RankMixer: Scaling Up Ranking Models in Industrial Recommenders

Jie Zhu*, Zhifang Fan*, Xiaoxie Zhu*, Yuchen Jiang*, Hangyu Wang, Xintian Han, Haoran Ding, Xinmin Wang, Wenlin Zhao, Zhen Gong, Huizhi Yang, Zheng Chai, Zhe Chen, Yuchao Zheng, Qiwei Chen†, Feng Zhang, Xun Zhou, Peng Xu, Xiao Yang, Di Wu, Zuotao Liu

ByteDance

{zhujie.zj, zhuxiaoxie.777, jiangyuchen.jyc, wanghangyu.123, hanxintian, dinghaoran, xinmin.wang, zhaowenlin, gongzhen.666, yanghuizhi, chaizheng.cz, chenzhe.john, zhengyuchao.yc, feng.zhang, zhouxun, xupeng, wuqi.shaw, di.wu, michael.liu}@bytedance.com, {fanzhifangfzf, chenqiwei05}@gmail.com

Abstract

Recent progress on large language models (LLMs) has spurred interest in scaling up recommendation systems, yet two practical obstacles remain. First, training and serving cost on industrial Recommenders must respect strict latency bounds and high QPS demands. Second, most human-designed feature-crossing modules in ranking models were inherited from the CPU era and fail to exploit modern GPUs, resulting in low Model Flops Utilization (MFU) and poor scalability. We introduce RankMixer, a hardware-aware model design tailored towards a unified and scalable feature-interaction architecture. RankMixer retains the transformer's high parallelism while replacing quadratic self-attention with multi-head token mixing module for higher efficiency. Besides, RankMixer maintains both the modeling for distinct feature subspaces and cross-feature-space interactions with Per-token FFNs. We further extend it to one billion parameters with a Sparse-MoE variant for higher ROI. A dynamic routing strategy is adapted to address the inadequacy and imbalance of experts training. Experiments show RankMixer's superior scaling abilities on a trillion-scale production dataset. By replacing previously diverse handcrafted low-MFU modules with RankMixer, we boost the model MFU from **4.5%** to **45%**, and scale our online ranking model parameters by two orders of magnitude while maintaining roughly the same inference latency. We verify RankMixer's universality with online A/B tests across two core application scenarios (Recommendation and Advertisement). Finally, we launch 1B Dense-Parameters RankMixer for full traffic serving without increasing the serving cost, which improves user active days by **0.3%** and total in-app usage duration by **1.08%**.

CCS Concepts

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

Keywords

Scaling Laws, Ranking Model, Recommender System

1 Introduction

Recommender System (RS) is essential in the process of distributing information. As a significant machine learning scenario, RS predicts users' behaviors towards items based on a large amount of multi-field feature data, including numerical features such as diverse statistics, categorical features such as user and item IDs, user behavior features and content features [19, 41]. The state-of-the-art recommendation methods are based on Deep Learning Recommendation Models (DLRMs), which flexibly capture feature interactions based on neural networks as the dense interaction layers above the input embeddings from features. The Dense interaction layer in DLRM is critical for RS performance and diverse model structures are proposed [7, 11, 33, 38, 42].

Driven by advancements in Large Language Models (LLMs) that benefit from increasing parameters [1, 14, 16], scaling up DLRMs to take full advantage of the volume of data is an urgent need. Previous research has yielded numerous outcomes on the scaling DLRMs, early studies [2, 6, 40] just widen or stack feature interaction layers without modifying the structure. The benefits achieved in this way are modest and occasionally negative [18, 32]. Then the follow-up efforts, such as DHEN [39] and Wukong [38], focus on designing innovative DNN structures to boost the scaling performance. However, leveraging model scale to improve performance in recommendation presents unique practical challenges. Unlike NLP or vision tasks, industrial-scale recommendation systems must strictly adhere to tight latency constraints and support extremely high QPS (queries per second). As a result, the core challenge lies in finding a sweet spot that balances model effectiveness and computational efficiency.

Historically, the architecture of ranking models in recommendation has been shaped by CPU-era design principles. These models typically relied on combining heterogeneous diverse handcrafted cross-feature modules to extract feature interactions, but many of their core operators are memory-bound rather than compute-bound on modern GPUs, leading to poor GPU parallelism and extremely low MFU(Model Flops Utilization) often in the single-digit percentage range.Moreover, since the computational cost of CPU-era models was approximately proportional to the number of parameters, the potential ROI from aggressive scaling—as suggested by scaling laws—was difficult to realize in practice.

In summary, the research on scaling laws of DLRMs must overcome the following problems:

- Architectures should be hardware-aligned, maximizing MFU and compute throughput on modern GPUs.
- The model design must leverage the characteristics of recommendation data, such as the heterogeneous feature spaces and personalized cross-feature interactions among hundreds of fields.

To address these challenges, we propose a hardware-aware model design method, RankMixer. The core design of RankMixer is based on two scalable components: 1. *Multi-head token mixing* achieves

* Equal Contributions.
† Corresponding authors.

cross-token feature interactions only through the parameter-free operator. This strategy outperforms the self-attention mechanism in terms of performance and computational efficiency. 2. *Per-token feed-forward networks (FFNs)* expand model capacity significantly and tackle inter-feature-space domination problem by allocating independent parameters for different feature subspace modeling. These FFNs also align well with the recommendation data patterns, enabling a better scaling behavior. To further boost the ROI of large-scale models, we extend the per-token FFN modules into a *Sparse Mixture-of-Experts (MoE)* structure. By dynamically activating only a subset of experts per token for different data, we can significantly increase the model capacity with a minimal increase in computational cost. RankMixer adopts a highly parallel architecture similar to transformers, but overcomes several critical limitations of self-attention based feature-interaction: low training efficiency, the combinatorial explosion when modeling cross-space ID similarity and severe memory-bound caused by attention weights matrix. At the same time, RankMixer offers greater model capacity and learning capability under the same Flops compared with Vanilla Transformer.

