OneTrans: Unified Feature Interaction and Sequence Modeling with One Transformer in Industrial Recommender

Zhaoqi Zhang*
Nanyang Technological University
ByteDance
Singapore, Singapore
zhaoqi.zhang@bytedance.com

Haolei Pei*
ByteDance
Singapore, Singapore
haolei.pei@bytedance.com

Jun Guo*
ByteDance
Singapore, Singapore
jun.guo@bytedance.com

Tianyu Wang
ByteDance
Singapore, Singapore
tianyu.wang01@bytedance.com

Yufei Feng
ByteDance
Hangzhou, China
fengyihui@bytedance.com

Hui Sun
ByteDance
Hangzhou, China
sunhui.sunh@bytedance.com

Shaowei Liu†
ByteDance
Singapore, Singapore
liushaowei.nphard@bytedance.com

Aixin Sun†
Nanyang Technological University
Singapore, Singapore
axsun@ntu.edu.sg

Abstract

In recommendation systems, scaling up feature-interaction modules (e.g., Wukong, RankMixer) or user-behavior sequence modules (e.g., LONGER) has achieved notable success. However, these efforts typically proceed on separate tracks, which not only hinders bidirectional information exchange but also prevents unified optimization and scaling. In this paper, we propose OneTrans, a unified Transformer backbone that simultaneously performs user-behavior sequence modeling and feature interaction. OneTrans employs a unified tokenizer to convert both sequential and non-sequential attributes into a single token sequence. The stacked OneTrans blocks share parameters across similar sequential tokens while assigning token-specific parameters to non-sequential tokens. Through causal attention and cross-request KV caching, OneTrans enables precomputation and caching of intermediate representations, significantly reducing computational costs during both training and inference. Experimental results on industrial-scale datasets demonstrate that OneTrans scales efficiently with increasing parameters, consistently outperforms strong baselines, and yields a 5.68% lift in per-user GMV in online A/B tests.

CCS Concepts

• **Information systems → Recommender systems**.

*These authors contributed equally.
†Corresponding author.

Keywords

Recommender System, Ranking Model, Scaling Laws

ACM Reference Format:
Zhaoqi Zhang, Haolei Pei, Jun Guo, Tianyu Wang, Yufei Feng, Hui Sun, Shaowei Liu, and Aixin Sun. 2025. OneTrans: Unified Feature Interaction and Sequence Modeling with One Transformer in Industrial Recommender. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX).* ACM, New York, NY, USA, 9 pages.

1 Introduction

Recommendation systems (RecSys) play a fundamental role in various information services, such as e-commerce [9, 31], streaming media [2, 19, 26] and social networks [28]. Industrial RecSys generally adopt a cascaded ranking architecture [6, 16, 21]. First, a recall stage selects hundreds of candidates from billion-scale corpora [13, 32]. Then, a ranking stage — often with coarse- and fine-ranking — scores each candidate and returns the top-k items [11, 25, 26, 28, 33].

We focus on the ranking stage in this paper. For ranking, mainstream approaches iterate on two separate modules: (a) *sequence modeling*, which encodes user multi-behavior sequences into candidate-aware representations using local attention or Transformer encoders [1, 14, 23, 31], and (b) *feature interaction*, which learns high-order crosses among non-sequential features (e.g., user profile, item profile, and context) via factorization or explicit cross networks, or attention over feature groups [11, 12, 25, 33]. As shown in Fig. 1(a), these approaches typically encode user behaviors into a *compressed* sequence representation, then concatenate it with non-sequential features and apply a feature-interaction module to learn higher-order interaction; we refer to this design as the *encode-then-interaction* pipeline.

The success of large language models (LLMs) demonstrates that scaling model size (e.g., parameter size, training data) yields predictable gains in performance [15], inspiring similar investigations within RecSys [1, 28, 33]. For feature interaction, Wukong [28] stacks Factorization Machine blocks with linear compression to

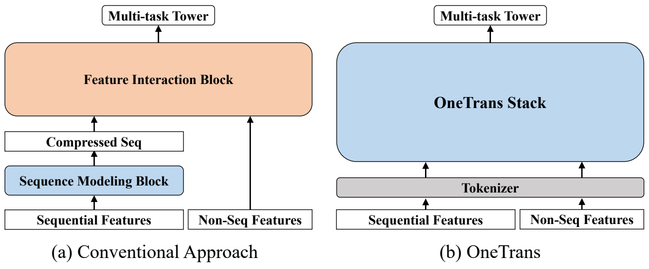


Figure 1: Architectural comparison. (a) Conventional *encode-then-interaction* pipeline encodes sequential features and merges non-sequential features before a post-hoc feature interaction block. (b) OneTrans performs joint modeling of both sequential and non-sequential features within a single OneTrans (Transformer-style) stack.

capture high-order feature interactions and establishes scaling laws, while RankMixer [33] achieves favorable scaling through hardware-friendly token-mixing with token-specific feed-forward networks (FFNs). For sequence modeling, LONGER[1] applies causal Transformers to long user histories and shows that scaling depth and width yields monotonic improvements. Although effective in practice, separating sequence modeling and feature interaction as independent modules introduces two major limitations. First, the encode-then-interaction pipeline restricts bidirectional information flow, limiting how static/context features shape sequence representations [27]. Second, module separation fragments execution and increases latency, whereas a single Transformer-style backbone can reuse LLM optimizations e.g., KV caching, memory-efficient attention, and mixed precision, for more effective scaling [11].

