated as follows:

$$\mathbf{h}_j = \text{RMSNorm}(\mathbf{W}_{attr}\mathbf{x}_j + \mathbf{W}_{cont}\mathbf{e}_j^{scene}), \quad (4)$$

where $\mathbf{W}_{attr}$ and $\mathbf{W}_{cont}$ are trainable parameters. Accordingly, the tokenized interaction sequence of user u_i is denoted as:

$$\mathbf{H}_i = [\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_{|\mathcal{H}_i|}]. \quad (5)$$

Moreover, the sparse embedding and projection layers are shared with the subsequent ranking whenever the corresponding features are available.

3.1.2 Next-Item Prediction. After obtaining the tokenized interaction sequence $\mathbf{H}_i$, we employ a stack of Transformer blocks to capture the sequential dependency among historical interactions. Specifically, each Transformer block comprises causal self-attention, a feed-forward network (FFN), normalization, and residual connections. Within the attention layer, we introduce pre-Norm, QKNorm, and attention gate [30] to stabilize the training process. Given the input representation $\mathbf{H}_i^{l-1}$ of the l -th layer, we first perform RMS normalization and project the normalized representations into Q/K/V matrices as follows:

$$\mathbf{Q}_r^l, \mathbf{K}_r^l, \mathbf{V}_r^l = \text{RMSNorm}(\mathbf{H}_i^{l-1}\mathbf{W}_r^Q, \mathbf{H}_i^{l-1}\mathbf{W}_r^K, \mathbf{H}_i^{l-1}\mathbf{W}_r^V), \quad (6)$$

where r denotes the attention head, $\mathbf{W}_r^Q$, $\mathbf{W}_r^K$ and $\mathbf{W}_r^V$ are trainable parameters. To prevent information leakage from future interactions, we adopt a causal attention mask $\mathbf{M}$. Thus, the output of each attention head is calculated as follows:

$$\text{Head}_r^l = \text{Softmax}\left(\frac{\mathbf{Q}_r^l \mathbf{K}_r^{l\top}}{\sqrt{d_k}} + \mathbf{M}\right) \mathbf{V}_r^l, \quad (7)$$

where d_k is the dimension of each attention head. The outputs of all n heads are concatenated and projected to obtain the multi-head

attention output as follows:

$$\text{MHA}^l = [\text{Head}_1^l \parallel \text{Head}_2^l \parallel \dots \parallel \text{Head}_n^l] \mathbf{W}^O, \quad (8)$$

where $\mathbf{W}^O$ denotes the trainable parameter.

We then apply residual connections to the attention output, followed by another normalization and a gated feed-forward network, which can be formulated as follows:

$$\begin{aligned} \tilde{\mathbf{H}}_i^l &= \mathbf{H}_i^{l-1} + \text{MHA}^l, \\ \mathbf{H}_i^l &= \tilde{\mathbf{H}}_i^l + \text{FFN}^l(\text{RMSNorm}(\tilde{\mathbf{H}}_i^l)), \end{aligned} \quad (9)$$

Ultimately, we obtain the final sequential representations $\mathbf{Z}_i = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_{|\mathcal{H}_i|}]$.

To perform autoregressive generation, the hidden representation $\mathbf{z}_j$ is projected into the item ID embedding space to obtain $\mathbf{q}_j$, thus predicting the next interacted item v_{j+1} as follows:

$$\mathbf{q}_j = \mathbf{W}_{auto}\mathbf{z}_j, \quad (10)$$

where $\mathbf{W}_{auto}$ is a trainable parameter. We regard the item ID embedding $\mathbf{e}_{j+1}^{id}$ as the positive target and sample a set of negative items $\mathcal{N}_{i,j}$ from other interaction sequences within the same mini-batch. The probability of generating the next item is formulated as:

$$p(v_{j+1} \mid \mathcal{H}_{i,\leq j}) = \frac{\exp(s(\mathbf{q}_j, \mathbf{e}_{j+1}^{id}))}{\sum_{v \in \{v_{j+1}\} \cup \mathcal{N}_{i,j}} \exp(s(\mathbf{q}_j, \mathbf{e}_v^{id}))}, \quad (11)$$

where $s(\mathbf{a}, \mathbf{b}) = \text{sim}(\mathbf{a}, \mathbf{b})/\tau$ denotes the temperature-scaled cosine similarity, and τ is a learnable temperature parameter. Accordingly, the loss function of generative pre-training is defined as follows:

$$\mathcal{L}_{pre} = - \sum_{u_i \in \mathcal{U}} \sum_{j=1}^{|\mathcal{H}_i|-1} \log p(v_{j+1} \mid \mathcal{H}_{i,\leq j}). \quad (12)$$

Through generative pre-training, LazFormer jointly learns transferable sparse item representations and dense Transformer parameters, which are subsequently used to initialize the ranking model.

3.2 Transferable Residual Adapter

The ranking process usually includes additional ranking-specific features that are unavailable during pre-training. Directly concatenating these features with the pre-trained representations may disturb the knowledge learned from sequential generation, leading to the negative transfer of dense parameters. To address this issue, we introduce a transferable residual adapter that injects additional ranking-specific features into the ranking process in a residual manner, thus transferring knowledge across pre-training and ranking.

Specifically, the additional ranking-specific features of item v_j during the ranking process include add to cart, order, and gap (i.e., $e_j^{atc}, e_j^{order}, e_j^{atc-gap}, e_j^{order-gap}$), where gap denotes the time interval between adding to cart or ordering and the most recent click. These features provide an efficient way to incorporate multi-behavior signals beyond clicks without explicitly extending the interaction sequence, thereby avoiding the substantial computational overhead brought by longer self-attention inputs. The transferable residual adapter can be formulated as:

$$\begin{aligned} s_j &= [e_j^{atc} \parallel e_j^{order} \parallel e_j^{atc-gap} \parallel e_j^{order-gap}], \\ a_j &= W_{up} \text{GELU}(W_{down} s_j), \end{aligned} \quad (13)$$

where a_j denotes the residual representation, W_{down} and W_{up} are trainable parameters. Subsequently, a_j is added to the interaction token loaded from the generative pre-training as follows:

$$h_j^{rank} = \text{RMSNorm}(W_{attr} x_j + W_{cont} e_j^{scene} + a_j), \quad (14)$$

where h_j^{rank} represents the interaction token for the subsequent ranking process.