In production deployment on Douyin's recommendation system, we demonstrate that it is feasible to scale model parameters by over 100× while maintaining shorter inference latency compared with previous baseline. This is made possible by the RankMixer architecture's ability to decouple parameter growth from FLOPs, and decouple FLOPs growth from actual cost through high MFU and engineering optimization.

The main contributions can be summarized as follows:

- We propose a novel architecture called RankMixer follows a hardware-aware model-design philosophy. We design the multi-head token mixing and per-token FFN strategies to capture heterogeneous feature interactions efficiently, and using dynamic routing strategy to improve the scalability of SparseMoE in RankMixer.
- Leveraging levers of high MFU and performance-optimization, we scale the model parameters by 70× without increasing the inference cost, including improving MFU and Quantization.
- We conduct extensive offline and online experiments and investigate the model's scaling law on the trillion-scale industrial recommendation dataset. The RankMixer model has been successfully deployed on the Douyin Feed Recommendation Ranking for full-traffic serving, achieving 0.3% and 1.08% increase on active days and App duration, respectively.

2 Related Work

Modern recommendation systems are based on Deep Learning Recommendation Models (DLRMs) and how to effectively model the feature interactions is a crucial factor for DLRMs [5, 11, 15, 22, 31, 33, 39]. Wide&Deep [5] is one of the earliest efforts, which combines the logistic regression (wide part) and DNN (deep part) to capture low-order and high-order feature interactions respectively. DeepFM [11] is another achievement, which integrates the Factorization Machine (FM) and the DNN. In addition, DeepCross [26] is an extension of the residual network [13], aiming to learn automatic feature interactions implicitly. While it has proven to be quite challenging to simply rely on DNNs to learn high-order feature interactions [22, 25]. Explicit cross methods design different operators to explicitly capture high-order feature interactions, such as PNN [22], DCN [32] and its successor DCNv2 [33], xDeepFM [18], FGCNN [20] and FiGNN [17]. AutoInt [28] and Hiformer [10] employ attention mechanism with residual connections to learn complex interactions. DHEN [39] proposes to combine multiple interaction operators together. Despite improving accuracy, these new architectures increase the model latency and memory consumption and their model size is relatively small.

Scaling law has emerged as a fundamental theme in deep learning and a pivotal catalyst for numerous breakthroughs over the past decade, particularly in Natural Language Processing (NLP) [14, 16], Computer Vision (CV) [8, 37], and multi-modality modeling [23, 24], which describe the power-law correlations between model performance and scaling factors, such as model size, data volume, and computational capacity [1, 16, 29, 30]. Recently, scaling laws in recommendation systems have drawn much attention from researchers [12]. Studies have explored the scaling strategy for pre-training user activity sequences [6], general-purpose user representations [27, 40] and online retrieval [9, 34]. Wukong [38] stacks FMs and LCB to learn the feature interactions. Orthogonally, Zhang et al. [40] scaled up a sequential recommendation model to 0.8B parameters. HSTU [36] enhances the scaling effect of Generative Recommenders (GRs) which focus more on the sequence part.

3 Methodology

3.1 Overall Architecture

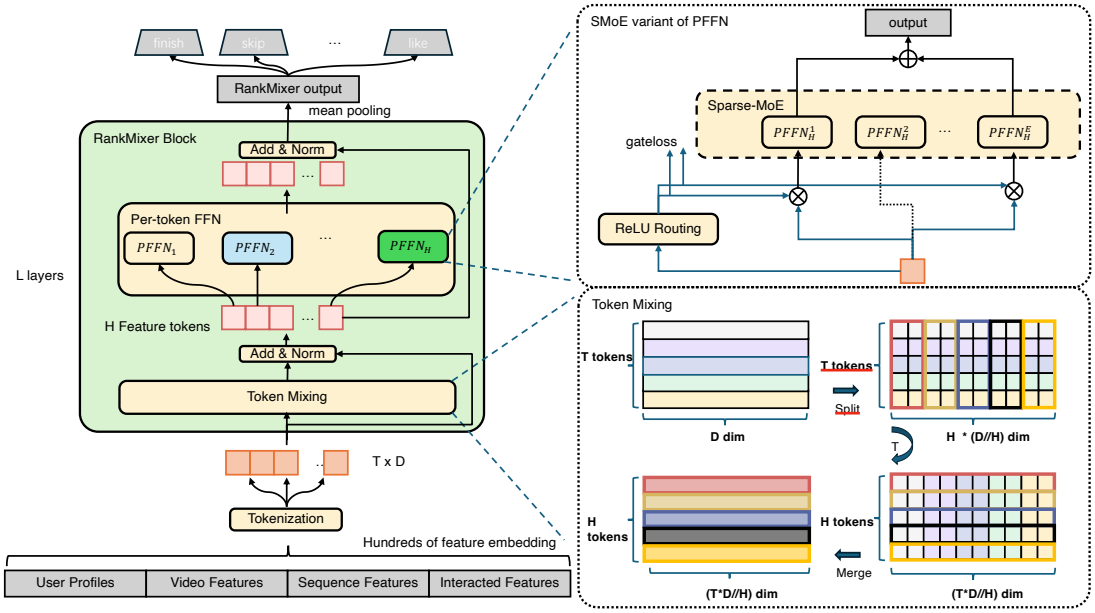
RankMixer's overall architecture consists of T input tokens processed by L successive RankMixer blocks, followed by an output pooling operator. Each RankMixer block has two main components: (1) a Multi-Head Token Mixing layer, and (2) a Per-Token Feed-Forward Network (PFFN) layer, as illustrated in Figure 1 . First, the input vector $\mathbf{e}_{\text{input}}$ is tokenized into T feature tokens $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$, each representing a coherent feature vector. RankMixer blocks iteratively refine token representations for L layers through:

$$\begin{aligned} \mathbf{S}_{n-1} &= \text{LN}\left(\text{TokenMixing}\left(\mathbf{X}_{n-1}\right) + \mathbf{X}_{n-1}\right), \\ \mathbf{X}_{n} &= \text{LN}\left(\text{PFFN}\left(\mathbf{S}_{n-1}\right) + \mathbf{S}_{n-1}\right), \end{aligned} \tag{1}$$

where $\text{LN}(\cdot)$ is a layer normalization function, $\text{TokenMixing}(\cdot)$ and $\text{PFFN}(\cdot)$ are the multi-head Token Mixing module and the per-token FFN module, $\mathbf{X}_n \in \mathbb{R}^{T \times D}$ is the output of the n-th RankMixer block, $\mathbf{X}_0 \in \mathbb{R}^{T \times D}$ is stacked by $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$, and D is the hidden dimension of the model. The output representation $\mathbf{o}_{\text{output}}$ derives from the mean pooling of final layers's representations $\mathbf{X}_L$ which will be used to compute the different task's predictions.

3.2 Input Layer and Feature Tokenization

The first thing to build a large-scale recommendation model is to prepare inputs with abundant information, such as **User features** include user ID and other user information, **candidate features** include video ID, author ID etc., **Sequence Features** processed through Sequence Module [4, 42] to capture temporal interests, yielding $\mathbf{e}_s$ and **Cross Features**. the cross features between the user and the candidate. All the features will be transformed into embeddings with diverse dimensions.



这里的T Tokens论文说按照语义分组拼接，然后再拿去做Split
分组策略（数量，内部特征），组内feature拼接的顺序都会影响到信息流交互部分
那么分组策略，以及同一个Token内部feature拼接策略，有没有技巧?

Figure 1: The architecture of a RankMixer block. A RankMixer block consists of two modules: Multi-head Token Mixing and SMoE based Per-token FFN. The token-mixing module divides each token's embedding into H smaller parts (heads), then recombines these parts across tokens to create new mixed tokens. This allows information from different features to interact with each other.

To enable efficient parallel computation in later stages, embeddings of varying dimensions must be transformed into dimension-aligned vectors, called feature-tokens. We refer to this embedding alignment process as tokenization. The simplest strategy is assigning one embedding per feature which introduces several challenges given typically hundreds of features. Having hundreds of tokens inevitably reduces the parameters and computation allocated per token to small fragments, resulting in insufficient modeling of important features and under-utilization of GPU cores. Conversely, too few tokens (e.g. a single token) degrade the model structure into a simple Deep Neural Network (DNN), unable to distinctly represent diverse feature spaces, which risks dominant features overshadowing others.

To overcome these issues, we propose a semantic-based tokenization approach with domain knowledge and group features into several semantically coherent clusters. These grouped features are sequentially concatenated into one embedding vector $e_{\text{input}} = [e_1; e_2; \ldots; e_N]$, and subsequently partitioned into an appropriate number of tokens with fixed dimension sizes. Each feature token $x_i \in \mathbb{R}^D$ captures a set of feature embeddings that represent a similar semantic aspect.

$$\mathrm{x}_i = \mathrm{Proj}(e_{\text{input}}[d \cdot (i-1) : d \cdot i]), \quad i = 1, \ldots, T, \tag{2}$$

where e_{input} is the concatenated embedding vector, d is the fixed dimension per token, N is the number of feature-groups, and T is the resulting token count and the Proj function maps the splitted embedding into D dimension.

3.3 RankMixer Block

3.3.1 Multi-head Token Mixing. To facilitate effective information exchange across tokens which is important for feature cross and global information modeling, we introduce the multi-head Token Mixing module. Each token is evenly divided into H heads, with the h-th head of token $\mathbf{x}_t$ denoted as x_t^h:

$$\left[\mathbf{x}_t^{(1)} \parallel \mathbf{x}_t^{(2)} \parallel \ldots \parallel \mathbf{x}_t^{(H)}\right] = \mathrm{SplitHead}\left(\mathbf{x}_t\right). \tag{3}$$

These heads can be seen as projections of the token $\mathbf{x}_t$ into a lower-dimensional feature subspace since recommendation is task taking different perspectives into considerations. Token Mixing is used to fuse these sub-space vector for global feature interactions. Formally, the h-th token $\mathbf{s}^h$ corresponding to the h-th head after the multi-head Token Mixing is constructed as:

$$\mathbf{s}^h = \mathrm{Concat}\left(\mathbf{x}_1^h, \mathbf{x}_2^h, ..., \mathbf{x}_T^h\right). \tag{4}$$

The output of the multi-head Token Mixing module is $\mathbf{S} \in \mathbb{R}^{H \times \frac{TD}{H}}$, which is stacked by all the shuffled tokens $\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_H$. In this work, we set $H = T$ to maintain the same number of tokens after Token Mixing for residual connection.

After a residual connection and normalization module we can generate

$$\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_T = \text{LN}\left(\text{TokenMixing}\left(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\right) + \left(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\right)\right), \tag{5}$$

Although self-attention has proven highly effective in large language models, we find it to be suboptimal for recommendation systems. In self-attention, attention weights are calculated using the inner product of tokens. This method works well for NLP, where all tokens share a unified embedding space. However, in recommendation tasks, the feature spaces are inherently heterogeneous. Computing an inner-product similarity between two heterogeneous semantic spaces is notoriously difficult—particularly in recommender systems, where the ID space of the features from user and item side may contain hundreds of millions of elements. Consequently, applying self-attention to such diverse inputs does not outperform the parameter-free multi-head Token Mixing approach and consumes more computations, Memory IO operations and GPU memory usage.