In this paper, we propose **OneTrans**, an innovative architectural paradigm with a unified Transformer backbone that jointly performs user-behavior sequence modeling and feature interaction. As shown in Fig. 1(b), OneTrans enables bidirectional information exchange within the unified backbone. It employs a unified tokenizer that converts both *sequential* features (diverse behavior sequences) and *non-sequential* features (static user/item and contextual features) into a single token sequence, which is then processed by a pyramid of stacked OneTrans blocks, a Transformer variant tailored for industrial RecSys. To accommodate the diverse token sources in RecSys, unlike the text-only tokens in LLMs, each OneTrans block adopts a *mixed* parameterization similar to HiFormer [11]. Specifically, all *sequential* tokens (from sequential features) share a single set of Q/K/V and FFN weights, while each *non-sequential* token (from non-sequential features) receives *token-specific* parameters to preserve its distinct semantics.

Unlike conventional encode-then-interaction frameworks, OneTrans eliminates the architectural barrier between *sequential* and *non-sequential* features through a unified causal Transformer backbone. This formulation brings RecSys scaling in line with LLM practices: the *entire* model can be scaled by adjusting backbone depth and width, while seamlessly inheriting mature LLM optimizations, such as FlashAttention [7], and mixed precision training [17]. Particularly, cross-candidate and cross-request KV caching [1] reduces the time complexity from $O(C)$ to $O(1)$ for sessions with C candidates, making large-scale OneTrans deployment feasible.

In summary, our main contributions are fourfold: **(1) Unified framework.** We present OneTrans, a single Transformer backbone for ranking, equipped with a *unified tokenizer* that encodes sequential and non-sequential features into one token sequence, and a *unified Transformer block* that jointly performs sequence modeling and feature interaction. **(2) Customization for recommenders.** To bridge the gap between LLMs and RecSys tasks, OneTrans introduces a *mixed parameterization* that allocates token-specific parameters to diverse non-sequential tokens while sharing parameters for all sequential tokens. **(3) Efficient training and serving.** We improve efficiency with a *pyramid strategy* that progressively prunes sequential tokens and a *cross-request KV Caching* that reuses user-side computations across candidates. In addition, we adopt LLM optimizations such as FlashAttention, mixed-precision training, and half-precision inference to further reduce memory and compute. **(4) Scaling and deployment.** OneTrans demonstrates near log-linear performance gains with increased model size, providing evidence of a scaling law in real production data. When deployed online, it achieves statistically significant lifts on business KPIs while maintaining production-grade latency.

2 Related Work

Early RecSys like DIN [31] and its session-aware variants (DSIN) [9] use local attention to learn candidate-conditioned summaries of user histories, but compress behaviors into fixed-length vectors per candidate, limiting long-range dependency modeling [30]. Self-attentive methods like SASRec [14], BERT4Rec [23], and BST [4] eliminate this bottleneck by letting each position attend over the full history and improve sample efficiency with bidirectional masking. Recently, as scaling laws [15] in RecSys are increasingly explored, LONGER [1] pushes sequence modeling toward industrial scales by targeting ultra-long behavioral histories with efficient attention and serving-friendly designs. However, in mainstream pipelines these sequence encoders typically remain *separate* from the feature-interaction stack, leading to late fusion rather than joint optimization with static contextual features [27].

On the feature-interaction side, early RecSys rely on manually engineered cross-features or automatic multiplicative interaction layers. Classical models such as Wide&Deep [5], FM/DeepFM [3, 12], and DCN/DCNv2 [24, 25] provide efficient low-order or bounded-degree interactions. However, as recent scaling studies observe [28], once the model stacks enough cross layers, adding more stops helping: model quality plateaus instead of continuing to improve. To overcome the rigidity of preset cross forms, attention-based approaches automatically learn high-order interactions. AutoInt [22] learns arbitrary-order relations, and HiFormer [11] introduces group-specific projections to better capture heterogeneous, asymmetric interactions. With scaling up increasingly applied to *feature-interaction* modules, large-scale systems such as Wukong [28] demonstrate predictable gains by stacking FM-style interaction blocks with linear compression, while RankMixer [33] achieves favorable scaling via parallel token mixing and sparse MoE under strict latency budgets. However, these interaction modules typically adhere to the