To ensure training stability during the initial phase of the ranking process, we initialize $W_{up} = 0$. Therefore, the adapter initially produces a zero residual as follows:

$$a_j = 0 \rightarrow h_j^{rank} = \text{RMSNorm}(W_{attr} x_j + W_{cont} e_j^{scene}), \quad (15)$$

This zero initialization makes the ranking process start from the features learned during generative pre-training, while gradually incorporating ranking-specific features during optimization. Unlike methods [26, 27] that freeze the pre-trained parameters, LazFormer utilizes the transferable residual adapter to enable the transfer of sparse and dense parameters, allowing the pre-trained knowledge to adapt to the ranking objective.

3.3 Request-Aware Ranking

After generative pre-training, we utilize a transferable residual adapter to transfer the learned sparse and dense parameters to initialize the ranking process. However, the ranking process typically involves substantially longer interaction sequences and multiple target items within the same request. If we continue to train on individual target items, the same long sequence will be encoded repeatedly. As the sequence length increases, storage and bandwidth become bottlenecks. Thus, to efficiently scale Transformers to such an industrial ranking process, we introduce request-aware ranking, which consists of long-sequence compression in a coarse-to-fine manner, hybrid sparse attention, and a request-aware paradigm.

3.3.1 Long-Sequence Compression in a Coarse-to-Fine Manner. Users' interaction sequence usually contains both fine-grained

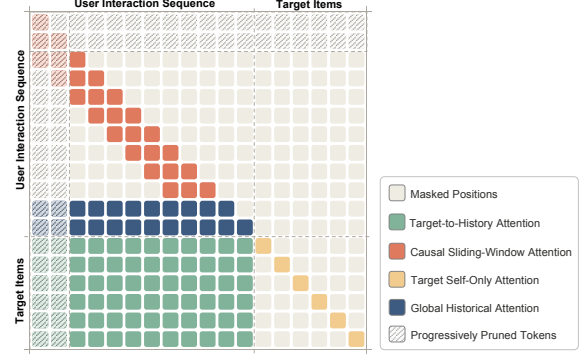


Figure 2: An illustration of hybrid sparse attention mask.

short-term preference and relatively coarse-grained long-term preference. Directly encoding all historical interactions results in excessive computational and memory costs. Consequently, we preserve the short-term interactions while compressing the remaining long-term interactions through group-wise sum pooling.

Let $L_i = |\mathcal{H}_i|$ denote the original historical interaction sequence length and $\ell_i = \min(L_i, L_s)$ denote the number of short-term tokens retained, where L_s is a predefined short-term sequence length. We divide the remaining interactions into groups of size g , and calculate the representation of the k -th group as follows:

$$c_k = \sum_{j=\ell_i+(k-1)g+1}^{\min(\ell_i+kg, L_i)} h_j^{rank}, \quad (16)$$

The compressed historical sequence can be formulated as:

$$\hat{H}_i^{rank} = [c_1, \dots, c_{N_i}, h_1^{rank}, \dots, h_{\ell_i}^{rank}], \quad (17)$$

where $N_i = \left\lceil \frac{L_i - \ell_i}{g} \right\rceil$ denotes the number of compressed long-term groups. Consequently, the historical interaction sequence length of u_i is reduced from L_i to $N_i + \ell_i$, while the fine-grained short-term preference are retained.

3.3.2 Hybrid Sparse Attention. Based on the compressed historical sequence, we append all target item tokens belonging to the same request to construct the ranking input as follows:

$$\begin{aligned} X_i^0 &= [\hat{H}_i^{rank} \parallel T_i], \\ &= [c_1, \dots, c_{N_i}, h_1^{rank}, \dots, h_{\ell_i}^{rank}, t_1, \dots, t_{|T_i|}], \end{aligned} \quad (18)$$

where $\hat{H}_i^{rank}$ denotes the compressed historical sequence, and T_i represents the target item tokens, which are constructed using the shared feature embeddings and projection layers.

To facilitate long-range information propagation, we designate the last G_i history tokens, i.e., the G_i most recent behaviors, as global tokens, and treat the remaining tokens as local historical tokens. Thus, the global historical tokens retain user's short-term fine-grained preference.

Since the full self-attention has quadratic cost, we construct a hybrid sparse attention (as shown in Figure 2) pattern by combining causal sliding-window attention and global historical tokens. Specifically, let p and q denote the query and key positions, the

hybrid sparse attention relation can be defined as follows:

$$\mathcal{A}_i(p, q) = \begin{cases} 1, & p, q \leq \tilde{L}_i, q \leq p, (p - q < W \text{ or } p > \tilde{L}_i - G_i), \\ 1, & p > \tilde{L}_i, q \leq \tilde{L}_i, \\ 1, & p > \tilde{L}_i, q > \tilde{L}_i, p = q, \\ 0, & \text{otherwise,} \end{cases} \quad (19)$$

where W denotes the sliding-window size, $\tilde{L}_i = N_i + \ell_i$ denotes the number of history tokens after compression. Specifically, each local token attends to its preceding local window, while global tokens enable long-range propagation. The global tokens provide persistent information anchors for information propagation across distant historical interactions. Each target item attends to all valid historical tokens, including both global and local ones. In contrast, a target item can attend only to itself within the target sequence, preventing information leakage across candidate predictions.