3.3.2 Per-token FFN. Previous DLRM and DHEN models tend to mix features from many disparate semantic spaces in a single interaction module, which may causes high-frequency fields dominate drowning out lower-frequency or long-tail signals and ultimately hurting overall recommendation quality. We introduce a parameter-isolated feed-forward network architecture, termed per-token FFN. In traditional designs, the parameters of FFN are shared across all tokens, but our approach processes each token with dedicated transformations, thus isolating the parameters for each token. For the t-th token $\mathbf{s}_t$, the per-token FFN can be expressed as

$$\mathbf{v}_t = f^{t,2}_{\text{pffn}}\left(\text{Gelu}\left(f^{t,1}_{\text{pffn}}\left(\mathbf{s}_t\right)\right)\right), \tag{6}$$

where

$$f^{t,i}_{\text{pffn}}(\mathbf{x}) = \mathbf{x}\mathbf{W}^{t,i}_{\text{pffn}} + \mathbf{b}^{t,i}_{\text{pffn}} \tag{7}$$

is the i-th layer MLP of the per-token FFN, $\mathbf{W}^{t,1}_{\text{pffn}} \in \mathbb{R}^{D \times kD}$, $\mathbf{b}^{t,1}_{\text{pffn}} \in \mathbb{R}^{kD}$, $\mathbf{W}^{t,2}_{\text{pffn}} \in \mathbb{R}^{kD \times D}$, $\mathbf{b}^{t,2}_{\text{pffn}} \in \mathbb{R}^{D}$, k is a hyperparameter to adjust the hidden dimension of the per-token FFN, Gelu$(\cdot)$ is the Gelu activation function, $\mathbf{s}_t \in \mathbb{R}^D$ is the t-th token.

We summarize the Per-token FFN module as

$$\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_T = \text{PFFN}\left(\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_T\right), \tag{8}$$

where

$$\text{PFFN}\left(\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_T\right) = f^{t,2}_{\text{pffn}}\left(\text{Gelu}\left(f^{t,1}_{\text{pffn}}\left(\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_T\right)\right)\right). \tag{9}$$

Compared to the parameter-all-shared FFN, per-token FFN enhances the modeling ability by introducing more parameters while keeping the computational complexity unchanged.

It is worth emphasizing that a Per-token FFN differs from MMoE experts in that each Per-token FFN sees a distinct token input, whereas all experts in a MMoE share the same input. Unlike a MMoE, where many experts process the same input, and unlike a Transformer, where different input share one FFN, RankMixer splits the inputs and the parameters simultaneously which is good for learning diversity in different feature sub-spaces.

3.4 Sparse MoE in RankMixer

To further increase the scaling ROI, we can replace the dense FFNs of each pertoken with *Sparse Mixture-of-Experts* (MoE) blocks, so that the model's capacity grows while computation cost remains roughly constant. Vanilla Sparse-MoE, however, degrades in RankMixer because: (i) **uniform k-expert routing**. Top-k selection treats all feature tokens equally, wasting the budget on low-information tokens and starving high-information ones, which hinders the model from capturing differences between tokens. (ii) **expert under-training**. Per-token FFNs already multiply parameters by #tokens; adding non-shared experts further explodes the expert count, producing highly unbalanced routing and poorly trained experts;

We combine two complementary training strategies to solve the above problems.

ReLU Routing. To grant tokens flexible expert counts and maintain differentiability, we replace the common Top$k$+softmax with a ReLU gate plus an adaptive ℓ_1 penalty [35]. Given the j-th expert $e_{i,j}(\cdot)$ for token $s_i \in \mathbb{R}^{d_h}$ and router $h(\cdot)$:

$$G_{i,j} = \text{ReLU}\big(h(\mathbf{s}_i)\big), \quad \mathbf{v}_i = \sum_{j=1}^{N_e} G_{i,j}\, e_{i,j}(\mathbf{s}_i), \tag{10}$$

where N_e is the number of experts per token, N_t is the number of tokens. ReLU Routing will activate more experts for high-information tokens and improve the parameter efficiency. Sparsity is steered by $\mathcal{L}_{\text{reg}}$ with an coefficient λ that keeps the average active-expert ratio near the budget:

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda\, \mathcal{L}_{\text{reg}}, \quad \mathcal{L}_{\text{reg}} = \sum_{i=1}^{N_t} \sum_{j=1}^{N_e} G_{i,j}. \tag{11}$$

Dense-training / Sparse-inference (DTSI-MoE). Inspired by [21], two routers h_{train} and h_{infer} are adopted, and $\mathcal{L}_{\text{reg}}$ is applied only to h_{infer}. Both h_{train} and h_{infer} are updated during training, while only h_{infer} is used in inference. It turns out that DS-MoE makes experts do not suffer from under-training while reducing inference cost.

3.5 Scaling Up Directions

RankMixer is intrinsically a highly parallel and extensible architecture. Its parameter count and computational cost can be expanded along four orthogonal axes: Token count T, Model width D, Layers L and Expert Numbers E. For full-dense-activated version, parameter and forward FLOPs for one sample can be computed as

$$\#\text{Param} \;\approx\; 2\,k\,L\,T\,D^2, \qquad \text{FLOPs} \;\approx\; 4\,k\,L\,T\,D^2, \tag{12}$$

The k is the scale ratio adjusting the hidden dimension of FFN. In the *Sparse-MoE* version, the *effective* parameters and compute per token are further scaled by the sparsity ratio $s = \frac{\#\text{Activated\_Param}}{\#\text{Total\_Param}}$.

4 Experiments

4.1 Experiment Settings

4.1.1 Datasets and Environment. The offline experiments were conducted using the training data from the Douyin recommendation system. These data are derived from Douyin's online logs and user feedback labels. The training dataset includes over 300 features, such as numerical features, ID features, cross features, and sequential features, involving billions of user IDs and hundreds of millions of video IDs, which are all converted into embeddings. The data covers trillions of records per day, and experiments were conducted on data collected over a two-week period.