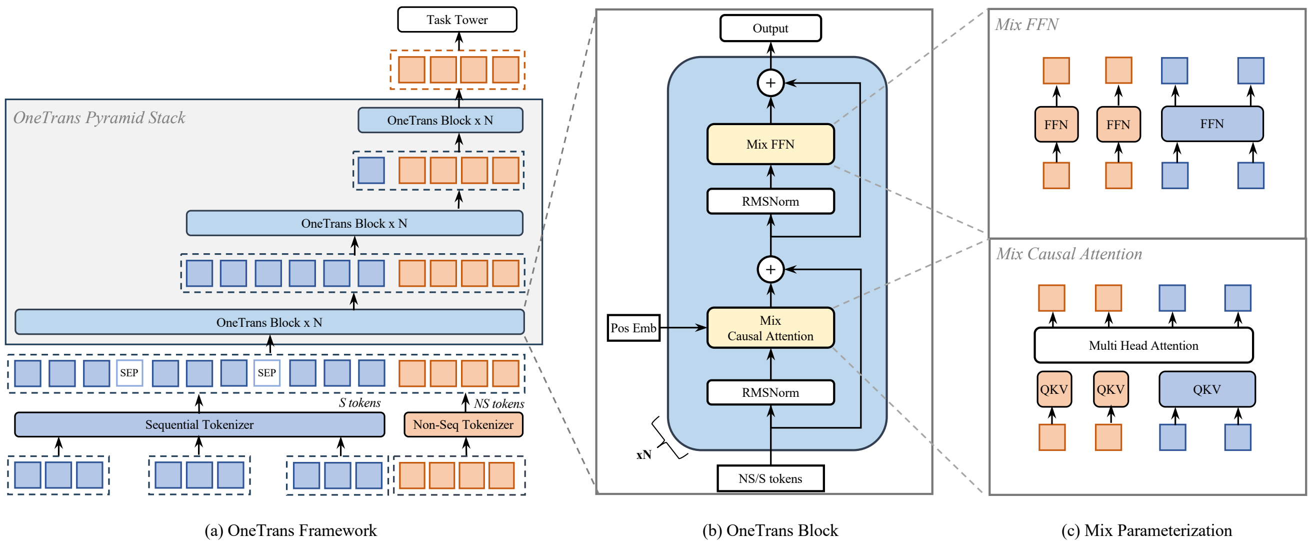


Figure 2: System Architecture. (a) OneTrans overview. Sequential (S, blue) and non-sequential (NS, orange) features are tokenized separately. After inserting [SEP] between user behavior sequences, the unified token sequence is fed into stacked *OneTrans Pyramid Blocks* that progressively shrink the token length until it matches the number of NS tokens. (b) OneTrans Block: a causal pre-norm Transformer Block with RMSNorm, *Mixed Causal Attention* and *Mixed FFN*. (c) "Mixed" = mixed parameterization: S tokens share one set of QKV/FFN weights, while each NS token receives its own token-specific QKV/FFN.

interaction paradigm, which pushes interactions to a separate stage and blocks unified optimization with user sequence modeling [27].

To date, progress in RecSys has largely advanced along two independent tracks: sequence modeling and feature interaction. InterFormer [27] attempts to bridge this gap through a summary-based bidirectional cross architecture that enables mutual signal exchange between the two components. However, it still maintains them as separate modules, and the cross architecture introduces both architectural complexity and fragmented execution. Without a unified backbone for joint modeling and optimization, scaling the system as an integrated whole remains challenging.

3 Methodology

Before detailing our method, we briefly describe the task setting. In a cascaded industrial RecSys, each time the recall stage returns a candidate set (typically hundreds of candidate items) for a user u. The ranking model then predicts a score to each candidate item i:

$$\hat{y}_{u,i} = f\left(i \,\middle|\, \mathcal{NS}, \mathcal{S}; \Theta\right) \tag{1}$$

where $\mathcal{NS}$ is a set of non-sequential features derived from the user, the candidate item, and the context; $\mathcal{S}$ is a set of historical behavior sequences from the user; and Θ are trainable parameters. Common task predictions include the click-through rate (CTR) and the post-click conversion rate (CVR).

$$\begin{aligned} \mathrm{CTR}_{u,i} &= P\left(\text{click} = 1 \,\middle|\, \mathcal{NS}, \mathcal{S}; \Theta\right), \\ \mathrm{CVR}_{u,i} &= P\left(\text{conv} = 1 \,\middle|\, \text{click} = 1, \mathcal{NS}, \mathcal{S}; \Theta\right). \end{aligned} \tag{2}$$

3.1 OneTrans Framework Overview

As illustrated in Fig. 2(a), OneTrans employs a *unified tokenizer* that maps sequential features $\mathcal{S}$ to S-tokens, and non-sequential features $\mathcal{NS}$ to NS-tokens. A *pyramid-stacked Transformer* then consumes the unified token sequence jointly within a single computation graph. We denote the initial token sequence as

$$\mathbf{X}^{(0)} = \left[\text{S-tokens}\,;\,\text{NS-tokens}\right] \in \mathbb{R}^{(L_{\mathrm{S}}+L_{\mathrm{NS}})\times d}. \tag{3}$$

This token sequence is constructed by concatenating L_{S} number of S-tokens and L_{NS} number of NS-tokens, with all tokens having dimensionality d. Note that, the S-tokens contain learnable [SEP] tokens inserted to delimit boundaries between different kind of user-behavior sequences. As shown in Fig. 2(b), each OneTrans block progressively refines the token states through:

$$\mathbf{Z}^{(n)} = \mathrm{MixedMHA}\Big(\mathrm{Norm}\big(\mathbf{X}^{(n-1)}\big)\Big) + \mathbf{X}^{(n-1)}, \tag{4}$$

$$\mathbf{X}^{(n)} = \mathrm{MixedFFN}\Big(\mathrm{Norm}\big(\mathbf{Z}^{(n)}\big)\Big) + \mathbf{Z}^{(n)}. \tag{5}$$

Here, MixedMHA (Mixed Multi-Head Attention) and MixedFFN (Mixed Feed-Forward Network) adopt a mixed parameterization strategy (see Fig. 2(c)) sharing weights across sequential tokens, while assigning separate parameters to non-sequential tokens in both the attention and feed-forward layers.

A unified causal mask enforces autoregressive constraints, restricting each position to attend only to preceding tokens. Specifically, NS-tokens are permitted to attend over the entire history of S-tokens, thereby enabling comprehensive cross-token interaction. By stacking such blocks with pyramid-style tail truncation applied to S-tokens, the model progressively distills compact high-order information into the NS-tokens. The final token states are then passed to task-specific heads for prediction.

By unifying non-sequential and sequential features into a unified token sequence and modeling them with a causal Transformer, OneTrans departs from the conventional *encode-then-interaction* pipeline. This unified design naturally enables (i) *intra-sequence* interactions within each behavior sequence, (ii) *cross-sequence* interactions across multiple sequences, (iii) *multi-source feature* interactions among item, user, and contextual features, and (iv) *sequence-feature* interactions, *all within a single Transformer stack.*

The unified formulation enables us to seamlessly inherit mature LLM engineering optimizations, including KV caching and memory-efficient attention, thereby substantially reducing inference latency. We argue this unified formulation is well suited to tackling multi-sequence and cross-domain recommendation challenges in a single, and scalable architecture. Next, we detail the design.