Consequently, the corresponding attention mask $\mathbf{M}_{p,q}$ is constructed as follows:

$$\mathbf{M}_{p,q} = \begin{cases} 0, & \mathcal{A}_i(p, q) = 1, \\ -\infty, & \mathcal{A}_i(p, q) = 0, \end{cases} \quad (20)$$

The hybrid sparse attention operation is then formulated as:

$$\text{SparseAttn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right)\mathbf{V}, \quad (21)$$

Compared with full self-attention, the hybrid sparse attention pattern avoids unnecessary historical interaction pairs while retaining local sequential dependencies, global information propagation, and complete target to history interactions. We materialise this sparsity via flex attention [7], which compiles the block-sparse pattern into fused CUDA kernels.

Moreover, based on the compressed historical sequence, we progressively prune historical tokens across Transformer layers to reduce computation in deeper layers. Let N_i^l denote the number of historical tokens retained at the l -th layer. We linearly decrease the historical sequence length from $\tilde{L}_i$ to a predefined minimum length $N_{\min}$ as follows:

$$N_i^l = \min(\tilde{L}_i, \max(N_{\min}, \text{Round}[\tilde{L}_i - \frac{l-1}{L-1}(\tilde{L}_i - N_{\min})])), \quad (22)$$

where L denotes the number of Transformer layers. After the l -th layer, only the *most recent* N_i^{l+1} history representations and all target-item representations are propagated to the next layer. Consequently, the combination of long-sequence compression, hybrid sparse attention, and progressive token pruning enables LazFormer to reduce the computation required by long-sequence modeling.

3.3.3 Request-Aware Paradigm. Industrial logs contain multiple target items within the same request. Traditional ranking models construct an independent sample $(u_i, \mathcal{H}_i, v_j, y_{ij})$ for each target item v_j . Consequently, the same historical interaction sequence $\mathcal{H}_i$ is repeatedly serialized, stored, transferred from CPU to GPU, and encoded for all target items within the request. As the length of $\mathcal{H}_i$ increases, redundant data movement and user-side computation become substantial system bottlenecks, leading to a waste of computational resources.

To address this issue, we organize the training samples at the request level, which can be formulated as follows:

$$\mathcal{R}_i = (u_i, \mathcal{H}_i, \mathcal{T}_i, \mathcal{Y}_i), \quad (23)$$

where all target items $\mathcal{T}_i$ and their labels $\mathcal{Y}_i$ are organized within the same request, sharing a single historical sequence $\mathcal{H}_i$. Instead of independently encoding $\mathcal{H}_i$ for each target item, LazFormer serializes and transfers $\mathcal{H}_i$ only once and simultaneously processes all target items.

This request-aware paradigm serves as a system-side complement to hybrid sparse attention: hybrid sparse attention reduces the computational cost of long-sequence modeling, while request-level training removes redundant storage, data transfer, and repeated history encoding.

After passing the input $\mathbf{X}_i^0$ through the ranking Transformer, we extract the contextualized representations of all target items as:

$$\mathbf{Z}_i^t = [z_1^t, z_2^t, \dots, z_{|\mathcal{T}_i|}^t]. \quad (24)$$

Each target representation is further fed into the corresponding prediction network as follows:

$$\hat{y}^{xtr} = f_{xtr}(\mathbf{Z}_i^t), \quad (25)$$

where f_{xtr} denotes the task-specific prediction network. Since target items are isolated by the hybrid sparse attention mask, the request-aware paradigm preserves the independence of individual target predictions while sharing the storage, transfer, and encoding of the same user history.

Ultimately, the ranking process is optimized by the XTRs prediction loss defined in Eq. (2) as follows:

$$\mathcal{L}_{rank} = \mathcal{L}_{XTRs}, \quad (26)$$

During the ranking process, the sparse and dense parameters are jointly optimized with the transferable residual adapter and prediction networks.

3.4 Asymmetric Multi-Epoch Training Strategy

Although generative pre-training provides favorable initialization for both sparse and dense parameters, a single epoch for ranking model may still be insufficient for the dense backbone to fully absorb the supervision contained in large-scale interaction logs. A straightforward solution is to train the ranking model for multiple epochs. However, prior studies [9, 22] have observed a one-epoch overfitting phenomenon, indicating that repeatedly updating sparse embeddings on the same training data may cause overfitting. To address this issue, we propose an asymmetric multi-epoch training strategy: sparse parameters are re-initialized to their pre-trained state at the beginning of each epoch, while dense parameters inherit the optimized state from the previous epoch.

Specifically, we divide the parameters of LazFormer into sparse parameters Θ^s and dense parameters Θ^d . At the beginning of each epoch, we reset Θ^s to its initial state Θ_{init}^s , which is transferred from generative pre-training. In contrast, the dense parameters inherit the optimized state from the previous epoch, which can be formulated as follows:

$$\begin{aligned} \Theta_{e,0}^s &= \Theta_{init}^s, \\ \Theta_{e,0}^d &= \begin{cases} \Theta_{init}^d, & e = 1, \\ \Theta_{e-1, K_{e-1}}^d, & e > 1, \end{cases} \end{aligned} \quad (27)$$

Table 1: Statistics of datasets.

Dataset	# Users	# Tokens	# Impressions
Pre-training	16M	11B	–
Ranking	11M	–	370M

where e denotes the epoch index, and K_{e-1} denotes the number of training steps in epoch $e - 1$. During each epoch, both sparse and dense parameters are jointly optimized by $\mathcal{L}_{rank}$.

Consequently, dense parameters can continuously absorb ranking supervision across epochs, while resetting sparse parameters helps preserve the transferable sparse representations learned during pre-training and alleviate overfitting. In this way, the proposed asymmetric multi-epoch training strategy improves effective data scaling while maintaining transferable sparse knowledge.