4.1.2 Evaluation Metrics. To evaluate model performance, we use AUC (Area Under the Curve) and UAUC (User-level AUC) as the primary performance metrics and Parameter count, FLOPs and MFU as the efficiency metrics listed as follows: **Finish/Skip AUC/UAUC:** A finish=1/0 or skip=1/0 label indicates whether a user completed watching a video or slide to the next video in a short-time. We evaluate the AUC and UAUC for this finish label. an AUC increase of 0.0001 can be regarded as a confidently significant improvement. **Dense-Param:** The number of parameters in the dense part, excluding the sparse embedding parameters. Training **Flops/Batch:** The number of floating-point operations (FLOPs) required to run a single batch of 512 through the model, representing the computational cost of training. **MFU:** MFU(Model FLOPs Utilization) is a metric that measures how effectively a model utilizes floating-point operations provided by the hardware, calculated by dividing the model's actual FLOPs consumption by the hardware's theoretical FLOPs capacity.

4.1.3 Baselines. We compare against the following widely recognized SOTA baselines: **DLRM-MLP**:which is the vanilla MLP for feature crossing as the experiment baseline, **DCNv2**[33],**RDCN**[3] the Sota of feature cross model. **MoE** scales up by using mutliply parallel experts. **AutoInt**[28], **Hiformer**[10] combines heterogeneous self-attention layer and low-rank approximation matrix computation. **DHEN**[39]: which combines different kind of feature-cross block and stacks multiple layers (DCN/self-attention/FM/LR) **Wukong**[38] investigates the scaling law of feature interaction following the arch of DHEN with Factorization Machine Block (FMB) and Linear Compress Block (LCB).

All experiments were conducted on hundreds of GPUs in a hybrid distributed training framework that the sparse part is updated asynchronously, while the dense part is updated synchronously. The optimizer hyperparameters were kept consistent across all models. For the dense part, we used the RMSProp optimizer with a learning rate of 0.01, while the sparse part used the Adagrad optimizer.

4.2 Comparison with SOTA methods

To explore how to scale up the model, we compare models with similar parameter sizes around 100 million to determine which model structure performs best with the same computational cost.

The main results of the performance of our method and baselines are summarized in Table 1. We can observe RankMixer significantly outperforms other SOTA models across multiple objectives and metrics.

Table 1: Performance & efficiency comparison of ~100 M-parameter recommendation models (best values in bold).

Model	Finish		Skip		Efficiency	
	AUC↑	UAUC↑	AUC↑	UAUC↑	Params	FLOPs/Batch
DLRM-MLP (base)	0.8554	0.8270	0.8124	0.7294	8.7 M	52 G
DLRM-MLP-100M	+0.15%	—	+0.15%	—	95 M	185 G
DCNv2	+0.13%	+0.13%	+0.15%	+0.26%	22 M	170 G
RDCN	+0.09%	+0.12%	+0.10%	+0.22%	22.6 M	172 G
MoE	+0.09%	+0.12%	+0.08%	+0.21%	47.6 M	158 G
AutoInt	+0.10%	+0.14%	+0.12%	+0.23%	19.2 M	307 G
DHEN	+0.18%	+0.26%	+0.36%	+0.52%	22 M	158 G
HiFormer	+0.48%	—	—	—	116 M	326 G
Wukong	+0.29%	+0.29%	+0.49%	+0.65%	122 M	442 G
RankMixer-100M	**+0.64%**	**+0.72%**	**+0.86%**	**+1.33%**	107 M	233 G
RankMixer-1B	**+0.95%**	**+1.22%**	**+1.25%**	**+1.82%**	**1.1 B**	**2.1T**

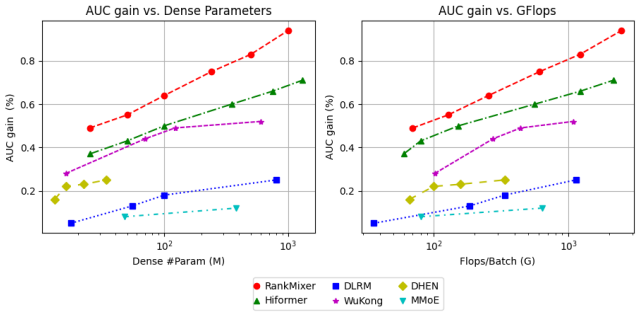


Figure 2: Scaling laws between finish Auc-gain and Params/Flops of different models. The x-axis uses a logarithmic scale.

We take a closer look at each model. First, simply scaling DLRM up to 100 million parameters yields only limited gains, underscoring the necessity of designing models tailored to recommendation data characteristics for better scaling performance. We then compare the RankMixer model with other classic cross-structure designs such as DCN, RDCN, AutoInt, and DHEN, and find they suffer from an imbalance between model parameters and computational cost. Even with relatively small parameter sizes, these models already exhibit large FLOPs, indicating design shortcomings that constrain their results in the table. Meanwhile, although RankMixer achieves the best performance, its FLOPs remain relatively moderate among all the models when scaled to 100 million parameters, reflecting a balanced approach to model capacity and computational load.

We also compare RankMixer with several commonly used state-of-the-art scaling-up models—like Hiformer and wukong—and find that under similar parameter settings, RankMixer not only performs better but also has lower computational requirements.

4.3 Scaling Laws of different models

In the Figure 2, we show the scaling law curves observed from both parameter size and Flops. The RankMixer model shows the steepest scaling law both interms of parameters, and FLOPs. RankMixer is consistently superior to the other models.

Although the wukong model exhibits a relatively steep parameter curve, its computational cost increases even more rapidly; as a result, the gap between it and both RankMixer and hiformer is even larger on the AUC vs FLOPs curve. Moreover, hiformer's performance is slightly inferior to RankMixer's, reflecting that its reliance on token segmentation at a feature level and on Attention affects its efficiency. The scaling of DHEN is not ideal, reflecting the limited scalability of its cross structure. Moreover, MoE's strategy of scaling by adding experts brings about challenges in maintaining expert balance, which results in suboptimal scaling performance.

To be Sepcific, We can scale up RankMixer model by increasing width (D), feature token (T) and Layer(L). In our experiments we observed a conclusion shared with LLM scaling laws: model quality correlates primarily with the total number of parameters, and different scaling directions (depth L, width D, tokens T) yield almost identical performance. From a compute-efficiency standpoint, while larger hidden-dim generate larger matrix-multiply shapes and thus achieve higher MFU than stacking more layers. So the final configuration for 100M and 1B are set as (D=768, T=16, L=2) and (D=1536, T=32, L=2) respectively.