3.2 Features and Tokenization

To construct the initial token sequence $\mathbf{X}^{(0)}$, OneTrans first applies a feature preprocessing pipeline that maps all raw feature inputs into embedding vectors. These embeddings are then partitioned into (i) a multi-behavior *sequential* subset and (ii) a *non-sequential* subset representing user, item, or context features. Separate tokenizers are applied to each subset.

3.2.1 *Non-Sequential Tokenization.*

Non-sequential features $\mathcal{NS}$ include both numerical inputs (e.g., price, CTR) and categorical inputs (e.g., user ID, item category). All features are either bucketized or one-hot encoded and then embedded. Since industrial systems typically involve hundreds of features with varying importance, there are two options for controlling the number of non-sequential tokens, denoted by L_{NS}:

Group-wise Tokenizer (aligned with RankMixer [33]). Features are manually partitioned into semantic groups $\{\mathbf{g}_1, \ldots, \mathbf{g}_{L_{NS}}\}$. Each group is concatenated and passed through a group-specific MLP:

$$\text{NS-tokens} = \left[\, \text{MLP}_1(\text{concat}(\mathbf{g}_1)), \ldots, \text{MLP}_{L_{NS}}(\text{concat}(\mathbf{g}_{L_{NS}}))\,\right]. \tag{6}$$

Auto-Split Tokenizer. Alternatively, all features are concatenated and projected once by a single MLP, then split:

$$\text{NS-tokens} = \text{split}\Big(\text{MLP}(\text{concat}(\mathcal{NS})),\ L_{NS}\Big). \tag{7}$$

Auto-Split Tokenizer reduces kernel launch overhead compared with Group-wise approach, by using a single dense projection. We will evaluate both choices through experiments.

Ultimately, non-sequential tokenization yields L_{NS} number of non-sequential tokens, each of dimensionality d.

3.2.2 *Sequential Tokenization.*

OneTrans accepts multi-behavior sequences as

$$\mathcal{S} = \{\mathrm{S}_1, \ldots, \mathrm{S}_n\}, \quad \mathrm{S}_i = \left[\mathbf{e}_{i1}, \ldots, \mathbf{e}_{iL_i}\right]. \tag{8}$$

Each sequence S_i consists of L_i number of event embeddings $\mathbf{e}$, which is constructed by concatenating the item ID with its corresponding side information like item category and price.

Multi-behavior sequences can vary in their raw dimensionality. Hence, for each sequence S_i, we use one shared projection MLP_i to convert its all event $\mathbf{e}_{ij}$ as a common dimensionality d:

$$\tilde{\mathrm{S}}_i = \left[\, \text{MLP}_i(\mathbf{e}_{i1}), \ldots, \text{MLP}_i(\mathbf{e}_{iL_i})\,\right] \in \mathbb{R}^{L_i \times d}. \tag{9}$$

Aligned sequences $\tilde{\mathrm{S}}_i$ are merged into a single token sequence by one of two rules: 1) *Timestamp-aware*: interleave all events by time, with sequence-type indicators; 2) *Timestamp-agnostic*: concatenate sequences by event impact, e.g., purchase $\rightarrow$ add-to-cart $\rightarrow$ click, inserting learnable `[SEP]` tokens between sequences. In the latter, behaviors with higher user intent are placed earlier in the sequence. Ablation results indicate that, when timestamps are available, the timestamp-aware rule outperforms the impact-ordered alternative. Formally, we have:

$$\text{S-Tokens} = \text{Merge}\big(\tilde{\mathrm{S}}_1, \ldots, \tilde{\mathrm{S}}_n\big) \in \mathbb{R}^{L_S \times d}, \quad L_S = \sum_{i=1}^{n} L_i + L_{\text{SEP}}. \tag{10}$$

3.3 OneTrans Block

As shown in Fig. 2(b), each OneTrans block is a pre-norm causal Transformer applied to a *normalized* token sequence: L_S *sequential* S-tokens, followed by L_{NS} *non-sequential* NS-tokens. Inspired by the findings on heterogeneous feature groups [11], we make a lightweight modification to Transformer to allow a mixed parameter scheme, see Fig. 2(c). Specifically, homogeneous S-tokens share one set of parameters. The NS-tokens, being heterogeneous across sources/semantics, receive token-specific parameters.

Unlike LLM inputs, the token sequence in RecSys combines sequential S-tokens with diverse NS-tokens whose value ranges and statistics differ substantially. Post-norm setups can cause attention collapse and training instability due to these discrepancies. To prevent this, we apply RMSNorm [29] as pre-norm to *all* tokens, aligning scales across token types and stabilizing optimization.