4 Experiments

In this section, we conduct extensive offline experiments and online A/B testing on our industrial recommendation system to answer the following research questions:

- **RQ1:** Can LazFormer outperform existing scalable ranking Transformers and pre-training-ranking paradigms?
- **RQ2:** Does the transferable residual adapter effectively bridge generative pre-training and downstream ranking?
- **RQ3:** Does the asymmetric multi-epoch training strategy enable effective data scaling while avoiding sparse overfitting?
- **RQ4:** Can LazFormer continuously benefit from scaling model capacity, sequence length, and training data under practical efficiency constraints?
- **RQ5:** How do the efficiency-oriented designs of request-aware ranking affect the trade-off between recommendation performance and training efficiency?
- **RQ6:** How does LazFormer perform in real-world online A/B testing under industrial deployment settings?

4.1 Experimental Settings

4.1.1 Datasets. For offline evaluation, we use large-scale industrial data from a real-world e-commerce recommendation platform. The data consist of a generative pre-training dataset and a ranking dataset. Following [26], the pre-training dataset is constructed by chronologically sorting each user’s interactions over one year and segmenting them into subsequences. The ranking dataset is built from chronologically ordered user interaction sequences, using logs from the past 25 consecutive days for training and the subsequent day for testing. Table 1 summarizes the dataset statistics.

4.1.2 Baselines. We compare **LazFormer** with representative baselines covering vanilla Transformer (*i.e.*, **Transformer**), scalable ranking Transformers (*i.e.*, **HSTU** [31], **OneTrans** [33]), pre-training and ranking paradigm (*i.e.*, **SORT** [27]), and efficient long-sequence modeling (*i.e.*, **STCA** [11]). For fair comparison, all baselines are initialized with the same pre-trained sparse parameters in our experiments, although the original ranking-only baselines are typically trained from scratch in prior work. Only LazFormer further transfers dense parameters from generative pre-training.

Table 2: Offline performance in terms of AUC and GAUC, where the best results are in bold and the best baseline ones are underlined. Indented rows are derived from LazFormer. All baselines load the same pre-trained sparse parameters, while LazFormer further transfers the dense ones. Results are averaged over three runs.

Models ^a	CTR		CVR		Efficiency ^b	
	AUC	GAUC	AUC	GAUC	Params	FLOPs
Transformer	0.7630	0.6704	0.8835	0.7001	6M	20G
HSTU	0.7636	0.6725	0.8826	0.6942	6M	22G
STCA	0.7636	0.6732	0.8830	0.6996	7M	3G
OneTrans	<u>0.7661</u>	0.6738	0.8840	0.7053	28M	15G
SORT	0.7652	<u>0.6740</u>	<u>0.8850</u>	<u>0.7068</u>	16M	29G
LazFormer	0.7735	0.6868	0.8905	0.7130	9M	25G
Improve (pt) ↑	+0.74	+1.28	+0.55	+0.62	–	–
LazFormer*	0.7704	0.6808	0.8885	0.7094	7M	14G
LazFormer [†]	0.7761	0.6891	0.8933	0.7143	9M	75G

^a LazFormer: our default configuration (5 layers, sequence length 2,048) used in all subsequent experiments. LazFormer*: downscaled to the baseline configuration (3 layers, sequence length 1,024). LazFormer[†]: with asymmetric multi-epoch training. Improve: absolute gain of LazFormer over the underlined best baseline of each metric in percentage points.

^b Params exclude embedding parameters, amounting to about 5B for all models. FLOPs are measured for one training pass (forward and backward) over same request-level sample and are accumulated over epochs. FLOPs count unmasked attention pairs only.

4.1.3 Evaluation Metrics. Following previous works [8, 15, 20, 33], we evaluate the effectiveness of LazFormer and baselines using two widely adopted metrics, including AUC and GAUC.

For efficiency and scalability evaluation, we further report the number of trainable model parameters and floating-point operations (*i.e.*, FLOPs), which serve as proxies for computational complexity and indicate the scalability and deployment cost of different models in real-world industrial recommendation systems.

4.1.4 Implementation Details. We implement LazFormer with PyTorch on NVIDIA H100 GPUs using AdamW [18]. Unless otherwise specified, Transformer, HSTU, OneTrans, SORT, STCA, and LazFormer* share the same backbone configuration: 3 Transformer layers, hidden size 256, 4 attention heads, FFN dimension 1,024, and sequence length 1,024. SORT uses one shared expert and four routed experts with two activated experts.

Generative pre-training uses a 5-layer Transformer with item-ID embedding dimension 64 and temperature coefficient $\tau = 0.07$ with 16,000 in-batch negatives. Compared with LazFormer*, LazFormer scales the ranking architecture to 5 Transformer layers and sequence length 2,048, where the number of retained short-term tokens is $\ell_i = 1,024$, the compression window size is $g = 8$, and W , G_i , and $N_{\min}$ are all set to 128. All models are trained for one epoch by default, while LazFormer[†] denotes LazFormer trained with the asymmetric multi-epoch training strategy for three epochs (please refer to Section 4.4).

4.2 Offline Performance (RQ1)

Table 2 reports the offline performance of LazFormer and representative baselines. Note that for our industrial dataset, an improvement

Table 3: Ablation study of the transferable residual adapter. Sparse denotes sparse-only transfer. NoFeat removes ranking-specific features and keeps only shared features. Direct directly fuses ranking-specific features with shared features. PreProj uses a pre-projection fusion layer before the transferred input projection.

Variants	CTR		CVR	
	AUC	GAUC	AUC	GAUC
Sparse	0.7699	0.6810	0.8898	0.7109
NoFeat	0.7694	0.6843	0.8809	0.7038
Direct	0.7701	0.6817	0.8897	0.7091
PreProj	0.7699	0.6810	0.8889	0.7050
LazFormer	0.7735	0.6868	0.8905	0.7130

of 0.1 pt (*i.e.*, 0.001) in AUC or GAUC is already considered significant for CTR prediction, as also observed in [3, 35]. LazFormer* already outperforms all baselines by a clear margin, even under the aligned depth and sequence length. It also requires fewer training FLOPs than the vanilla Transformer because of the hybrid sparse attention and progressive token pruning. The full LazFormer[†] further achieves the best overall performance.