4.4 Ablation Study

Table 2: Ablation on components of RankMixer-100M

Setting	ΔAUC
w/o skip connections	−0.07%
w/o multi-head token mixing	−0.50%
w/o layer normalization	−0.05%
Per-token FFN → shared FFN	−0.31%

Table 3: Token2FFN Routing–strategy comparison

Routing strategy	ΔAUC	ΔParams	ΔFLOPs
All-Concat-MLP	−0.18%	0.0%	0.0%
All-Share	−0.25%	0.0%	0.0%
Self-Attention	−0.03%	+16%	+71.8%

In the RankMixer-100M model, we performed ablation studies on residual connections, Multi-Head Token-Mixing. From Table 2, we can see that removing these components significantly decreased the model's performance. Removing Multi-Head Token-Mixing loses global information, as each FFN only models partial features without interaction. Removing residual connections and LayerNorm also worsens performance, reducing training stability and making gradient explosion or vanishing issues more likely.

We further analyzed the token mixing strategies, i.e., the routing strategies from feature tokens to FFNs in the Table 3. The routing strategies compared with Multi-Head Token Mixing (Multi-Head Token-Mixing) include: *All-Concat-MLP:* Concatenates all tokens and processes them through a large MLP before splitting them into the same number of tokens. The decrease of performance shows the challenges in learning large matrices and weakening local information learning. *All-Share:* No splitting, the entire input vector is shared and fed to each per-token FFN similar as MoE. The performane declines siginifantly which show the importance of feature subspace split and independent modeling contrast to all-shared input. *Self-Attention:* Applies self-attention between tokens for routing. Its performance is slightly inferior to the Multi-Head Token-Mixing and also suffers from high computational cost, which shows the difficulty of learning similarity across hundreds of different feature subspaces.

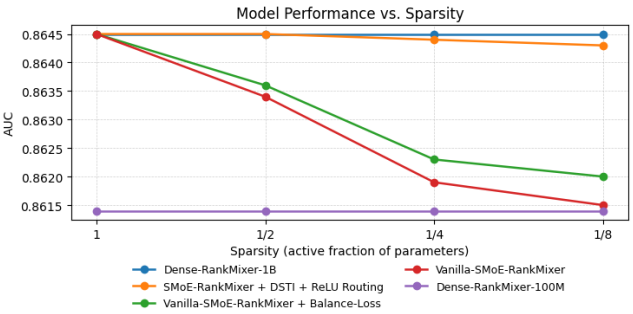


Figure 3: AUC performance of RankMixer variants under decreasingly sparse activation ratio (1,1/2,1/4,1/8 of experts): dense-training + ReLU-routed SMoE preserves almost all accuracy of the 1 B dense model.

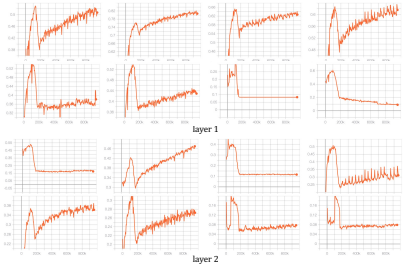


Figure 4: activated expert ratio for different token in RankMixer.

4.5 Sparse-MoE Scalability and Expert Balance

Scalability. Figure 3 plots offline AUC gains versus Sparsity of SMoE. Combining Dense-Training-Sparse-Inference with ReLU routing is essential for preserving accuracy under aggressive sparsity, enabling RankMixer to scale parameter capacity (and memory footprint) by > 8× with nearly no loss in AUC and with substantial inference-time savings (+50% improvement over throughput). Vanilla SMoE's performance drops monotonically as fewer experts are activated, illustrating the expert-imbalance and under-training issues we identified. Adding a load-balancing loss reduces the degradation relative to vanilla SMoE, yet still falls short of the DTSI + ReLU version because the problems mostly lies in the expert training instead of the router. This validates Sparse-MoE as the path to scale RankMixer from the current 1 B parameters to future 10 B-scale deployments without breaching the cost budgets.

Table 4: Online AB test result for Feed Recommendation Scenarios in both Douyin and Douyin lite app and the improvements are all statistically significant. According to long-term reverse A/B tests and continuous observations over long-term reverse experiments, the gains provided by RankMixer-1B have not yet converged, and the results presented here are still improving.

	Douyin app					Douyin lite				
	Active Day↑	**Duration↑**	**Like↑**	**Finish↑**	**Comment↑**	**Active Day↑**	**Duration↑**	**Like↑**	**Finish↑**	**Comment↑**
Overall	+0.2908%	+1.0836%	+2.3852%	+1.9874%	+0.7886%	+0.1968%	+0.9869%	+1.1318%	+2.0744%	+1.1338%
Low-active	+1.7412%	+3.6434%	+8.1641%	+4.5393%	+2.9368%	+0.5389%	+2.1831%	+4.4486%	+3.3958%	+0.9595%
Middle-active	+0.7081%	+1.5269%	+2.5823%	+2.5062%	+1.2266%	+0.4235%	+1.9011%	+1.7491%	+2.6568%	+0.6782%
High-active	+0.1445%	+0.6259%	+1.828%	+1.4939%	+0.4151%	+0.0929%	+1.1356%	+1.8212%	+1.7683%	+2.3683%

Table 5: Online lift of RankMixer in Advertising

	Advertising	
Metric	ΔAUC↑	ADVV↑
Lift	+0.73%	+3.90%

Table 6: Metrics of Online Model Deployment and Cost

Metric	OnlineBase-16M	RankMixer-1B	Change
#Param	15.8M	1.1B	↑ **70**×
FLOPs	107G	2106G	↑ 20.7×
Flops/Param(G/M)	6.8	1.9	↓ 3.6×
MFU	4.47%	**44.57**%	↑ 10×
Hardware FLOPs	fp32	fp16	↑ 2×
Latency	14.5ms	**14.3ms**	-

Expert balance and diversity. Vanilla Sparse MoE often suffers from expert imbalance, which in turn leaves some experts undertrained and eventually leads to "dying experts" (experts that are almost never activated) and only a few fixed experts are constantly activated. Figure 4 shows that combining DTSI (dense-training, sparse-inference) with ReLU routing effectively resolves this issue: Dense-training guarantees that most expert receives sufficient gradient updates, preventing expert starvation. ReLU routing makes the activation ratio dynamic across tokens—the activation proportion shows in the figure varies adaptively according to its information content, which aligns well with the diverse and highly dynamic distribution of recommendation data.