3.3.1 *Mixed (shared/token-specific) Causal Attention.*

OneTrans adopts a standard multi-head attention (MHA) with a causal attention mask; the only change is how Q/K/V are parameterized. Let $\mathbf{x}_i \in \mathbb{R}^d$ be the i-th token. To compute Q/K/V, we use a *shared* projection for S-tokens ($i \le L_S$) and L_{NS} *token-specific* projections for NS-tokens ($i > L_S$):

$$(\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i) \;=\; \big(\mathbf{W}_i^{Q}\mathbf{x}_i,\ \mathbf{W}_i^{K}\mathbf{x}_i,\ \mathbf{W}_i^{V}\mathbf{x}_i\big), \tag{11}$$

where $\mathbf{W}_i^{\Psi}$ ($\Psi \in \{Q, K, V\}$) follows a mixed parameterization scheme:

$$\mathbf{W}_i^{\Psi} \;=\; \begin{cases} \mathbf{W}_{\mathrm{S}}^{\Psi}, & i \le L_S \quad \text{(shared for S-tokens)}, \\ \mathbf{W}_{\mathrm{NS},i}^{\Psi}, & i > L_S \quad \text{(token-specific for NS-tokens)}. \end{cases} \tag{12}$$

Attention uses a standard *causal* mask, with NS-tokens placed *after* S-tokens. This induces: (1) **S-side.** Each S-token attends only to earlier S positions. For *timestamp-aware* sequences, every event conditions on its history; for *timestamp-agnostic* sequences (ordered by intent, e.g., purchase $\rightarrow$ add-to-cart $\rightarrow$ click/impression), causal masking lets high-intent signals inform and filter later low-intent behaviors. (2) **NS-side.** Every NS-token attends to the *entire* S history, effectively a target-attention aggregation of sequence evidence, and to *preceding* NS-tokens, increasing token-level interaction diversity. (3) **Pyramid support.** On both S and NS sides, causal masking progressively concentrates information toward later positions, naturally supporting the pyramid schedule that prunes tokens layer by layer, to be detailed shortly.

3.3.2 *Mixed (shared/token-specific) FFN.*

Similarly, the feed-forward network follows the same parameterization strategy: token-specific

FFNs for NS-tokens, and a shared FFN for S-tokens,

$$\text{MixedFFN}(\mathbf{x}_i) = \mathbf{W}_i^2\,\phi(\mathbf{W}_i^1\mathbf{x}_i). \tag{13}$$

Here $\mathbf{W}_i^1$ and $\mathbf{W}_i^2$ follow the mixed parameterization of Eqn. (12), i.e., shared for $i \le L_S$ and token-specific for $i > L_S$.

In summary, relative to a standard causal Transformer, OneTrans changes only the *parameterization*: NS-tokens use *token-specific* QKV and FFN; S-tokens *share* a single set of parameters. A single causal mask ties the sequence together, allowing NS-tokens to aggregate the entire behavior history while preserving efficient, Transformer-style computation.

3.4 Pyramid Stack

As noted in Section 3.3, causal masking concentrates information toward later positions. Exploiting this recency structure, we adopt a *pyramid* schedule: at each OneTrans block layer, only a subset of the most recent S-tokens issue queries, while keys/values are still computed over the full sequence; the query set shrinks with depth.

Let $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^{L}$ be the input token list and $\mathcal{Q} = \{L-L'+1, \ldots, L\}$ denote a tail index set with $L' \le L$. Following Eqn. 12, we modify *queries* as $i \in \mathcal{Q}$:

$$\mathbf{q}_i \;=\; \mathbf{W}_i^{Q}\,\mathbf{x}_i, \qquad i \in \mathcal{Q}, \tag{14}$$

while keys and values are computed as usual over the full sequence $\{1, \ldots, L\}$. After attention, only outputs for $i \in \mathcal{Q}$ are retained, reducing the token length to L' and forming a pyramidal hierarchy across layers.

This design yields two benefits: (i) *Progressive distillation*: long behavioral histories are funneled into a small tail of queries, focusing capacity on the most informative events and consolidating information into the NS-tokens; and (ii) *Compute efficiency*: attention cost becomes $O(LL'd)$ and FFN scales linearly with L'. Shrinking the query set directly reduces FLOPs and activation memory.

3.5 Training and Deployment Optimization

3.5.1 Cross Request KV Caching. In industrial RecSys, samples from the same request are processed contiguously both during training and serving: their S-tokens remain identical across candidates, while NS-tokens vary per candidate item. Leveraging this structure, we integrate the widely adopted KV Caching [1] into OneTrans, yielding a unified two-stage paradigm.

Stage I (S-side, once per request). Process all S-tokens with causal masking and cache their key/value pairs and attention outputs. This stage executes *once* per request.

Stage II (NS-side, per candidate). For each candidate, compute its NS-tokens and perform cross-attention against the cached S-side keys/values, followed by token-specific FFN layers. Specially, candidate-specific sequences (e.g., SIM [20]) are pre-aggregated into NS-tokens via pooling, as they cannot reuse the shared S-side cache.

The KV Caching amortizes S-side computation across candidates, keeping per-candidate work lightweight and eliminating redundant computations for substantial throughput gains.

Since user behavioral sequences are append-only, we extend KV Caching *across requests*: each new request reuses the previous cache and computes only the incremental keys/values for newly added behaviors. This reduces per-request sequence computation from $O(L)$ to $O(\Delta L)$, where ΔL is the number of new behaviors since the last request.

Table 1: Dataset overview for OneTrans experiments.

Metric	Value
# Impressions (samples)	29.1B
# Users (unique)	27.9M
# Items (unique)	10.2M
Daily impressions (mean ± std)	118.2M ± 14.3M
Daily active users (mean ± std)	2.3M ± 0.3M

3.5.2 Unified LLM Optimizations. We employ FlashAttention-2 [8] to reduce attention I/O and the quadratic activation footprint of vanilla attention via tiling and kernel fusion, yielding lower memory usage and higher throughput in both training and inference. To further ease memory pressure, we use mixed-precision training (BF16/FP16) [18] together with activation recomputation [10], which discards selected forward activations and recomputes them during backpropagation. This combination trades modest extra compute for substantial memory savings, enabling larger batches and deeper models without architectural changes.