First, compared with scalable ranking Transformers (*i.e.*, HSTU and OneTrans), the gains of LazFormer are not merely from larger computation budgets, as evidenced by LazFormer* already outperforming them under the aligned configuration. Moreover, since all baselines are also initialized with the same pre-trained sparse parameters, the results show that the advantage of LazFormer does not come from sparse-only pre-trained initialization alone, but from going beyond sparse-only transfer toward transferable dense adaptation. Second, even compared with SORT, which also leverages pre-trained knowledge via sparse-only transfer, LazFormer’s transferable dense adaptation yields further gains. Finally, compared with the efficient long-sequence baseline STCA, LazFormer consistently achieves better performance. This result demonstrates the effectiveness of the proposed coarse-to-fine long-sequence modeling for capturing both short-term and long-term user preferences. Although STCA requires the fewest FLOPs by aggressively compressing the user history, it sacrifices fine-grained short-term preferences and thus underperforms LazFormer*, which we further analyze in Section 4.6.

4.3 Transferable Residual Adapter Analysis (RQ2)

To investigate whether LazFormer effectively bridges generative pre-training and downstream ranking under inconsistent feature spaces, we compare four variants that differ in whether dense parameters are transferred and how shared features and ranking-specific features are fused.

- **Sparse**: transfers only the pre-trained sparse parameters, while keeping the ranking model unchanged.
- **NoFeat**: transfers both sparse and dense parameters but removes the ranking-specific features, so the ranking input contains only the shared features used in pre-training.

Table 4: Study of the asymmetric multi-epoch training strategy, where “freeze”, “continue”, and “reset” denote sparse freezing, continuous sparse updating, and sparse reset.

Config	Loading Params		Epoch	CTR	
	Sparse	Dense		AUC	GAUC
Config-1	✗	✗	1	0.7577	0.6587
Config-2	✓, freeze	✗	1	0.7675	0.6775
Config-3	✓, continue	✗	1	0.7699	0.6810
Config-4	✓, continue	✗	3	0.7387	0.6694
Config-5	✓, reset	✗	3	0.7731	0.6851
Config-6	✓, reset	✓	3	0.7761	0.6891

- **Direct**: transfers both sparse and dense parameters, and directly fuses ranking-specific features with shared features at the tokenization stage.
- **PreProj**: transfers both sparse and dense parameters, and uses a pre-projection layer to fuse ranking-specific features with shared features before the transferred input projection.

As shown in Table 3, LazFormer consistently achieves the best performance. Compared with **Sparse**, the gain demonstrates the benefit of transferring dense parameters beyond sparse embeddings. Interestingly, **NoFeat** achieves a relatively strong CTR GAUC, suggesting that dense parameter transfer remains effective when ranking is performed using only shared features aligned with pre-training. However, its CVR performance drops substantially, indicating that shared features alone are insufficient for conversion modeling. This result suggests that ranking-specific features, especially those related to stronger purchase-intent signals such as e_j^{order} and $e_j^{\text{order-gap}}$, provide important task-specific cues for downstream conversion prediction and should not be removed merely to better align the ranking input with pre-training.

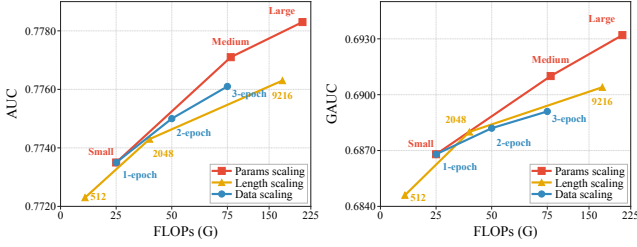
Compared with **Direct** and **PreProj**, LazFormer further shows that not only the presence of ranking-specific features but also their fusion manner is crucial for stable transfer: directly fusing ranking-specific features perturbs the input representation space learned during pre-training, while an additional randomly initialized pre-projection still perturbs the input representations fed into the pre-trained dense backbone. In contrast, the transferable residual adapter preserves the pre-trained representation at initialization via the zero-initialized residual branch and gradually incorporates ranking-specific features during optimization, thus alleviating the negative transfer of dense parameters.

4.4 Asymmetric Multi-Epoch Training Strategy Analysis (RQ3)

To investigate the effectiveness of the asymmetric multi-epoch training strategy, we compare different parameter loading and optimization strategies in Table 4. Specifically, compared with Config-1 trained from scratch, Config-2 and Config-3 achieve substantial improvements by loading pre-trained sparse parameters. Moreover, Config-3 consistently outperforms Config-2, indicating that adapting sparse parameters to the ranking objective is more effective than

Table 5: Different model scales of LazFormer.

Scale	Depth	Width	Intermediate Size	Params
Small	5	256	1024	9M
Medium	8	384	1536	23M
Large	12	512	2048	55M

**Figure 3: Scaling Laws of LazFormer. Performance scales monotonically with compute (FLOPs) across parameters, sequence length, and training data.**

freezing them during one-epoch training. We also find that continuously optimizing sparse parameters for three epochs in Config-4 leads to severe performance degradation, and Config-5 substantially outperforms Config-4 by resetting sparse parameters to their pre-trained state at the beginning of each epoch. This suggests that repeatedly updating enormous sparse parameters on the same ranking data may cause overfitting and disturb the transferable sparse knowledge. Finally, Config-6 further loads the pre-trained dense parameters and achieves the best performance. This verifies that our asymmetric multi-epoch training strategy enables dense parameters to continuously absorb ranking supervision from training data while preventing the overfitting of sparse parameters.