4.6 Online Serving cost

How can we prevent inference latency from exploding with a two-order-of-magnitude increase in parameters? In practical systems, latency is inversely proportional to throughput and directly proportional to the cost of serving machine resources. Compared to our previous fully-deployed 16M-parameter model (with a structure integrated with DLRM and DCN), our RankMixer model scaled parameters by approximately 70× to 1B. Despite this significant parameter increase, the final inference latency remained stable due to our hardware-aligned model design and optimization strategies.

When greatly increasing model parameters, latency can be decomposed into the following formula:

$$\text{Latency} = \frac{\#\text{Param} \times \text{FLOPs/Param ratio}}{\text{MFU} \times (\text{Theoretical Hardware FLOPs})}$$

As illustrated in Table 6, the two-order-of-magnitude parameter increase is progressively counteracted by **3.6**× decrease of FLOPs/Param ratio, **10** × increase of MFU and Quatization-based **2** × Hardware FlOPs improvements.

(1) **FLOPs/Param ratio.** The three row of Table 6 reports the number of floating-point operations (FLOPs) required *per parameter*. Thanks to the model design, RankMixer achieves a 70-fold increase in parameters with only around a 20-fold increase in FLOPs, achieving only one-third the FLOPs/Param ratio of the baseline—a **3.6,× efficiency gain**. In other words, RankMixer can accommodate three times more parameters than the baseline at the same FLOPs budget.
(2) **Model FLOPs Utilization (MFU).** Also shown in Table 6, MFU indicates the utilization of machine computing. By using large GEMM shapes, good parallel topology (fusing parallel per-token FFNs into one kernel), reduce the memory bandwith cost and overhead, RankMixer raises MFU by nearly **10** ×, shifting the model from an *Memory-bound* to a *Compute-bound* regime.
(3) **Quantization** Half-precision (fp16) inference will increase the theoretical peak Hardware FLOPS of GPUs by **2** ×.The primary computations in RankMixer consist of several large matrix multiplications that are well suited for half-precision (fp16) inference as mentioned before

4.7 Online Performance

To verify the universality of RankMixer as a scaling recommendation model framework, we conducted online experiments in the two core application scenarios of personalised ranking—*feed recommendation* and *advertising*, covering major use-cases of personalized ranking. For each scenario we monitor the following key performance indicators:

- **Feed–Recommendation.** *Active Days* is he average number of active days per user during the experiment period, a substitute for DAU growth.; *Duration* measures cumulative stay time on the App; *Finish/Like/Comment:* User's total complete plays, likes, and comments.
- **Advertising.** We report the model quality metrics (Δ*AUC*) and revenue metrics *ADVV* (Advertiser Value).

The Previous baselines in these scenarios are 16M parameter model combined with DLRM and DCN. We replaced the Dense part with the RankMixer-1B model improving a 0.7% AUC. We conduct online A/B test experiments across Feed Recommendation and Advertising. The long-term observation of A/B test result on

Feed Recommendation for 8 months are shown on the Table 4 [1] and the result of Advertising is shown on the Table 5.

RankMixer was deployed and evaluated in the three personalised-ranking application including the Feed Recommendation (RankMixer-1B) and Advertising (RankMixer-1B). RankMixer delivered statistically significant uplifts on all business-critical metrics. We can also observe that the improvements on low-active users are largest on the Table 4 compared to other activeness level user groups with over 1.7412% Active Days improvements, which demonstrates the model's strong generalization capability. These results demonstrate that the RankMixer as a unified backbone generalises reliably to different application scenarios.

5 Conclusion

In this article, we introduce our latest RankMixer model that has been fully deployed on Douyin Feed ranking. It combines the model designs for heterogeneous feature interactions and the highly parallelizable architecture for serving efficiency. Experiments have demonstrated its superior performance and steep scaling law. After its full deployment on the Douyin app, it has obtained 0.3% and 1% increase on active days and App duartion.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Newsha Ardalani, Carole-Jean Wu, Zeliang Chen, Bhargav Bhushanam, and Adnan Aziz. 2022. Understanding scaling laws for recommendation models. *arXiv preprint arXiv:2208.08489* (2022).

[3] Fedor Borisyuk, Mingzhou Zhou, Qingquan Song, Siyu Zhu, Birjodh Tiwana, Ganesh Parameswaran, Siddharth Dangi, Lars Hertel, Qiang Charles Xiao, Xiaochen Hou, Yunbo Ouyang, Aman Gupta, Sheallika Singh, Dan Liu, Hailing Cheng, Lei Le, Jonathan Hung, S. Sathiya Keerthi, Ruoyan Wang, Fengyu Zhang, Mohit Kothari, Chen Zhu, Daqi Sun, Yun Dai, Xun Luan, Sirou Zhu, Zhiwei Wang, Neil Daftary, Qianqi Shen, Chengming Jiang, Haichao Wei, Maneesh Varshney, Amol Ghoting, and Souvik Ghosh. 2024. LiRank: Industrial Large Scale Ranking Models at LinkedIn. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*. 4804–4815.