4 Experiments

Through both offline evaluations and online tests, we aim to answer the following Research Questions (RQs): **RQ1: Unified stack vs. encode–then–interaction.** Does the *single Transformer stack* yield consistent performance gains under the comparable compute? **RQ2: Which design choices matter?** We conduct ablations on the *input layer* (e.g., tokenizer, sequence fusion) and the *OneTrans block* (e.g., parameter sharing, attention type, pyramid stacking) to evaluate the importance of different design choices for performance and efficiency. **RQ3: Systems efficiency.** Do pyramid stacking, cross-request KV Caching, FlashAttention-2, and mixed precision with recomputation reduce FLOPs/memory and latency under the same OneTrans graph? **RQ4: Scaling law.** As we scale length (token sequence length), width (d_{model}), depth (number of layers), do loss/performance exhibit the expected *log-linear* trend? **RQ5: Online A/B Tests.** Does deploying OneTrans online yield statistically significant lifts in key business metrics (e.g., order/u, GMV/u) under production latency constraints?

4.1 Experimental Setup

4.1.1 Dataset. For offline evaluation, we evaluate OneTrans in a large-scale industrial ranking scenario using production logs under strict privacy compliance (all personally identifiable information is anonymized and hashed). Data are split chronologically, with all features snapshotted at impression time to prevent temporal leakage and ensure online-offline consistency. Labels (e.g., clicks and orders) are aggregated within fixed windows aligned with production settings. Table 1 summarizes the dataset statistics.

4.1.2 Tasks and Metrics. We evaluate two binary ranking tasks as defined in Eqn. (2): CTR and CVR. Performance is measured by AUC and UAUC (impression-weighted user-level AUC).

Table 2: Offline effectiveness (CTR/CVR) and efficiency; higher AUC/UAUC is better. Efficiency figures are collected from prior works, and dashes indicate unavailable. * indicates models deployed in our production lineage in chronological order: *DCNv2+DIN → RankMixer+DIN → RankMixer+Transformer → OneTrans$_S$ → OneTrans$_L$ (default)*

Type	Model	CTR		CVR (order)		Efficiency	
		AUC ↑	UAUC ↑	AUC ↑	UAUC ↑	Params (M)	TFLOPs
(1) Base model	DCNv2 + DIN (base)*	0.79623	0.71927	0.90361	0.71955	10	0.06
(2) Feature-interaction	Wukong + DIN	+0.08%	+0.11%	+0.14%	+0.11%	28	0.54
	HiFormer + DIN	+0.11%	+0.18%	+0.23%	-0.20%	108	1.35
	RankMixer + DIN*	+0.27%	+0.36%	+0.43%	+0.19%	107	1.31
(3) Sequence-modeling	RankMixer + StackDIN	+0.40%	+0.37%	+0.63%	-1.28%	108	1.43
	RankMixer + LONGER	+0.49%	+0.59%	+0.47%	+0.44%	109	1.87
	RankMixer + Transformer*	+0.57%	+0.90%	+0.52%	+0.75%	109	2.51
(4) Unified framework	OneTrans$_S$*	+1.13%	+1.77%	+0.90%	+1.66%	91	2.64
	OneTrans$_L$ (default)*	+1.53%	+2.79%	+1.14%	+3.23%	330	8.62

Next-batch evaluation. Data are processed chronologically. For each mini-batch, we (i) log predictions in eval mode, then (ii) train on the same batch. AUC and UAUC are computed daily from each day's predictions and finally macro-averaged across days.

Efficiency metrics. We report *Params* (model parameters excluding sparse embeddings) and *TFLOPs* (training compute in TFLOPs at batch size 2048).

4.1.3 Baselines. We construct industry-standard model combinations as baselines using the same features and matched compute budgets. Under the *encode-then-interaction* paradigm, we start from the widely-used production baseline **DCNv2+DIN** [25, 31] and progressively strengthen the **feature-interaction** module: DCNv2 → Wukong [28] → HiFormer [11] → RankMixer [33]. With RankMixer fixed, we then vary the **sequence-modeling** module: StackDIN → Transformer [4] → LONGER [1].

4.1.4 Hyperparameter Settings. We report two settings: **OneTrans-S** uses *6* stacked OneTrans blocks width d=256, and H=4 heads, targeting ≈ 100M parameters. **OneTrans-L** scales to *8* layers with width d=384 (still H=4). Inputs are processed through a unified tokenizer: multi-behavior sequences are fused in a *timestamp-aware* manner, while non-sequential features are tokenized via *Auto-Split*. The pyramid schedule linearly reduces tokens from 1190 to 12.

Optimization and infrastructure. We use a dual-optimizer strategy without weight decay: sparse embeddings are optimized with Adagrad (β_1=0.1, β_2=1.0), and dense parameters with RMSPropV2 (lr=0.005, momentum=0.99999). The per-GPU batch size is set to 2048 during training, with gradient clipping thresholds of 90 for dense layers and 120 for sparse layers to ensure stable optimization. For online inference, we adopt a smaller batch size of 100 per GPU to balance throughput and latency. Training uses data-parallel all-reduce on 16 H100 GPUs.

4.2 RQ1: Performance Evaluation

We anchor our comparison on DCNv2+DIN, the pre-scaling production baseline in our scenario (Table 2). Under the *encode-then-interaction* paradigm, scaling either component independently is beneficial: upgrading the *feature interaction* module (DCNv2 → Wukong → HiFormer → RankMixer) or the *sequence modeling* module (StackDIN → Transformer → LONGER) yields consistent gains in CTR AUC/UAUC and CVR AUC. In our system, improvements above +0.1% in these metrics are considered meaningful, while gains above +0.3% typically correspond to statistically significant effects in online A/B tests. However, CVR UAUC is treated cautiously due to smaller per-user sample sizes and higher volatility.