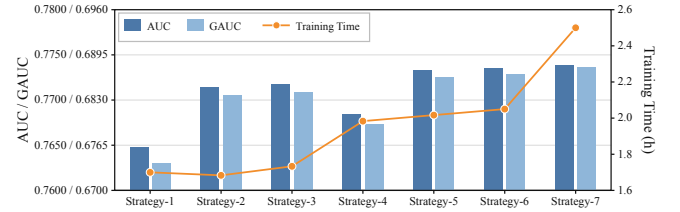
4.5 Scaling Laws Analysis (RQ4)

To investigate the scalability of LazFormer, we conduct experiments along three dimensions: model capacity (*i.e.*, parameters), training data, and sequence length. As shown in Table 5, we scale the model by jointly increasing its depth, width, and intermediate size. For data scaling, we train LazFormer for one, two, and three epochs using the asymmetric multi-epoch training strategy. For sequence length scaling, we increase the historical sequence length from 512 to 9,216. The results are presented in Figure 3.

We observe that both AUC and GAUC improve monotonically as the computational budget increases across all three scaling dimensions. Specifically, scaling LazFormer from Small to Medium and Large consistently improves performance, demonstrating that the Transformer backbone can effectively benefit from increased model capacity. Moreover, training for more epochs yields continuous gains. This observation validates that our asymmetric multi-epoch training strategy enables dense parameters to absorb more ranking supervision while preserving transferable sparse knowledge. Besides, increasing the historical sequence length also steadily improves performance, indicating that longer interaction sequences provide valuable signals for modeling user preferences.

Table 6: Configurations of different long-sequence compression strategies. ℓ_i denotes the number of retained short-term tokens, g denotes the group size for long-term compression, and $\ell_i + N_i$ denotes the sequence length after compression.

Strategy	1	2	3	4	5	6	7
ℓ_i	0	256	512	0	1024	1024	2048
g	4	8	16	2	16	8	–
$\ell_i + N_i$	512	480	608	1024	1088	1152	2048

**Figure 4: Comparison of different long-sequence compression strategies in terms of CTR prediction and training time.**

Meanwhile, long-sequence compression and hybrid sparse attention make sequence scaling computationally feasible. Overall, these results demonstrate the favorable scaling capability of LazFormer and verify that co-scaling model capacity and training data can continuously improve industrial recommendation performance.

4.6 Efficiency Analysis of Request-Aware Ranking (RQ5)

To investigate the efficiency-performance trade-off of the efficiency-oriented designs in request-aware ranking, we analyze long-sequence compression and hybrid sparse attention in turn. First, we compare seven long-sequence compression strategies listed in Table 6, with the results shown in Figure 4. We make the following observations.

First, directly compressing the entire sequence leads to suboptimal performance. In particular, although Strategy-4 retains 1,024 tokens after compression, it still underperforms Strategy-2 and Strategy-3, suggesting that fine-grained short-term sequences are essential for accurately capturing user preferences. Second, among the coarse-to-fine strategies, retaining more short-term interactions generally leads to better performance. Specifically, Strategy-6 (*i.e.*, LazFormer) preserves 1,024 short-term tokens and compresses the remaining interactions with a group size of 8, achieving the best performance among all compression strategies. Compared with the uncompressed Strategy-7, it incurs only a slight performance drop while significantly reducing training time. Overall, Strategy-6 achieves the best trade-off between recommendation performance and training efficiency, validating the effectiveness of the proposed coarse-to-fine long-sequence compression strategy.

Next, we examine hybrid sparse attention by varying the sliding-window size w and retained historical length L_{QKV} after pruning to construct four configurations with different sparsity levels. The results are presented in Figure 5.

	Sparsity-1: $w = 512, L_{QKV} = 512$		Sparsity-3: $w = 128, L_{QKV} = 512$		
	Sparsity-2: $w = 256, L_{QKV} = 512$		Sparsity-4: $w = 128, L_{QKV} = 128$		
AUC	0.7745	0.7739	0.7738	0.7736	0.7735
GAUC	0.6893	0.6873	0.6872	0.6870	0.6868
Attention Speedup	1.0×	3.7×	4.0×	4.2×	5.1×
Offline Sparsity	5.54%	88.04%	89.82%	91.02%	92.63%
	Full Self-Attention	Sparsity-1	Sparsity-2	Sparsity-3	Sparsity-4

Figure 5: Performance of hybrid sparse attention under different configurations in terms of CTR, speedup, and sparsity.

Table 7: Online A/B testing results.

Metrics	IPV	Orders	Buyers	GMV
% Improve ↑	+5.21%	+3.38%	+3.70%	+9.85%

We observe that all sparse configurations substantially accelerate attention computation while maintaining competitive CTR performance. Specifically, Sparsity-1 achieves a 3.7× speedup with 88.04% sparsity, while incurring only marginal AUC and GAUC degradation compared with full self-attention. As the sliding-window size decreases from Sparsity-1 to Sparsity-3, the sparsity and speedup continuously increase, whereas the performance declines only slightly. This indicates that attending to a limited local window is sufficient to capture most useful sequential dependencies. Moreover, further pruning the retained historical representations in Sparsity-4 increases the sparsity to 92.63% and achieves a 5.1× attention speedup. We adopt Sparsity-4 because reallocating its saved compute to depth, sequence length, and epochs outweighs 0.10 pt AUC cost, as shown in Figure 3. These results demonstrate that the hybrid sparse attention proposed in LazFormer effectively removes redundant attention computation, achieving a favorable trade-off between recommendation performance and training efficiency.