[4] Zheng Chai, Qin Ren, Xijun Xiao, Huizhi Yang, Bo Han, Sijun Zhang, Di Chen, Hui Lu, Wenlin Zhao, Lele Yu, Xionghang Xie, Shiru Ren, Xiang Sun, Yaocheng Tan, Peng Xu, Yuchao Zheng, and Di Wu. 2025. LONGER: Scaling Up Long Sequence Modeling in Industrial Recommenders. arXiv:2505.04421 [cs.IR] https://arxiv.org/abs/2505.04421

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[6] Sharad Chitlangia, Krishna Reddy Kesari, and Rajat Agarwal. 2023. Scaling generative pre-training for user ad activity sequences. (2023).

[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[8] Alexey Dosovitskiy. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[9] Yan Fang, Jingtao Zhan, Qingyao Ai, Jiaxin Mao, Weihang Su, Jia Chen, and Yiqun Liu. 2024. Scaling laws for dense retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1339–1349.

[10] Huan Gui, Ruoxi Wang, Ke Yin, Long Jin, Maciej Kula, Taibai Xu, Lichan Hong, and Ed H Chi. 2023. Hiformer: Heterogeneous Feature Interactions Learning with Transformers for Recommender Systems. *arXiv preprint arXiv:2311.05884* (2023).

[11] 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).

[12] Wei Guo, Hao Wang, Luankang Zhang, Jin Yao Chin, Zhongzhou Liu, Kai Cheng, Qiushi Pan, Yi Quan Lee, Wanqi Xue, Tingjia Shen, et al. 2024. Scaling New Frontiers: Insights into Large Recommendation Models. *arXiv preprint arXiv:2412.00714* (2024).

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

[14] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556* (2022).

[15] Yanhua Huang, Hangyu Wang, Yiyun Miao, Ruiwen Xu, Lei Zhang, and Weinan Zhang. 2022. Neural statistics for click-through rate prediction. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1849–1853.

[16] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[17] Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. 2019. Fi-gnn: Modeling feature interactions via graph neural networks for ctr prediction. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 539–548.

[18] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1754–1763.

[19] Jianghao Lin, Xinyi Dai, Yunjia Xi, Weiwen Liu, Bo Chen, Hao Zhang, Yong Liu, Chuhan Wu, Xiangyang Li, Chenxu Zhu, et al. 2023. How can recommender systems benefit from large language models: A survey. *ACM Transactions on Information Systems* (2023).

[20] Bin Liu, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang. 2019. Feature generation by convolutional neural network for click-through rate prediction. In *The World Wide Web Conference*. 1119–1129.

[21] Junwei Pan, Wei Xue, Ximei Wang, Haibin Yu, Xun Liu, Shijie Quan, Xueming Qiu, Dapeng Liu, Lei Xiao, and Jie Jiang. 2024. Ads recommendation in a collapsed and entangled world. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5566–5577.

[22] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-based neural networks for user response prediction over multi-field categorical data. *ACM Transactions on Information Systems (TOIS)* 37, 1 (2018), 1–35.

[23] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[24] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *International conference on machine learning*. Pmlr, 8821–8831.

[25] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural collaborative filtering vs. matrix factorization revisited. In *Proceedings of the 14th ACM Conference on Recommender Systems*. 240–248.

[26] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 255–262.

[27] Kyuyong Shin, Hanock Kwak, Su Young Kim, Max Nihlén Ramström, Jisu Jeong, Jung-Woo Ha, and Kyung-Min Kim. 2023. Scaling law for recommendation models: Towards general-purpose user representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 37. 4596–4604.

[28] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1161–1170.

[29] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295* (2024).

[30] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[31] Hangyu Wang, Jianghao Lin, Xiangyang Li, Bo Chen, Chenxu Zhu, Ruiming Tang, Weinan Zhang, and Yong Yu. 2024. FLIP: Fine-grained Alignment between ID-based Models and Pretrained Language Models for CTR Prediction. In *Proceedings of the 18th ACM Conference on Recommender Systems*. 94–104.

[1] This result was updated on July 24, 2025, since the long-term experiments indicating the gains had not yet saturated.

[32] 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.

[33] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*. 1785–1797.

[34] Yunli Wang, Zixuan Yang, Zhen Zhang, Zhiqiang Wang, Jian Yang, Shiyang Wen, Peng Jiang, and Kun Gai. 2024. Scaling Laws for Online Advertisement Retrieval. *arXiv preprint arXiv:2411.13322* (2024).

[35] Ziteng Wang, Jianfei Chen, and Jun Zhu. 2024. ReMoE: Fully Differentiable Mixture-of-Experts with ReLU Routing. *CoRR* abs/2412.14711 (2024). doi:10.48550/ARXIV.2412.14711 arXiv:2412.14711

[36] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, et al. [n. d.]. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. In *Forty-first International Conference on Machine Learning*.

[37] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. 2022. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 12104–12113.

[38] Buyun Zhang, Liang Luo, Yuxin Chen, Jade Nie, Xi Liu, Shen Li, Yanli Zhao, Yuchen Hao, Yantao Yao, Ellie Dingqiao Wen, et al. [n. d.]. Wukong: Towards a Scaling Law for Large-Scale Recommendation. In *Forty-first International Conference on Machine Learning*.

[39] Buyun Zhang, Liang Luo, Xi Liu, Jay Li, Zeliang Chen, Weilin Zhang, Xiaohan Wei, Yuchen Hao, Michael Tsang, Wenjun Wang, et al. 2022. DHEN: A deep and hierarchical ensemble network for large-scale click-through rate prediction. *arXiv preprint arXiv:2203.11014* (2022).

[40] Gaowei Zhang, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, and Ji-Rong Wen. 2024. Scaling law of large sequential recommendation models. In *Proceedings of the 18th ACM Conference on Recommender Systems*. 444–453.

[41] Weinan Zhang, Jiarui Qin, Wei Guo, Ruiming Tang, and Xiuqiang He. 2021. Deep learning for click-through rate estimation. *arXiv preprint arXiv:2104.10584* (2021).

[42] 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* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1059–1068. doi:10.1145/3219819.3219823