Moving to a unified design, OneTrans$_S$ surpasses the baseline by +1.13%/+1.77% (CTR AUC/UAUC) and +0.90%/+1.66% (CVR AUC/UAUC). At a comparable parameter scale, it also outperforms RankMixer+Transformer with similar training FLOPs (2.64T vs. 2.51T), demonstrating the benefits of unified modeling. Scaling further, OneTrans$_L$ delivers the best overall improvement of +1.53% /+2.79% (CTR AUC/UAUC) and +1.14%/+3.23% (CVR AUC/UAUC), showing a predictable quality performance as model capacity grows.

In summary, unifying sequence modeling and feature interaction in a single Transformer yields more reliable and compute-efficient improvements than scaling either component independently.

4.3 RQ2: Design Choices via Ablation Study

We perform an ablation study of the proposed **OneTrans$_S$** model to quantify the contribution of key design choices. The complete results are summarized in Table 3. We evaluate the following variants: **Input variants:** i) Replacing the *Auto-Split Tokenizer* with a *Group-wise Tokenizer* (Row 1); ii) Using a timestamp-*agnostic* fusion strategy instead of the timestamp-*aware* sequence fusion (Row 2); iii) Removing `[SEP]` tokens in the timestamp-*aware* sequence fusion (Row 3); **OneTrans block variants:** i) Sharing a single set of Q/K/V and FFN parameters across *all* tokens, instead of assigning separate parameters to NS-tokens (Row 4); ii) Replacing causal attention with full attention (Row 5); iii) Disabling the pyramid stack by keeping the full token sequence at *all* layers (Row 6).

In summary, the ablations show that 1) **Auto-Split Tokenizer** provides a clear advantage over manually grouping non-sequential features into tokens, indicating that allowing the model to automatically build non-sequential tokens is more effective than relying on

Table 3: Impact of the choices of input design and OneTrans block design, using the OneTrans$_S$ model as the reference.

Type	Variant	CTR AUC↑	CTR UAUC↑	CVR (order) AUC↑	CVR (order) UAUC↑	Efficiency Params (M)	Efficiency TFLOPs
Input	Group-wise Tokenzier	-0.10%	-0.30%	-0.12%	-0.10%	78	2.35
	Timestamp-agnostic Fusion	-0.09%	-0.22%	-0.20%	-0.21%	91	2.64
	Timestamp-agnostic Fusion w/o Sep Tokens	-0.13%	-0.32%	-0.29%	-0.33%	91	2.62
OneTrans Block	Shared parameters	-0.15%	-0.29%	-0.14%	-0.29%	24	2.64
	Full attention	+0.00%	+0.01%	-0.03%	+0.06%	91	2.64
	w/o pyramid stack	-0.05%	+0.06%	-0.04%	-0.42%	92	8.08

Table 4: Key efficiency comparison between OneTrans$_L$ and the DCNv2+DIN baseline.

Metric	**DCNv2+DIN**	**OneTrans$_L$**
TFLOPs	0.06	8.62
Params (M)	10	330
MFU	13.4	30.8
Inference Latency (p99, ms)	13.6	13.2
Training Memory (GB)	20	32
Inference Memory (GB)	1.8	0.8

human-defined feature grouping; 2) **Timestamp-aware fusion** beats intent-based ordering when timestamps exist, suggesting that temporal ordering should be prioritized over event impact; 3) Under timestamp-*agnostic* fusion, learnable **[SEP] tokens** help the model separate sequences; 4) Assigning **token-specific parameters** to NS-tokens yields clear gains over sharing one set across all tokens, demonstrating that modeling non-sequential features with individualized projections enables better feature discrimination; 5) **Causal and full attention** achieve similar results, indicating that allowing tokens to attend to future positions is not crucial in this setting. Notably, we emphasize that full attention prohibits the use of standard optimizations such as KV caching; 6) Retaining the full token list at every layer provides no benefit: OneTrans effectively summarizes information into a small tail of tokens, so the **pyramid design** can safely prune queries to save computation.

4.4 RQ3: Systems Efficiency

To quantify the optimizations in Section 3.5, we ablate them on an unoptimized **OneTrans$_S$** baseline and report training/inference metrics in Table 5. The unoptimized OneTrans$_S$ runs at 407 ms training runtime with 53.13 GB peak training memory, and 54.00 ms p99 inference latency with 1.70 GB inference memory, where *p99* denotes the 99th-percentile (tail) latency—a standard SLO metric for high-availability online services. These differences reflect distinct operating conditions: offline training uses large per-device batches, while online inference distributes micro-batches across machines for stability. As shown in the table, 1) **Pyramid stack** yields substantial savings (−28.7% training time, −42.6% training memory, −8.4% inference latency, −6.9% inference memory) by compressing long behavioral histories into compact query sets; 2) **Cross-request KV caching** eliminates redundant sequence-side computation, reducing runtime/latency by ~30% and memory by ~50% in both training and serving; 3) **FlashAttention** primarily benefits training, reducing runtime by ~50% and activation memory by ~58%. Inference gains are modest (~11−12% for latency and memory), as attention dominates training costs with larger batches and backpropagation; 4) **Mixed precision with recomputation** delivers the largest serving gains: p99 latency improves by ~69% and inference memory by ~30%, as inference can operate end-to-end in low precision. By contrast, training must retain full-precision optimizer states and gradient accumulators; even so, training runtime and memory improve by ~32% and ~49%.