4.7 Online A/B Testing (RQ6)

We deploy LazFormer in the ranking stage of the homepage recommendation system on a large-scale industrial e-commerce platform, where the online production baseline is a 3-layer SORT-like [27] ranking model. A two-week online A/B test is conducted, and the results are reported in Table 7. Compared with the baseline, LazFormer achieves relative improvements of 5.21% in IPV (*i.e.*, item page view), 3.38% and 3.70% in the numbers of orders and buyers, respectively, and 9.85% in GMV. These consistent gains across engagement, conversion, and revenue metrics demonstrate the practical effectiveness and commercial value of LazFormer in real-world industrial recommendation.

Although LazFormer incurs additional online computation due to its deeper Transformer backbone (from 3 to 5 layers) and the modeling of longer sequences, the overhead is effectively controlled by its

efficient sequence modeling design, including long-sequence compression and hybrid sparse attention. As a result, the single-machine throughput (measured in queries per second) on an NVIDIA A10 GPU decreases by only 12.1%. Given the substantial business gains, LazFormer demonstrates strong practicality and a favorable return on investment for large-scale industrial recommendation.

5 Related Work

Traditional industrial recommendation models generally consist of two major components: user sequence modeling and feature interaction. Early sequence modeling methods [5, 10, 34, 35] mainly focus on capturing users’ evolving interests from relatively short behavior sequences. To incorporate longer behavior histories, subsequent studies [3, 4, 23, 29] develop specialized retrieval and efficient encoding mechanisms. More recently, LONGER [2] improves the scalability of long-sequence encoding through efficient attention, hierarchical aggregation, and serving-oriented optimization. For feature interaction, early methods [6, 12, 19, 25, 28] capture implicit or explicit feature interactions through shallow neural networks. More recently, Wukong [32] and RankMixer [17, 36] explore scaling feature interaction networks by increasing model parameters and computational budgets. However, these methods cannot be expanded with a shared Transformer backbone, preventing them from fully exploiting the scaling capability of Transformer.

Inspired by the scaling capability of Transformer in LLMs, recent studies [13, 15, 16, 24, 31, 33] have explored **scalable Transformer** architectures for industrial recommendation. HSTU [31] and MTGR [13] reformulate recommendation as generative sequential modeling and demonstrate the potential of Transformers for large-scale recommendation. OneTrans [33] employs a shared Transformer to unify feature interaction and sequence modeling. MixFormer [15] and HyFormer [16] further improve heterogeneous feature interaction and long-sequence modeling within scalable Transformer-based architectures. These studies demonstrate that recommendation performance can continuously benefit from scaling model depth, width, and sequence length. However, most existing methods conduct scaling within a single ranking model and jointly optimize sparse and dense parameters from scratch, resulting in substantial computational resource consumption and slow convergence. Recently, pre-training methods [26, 27] introduce pre-trained sparse parameters into ranking, they typically freeze the sparse parameters, limiting their adaptation to the ranking process. In contrast, LazFormer introduces a transferable residual adapter to mitigate the negative transfer of dense parameters and combines request-aware ranking with an asymmetric multi-epoch training strategy to enable the co-scaling of model capacity and training data while alleviating the overfitting of sparse parameters.

6 Conclusion

In this paper, we propose a scaling Transformer for industrial recommendation via transferable generative pre-training, named LazFormer. We first propose a generative pre-training module to autoregressively generate sequence features, providing favorable initialization for both sparse and dense parameters in the subsequent ranking process. Additionally, we introduce a transferable residual adapter to incorporate ranking-specific features while preserving

the pre-trained knowledge, thus solving the negative transfer of dense parameters. Moreover, the request-aware ranking module combines long-sequence compression, hybrid sparse attention, and a request-aware paradigm to efficiently model users' long interaction sequences. Besides, an asymmetric multi-epoch training strategy is developed to scale the training data by continuously accumulating dense parameters, while resetting sparse parameters to alleviate overfitting. Extensive offline experiments and online A/B testing demonstrate the effectiveness and scalability of LazFormer. LazFormer has been deployed in an industrial recommendation system, delivering significant gains for the platform.

References

- [1] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).
- [2] Zheng Chai, Qin Ren, Xijun Xiao, Huizhi Yang, Bo Han, Sijun Zhang, Di Chen, Hui Lu, Wenlin Zhao, Lele Yu, et al. 2025. Longer: Scaling up long sequence modeling in industrial recommenders. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems*. 247–256.
- [3] Jianxin Chang, Chenbin Zhang, Zhiyi Fu, Xiaoxue Zang, Lin Guan, Jing Lu, Yiqun Hui, Dewei Leng, Yanan Niu, Yang Song, et al. 2023. TWIN: Two-stage interest network for lifelong user behavior modeling in CTR prediction at kuaishou. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3785–3794.
- [4] Qiwei Chen, Changhua Pei, Shanshan Lv, Chao Li, Junfeng Ge, and Wenwu Ou. 2021. End-to-end user behavior retrieval in click-through rate prediction model. *arXiv preprint arXiv:2108.04468* (2021).
- [5] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior sequence transformer for e-commerce recommendation in alibaba. In *Proceedings of the 1st international workshop on deep learning practice for high-dimensional sparse data*. 1–4.
- [6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishikesh Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ipsir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.
- [7] Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. 2024. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496* (2024).
- [8] Alin Fan, Hanqing Li, Sihan Lu, Jingsong Yuan, and Jiantong Zhang. 2026. Decoupled multimodal fusion for user interest modeling in click-through rate prediction. In *2026 IEEE 42nd International Conference on Data Engineering (ICDE)*. IEEE, 4034–4046.
- [9] Zhongxiang Fan, Zhaocheng Liu, Jian Liang, Dongying Kong, Han Li, Peng Jiang, Shuang Li, and Kun Gai. 2024. Multi-epoch learning with data augmentation for deep click-through rate prediction. *arXiv preprint arXiv:2407.01607* (2024).
- [10] Yufei Feng, Fuyu Lv, Weichen Shen, Menghan Wang, Fei Sun, Yu Zhu, and Keping Yang. 2019. Deep session interest network for click-through rate prediction. *arXiv preprint arXiv:1905.06482* (2019).
- [11] Lin Guan, Jia-Qi Yang, Zhishan Zhao, Beichuan Zhang, Bo Sun, Xuanyuan Luo, Jinan Ni, Xiaowen Li, Yuhang Qi, Zhifang Fan, et al. 2026. Make it long, keep it fast: End-to-end 10k-sequence modeling at billion scale on Douyin. In *Proceedings of the ACM Web Conference 2026*. 7989–7998.
- [12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).
- [13] Ruidong Han, Bin Yin, Shangyu Chen, He Jiang, Fei Jiang, Xiang Li, Chi Ma, Mincong Huang, Xiaoguang Li, Chunzhen Jing, et al. 2025. Mtgr: Industrial-scale generative recommendation framework in meituan. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 5731–5738.
- [14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *Iclr* 1, 2 (2022), 3.
- [15] Xu Huang, Hao Zhang, Zhifang Fan, Yunwen Huang, Zhuoxing Wei, Zheng Chai, Jinan Ni, Yuchao Zheng, and Qiwei Chen. 2026. MixFormer: Co-Scaling Up Dense and Sequence in Industrial Recommenders. *arXiv preprint arXiv:2602.14110* (2026).
- [16] Yunwen Huang, Shiyong Hong, Xijun Xiao, Jinqiu Jin, Xuanyuan Luo, Zhe Wang, Zheng Chai, Shikang Wu, Yuchao Zheng, and Jingjian Lin. 2026. HyFormer: Revisiting the Roles of Sequence Modeling and Feature Interaction in CTR Prediction. *arXiv preprint arXiv:2601.12681* (2026).
- [17] Yuchen Jiang, Jie Zhu, Xintian Han, Hui Lu, Kunmin Bai, Mingyu Yang, Shikang Wu, Ruihao Zhang, Wenlin Zhao, Shipeng Bai, et al. 2026. Tokenmixer-large: Scaling up large ranking models in industrial recommenders. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 7489–7500.
- [18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [19] Xiangyang Li, Bo Chen, HuiFeng Guo, Jingjie Li, Chenxu Zhu, Xiang Long, Sujian Li, Yichao Wang, Wei Guo, Longxia Mao, et al. 2022. Inttower: the next generation of two-tower model for pre-ranking system. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 3292–3301.
- [20] Xiaodong Li, Ruochen Yang, Shuang Wen, Shen Wang, Yueyang Liu, Guoquan Wang, Weisong Hu, Qiang Luo, Jiawei Sheng, Tingwen Liu, et al. 2026. FARM: Frequency-Aware Model for Cross-Domain Live-Streaming Recommendation. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*. 2302–2311.
- [21] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Cheng-gang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437* (2024).
- [22] Zhaocheng Liu, Zhongxiang Fan, Jian Liang, Dongying Kong, and Han Li. 2023. Multi-Epoch Learning for Deep Click-Through Rate Prediction Models. *arXiv preprint arXiv:2305.19531* (2023).
- [23] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based user interest modeling with lifelong sequential behavior data for click-through rate prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*.
- [24] Zhenhao Song, Yufeng Gao, Xing Fang, Jing Wang, Guangxin Song, Bokang Wang, Yipin Dai, and He Guo. 2026. TMallGS: Scaling Unified Feature and Sequence Modeling for Generative E-commerce Search. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 7989–7998.
- [25] Liangcai Su, Junwei Pan, Ximei Wang, Xi Xiao, Shijie Quan, Xihua Chen, and Jie Jiang. 2024. STEM: unleashing the power of embeddings for multi-task recommendation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 38. 9002–9010.
- [26] Chunqi Wang, Bingchao Wu, Zheng Chen, Lei Shen, Bing Wang, and Xiaoyi Zeng. 2025. Scaling transformers for discriminative recommendation via generative pretraining. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 2893–2903.
- [27] Chunqi Wang, Bingchao Wu, Taotian Pang, Jiahao Wang, Jie Yang, Jia Liu, Hao Zhang, Hai Zhu, Lei Shen, Shizhun Wang, et al. 2026. SORT: A Systematically Optimized Ranking Transformer for Industrial-scale Recommenders. *arXiv preprint arXiv:2603.03988* (2026).
- [28] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.
- [29] Xue Xia, Pong Eksombatchai, Nikil Pancha, Dhruvil Deven Badani, Po-Wei Wang, Neng Gu, Saurabh Vishwas Joshi, Nazanin Farahpour, Zhiyuan Zhang, and Andrew Zhai. 2023. Transact: Transformer-based realtime user action model for recommendation at pinterest. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5249–5259.
- [30] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).
- [31] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. 2024. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. *arXiv preprint arXiv:2402.17152* (2024).
- [32] Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Daifeng Guo, Yanli Zhao, Shen Li, Yuchen Hao, Yantao Yao, et al. 2024. Wukong: Towards a scaling law for large-scale recommendation. *arXiv preprint arXiv:2403.02545* (2024).
- [33] Zhaoqi Zhang, Haolei Pei, Jun Guo, Tianyu Wang, Yufei Feng, Hui Sun, Shaowei Liu, and Aixin Sun. 2026. Onetrans: Unified feature interaction and sequence modeling with one transformer in industrial recommender. In *Proceedings of the ACM Web Conference 2026*. 8162–8170.
- [34] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*.
- [35] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1059–1068.
- [36] Jie Zhu, Zhifang Fan, Xiaoxie Zhu, Yuchen Jiang, Hangyu Wang, Xintian Han, Haoran Ding, Xinmin Wang, Wenlin Zhao, Zhen Gong, et al. 2025. Rankmixer: Scaling up ranking models in industrial recommenders. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*. 6309–6316.