These results demonstrate the effectiveness of LLM optimizations for large-scale recommendation. Building on the ablations conducted on **OneTrans$_S$**, we scale up to **OneTrans$_L$** and show that with these techniques, **OneTrans$_L$** maintains online efficiency comparable to the much smaller DCNv2+DIN baseline (Table 4). This demonstrates again that reformulating RecSys into a *unified* Transformer backbone enables seamless adoption of LLM optimizations, unlocking effective scaling previously unattainable in traditional *encode-then-interaction* architectures.

4.5 RQ4: Scaling-Law Validation

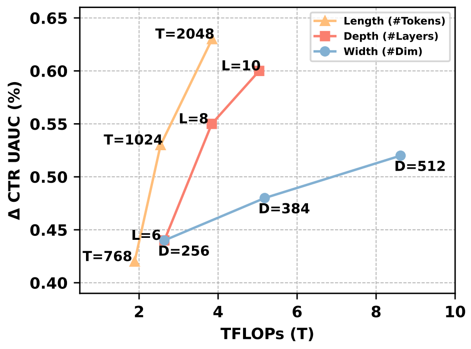
We probe *scaling laws* for OneTrans along three axes: (1) **length** - input token sequence length, (2) **depth** - number of stacked blocks, and (3) **width** - hidden-state dimensionality.

As shown in Fig. 3(a), increasing *length* yields the largest gains by introducing more behavioral evidence. Between *depth* and *width*, we observe a clear trade-off: increasing *depth* generally delivers larger performance improvements than simply widening *width*, as deeper stacks extract higher-order interactions and richer abstractions. However, deeper models also increase serial computation, whereas widening is more amenable to parallelism. Thus, choosing between *depth* and *width* should balance performance benefits against system efficiency under the target hardware budget.

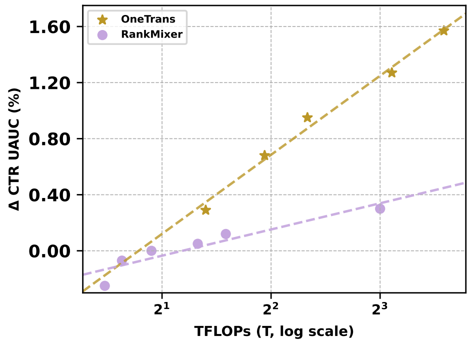
We further analyze scaling-law behavior by jointly widening and deepening OneTrans, and — for comparison — by scaling the **RankMixer+Transformer** baseline on the RankMixer side till 1B; we then plot ΔUAUC versus training FLOPs on a log scale. As shown in Fig. 3(b), OneTrans and RankMixer both exhibit clear log-linear trends, but OneTrans shows a *steeper* slope, likely because RankMixer-centric scaling lacks a unified backbone and its MoE-based expansion predominantly widens the FFN hidden

Table 5: Impact of variants against the unoptimized OneTrans$_S$. Memory is peak GPU usage.

Variant	Training		Inference	
	Runtime (ms)	Memory (GB)	Latency (p99; ms)	Memory (GB)
Unoptimized OneTrans$_S$	407	53.13	54.00	1.70
+ Pyramid stack	−28.7%	−42.6%	−8.4%	−6.9%
+ Cross-Request KV Caching	−30.2%	−58.4%	−29.6%	−52.9%
+ FlashAttention	−50.1%	−58.9%	−12.3%	−11.6%
+ Mixed Precision with Recomputation	−32.9%	−49.0%	−69.1%	−30.0%



(a) Trade-off: FLOPs vs. ΔUAUC



(b) Scaling law: ΔUAUC vs. FLOPs (log)

Figure 3: Comparison of trade-off and scaling law.

dimension. Together, these results suggest that OneTrans is more *parameter- and compute-efficient*, offering favorable performance–compute trade-offs for industrial deployment.

4.6 RQ5: Online A/B Tests

We assess the business impact of OneTrans in two large-scale industrial scenarios: (i) *Feeds* (home feeds), and (ii) *Mall* (the overall setting that includes Feeds and other sub-scenarios). Traffic is split at the user/account level with hashing and user-level randomization. Both the *control* and *treatment* models are trained and deployed with the past 1.5 years of production data to ensure a fair comparison.

Our prior production baseline, **RankMixer+Transformer**, serves as the *control* (≈ 100M neural-network parameters) and does not use sequence KV caching. The *treatment* deploys OneTrans$_L$ with the serving optimizations described in Section 3.5.

Table 6: Online A/B results: OneTrans$_L$ (treatment) vs. RankMixer+Transformer (control). Order/u and GMV/u are relative deltas (%). Latency is the *relative* end-to-end per-impression change Δ% (lower is better). * denotes $p < 0.05$, and ** for $p < 0.01$

Scenario	order/u	gmv/u	Latency (Δ%; p99)↓
Feeds	+4.3510%*	+5.6848%*	−3.91%
Mall	+2.5772%**	+3.6696%*	−3.26%

We report user-level **order/u** and **gmv/u** as relative deltas (Δ%) versus the *RankMixer+Transformer* control with two-sided 95% CIs (user-level stratified bootstrap), and **end-to-end latency**—measured as the relative change in p99 per-impression time from request arrival to response emission (Δ%; lower is better). As shown in Table 6, OneTrans$_L$ delivers consistent gains: in *Feeds*, +4.3510% `order/u`, +5.6848% `gmv/u`, and −3.91% latency; in *Mall*, +2.5772% `order/u`, +3.6696% `gmv/u`, and −3.26% latency — indicating the unified modeling framework lifts business metrics while reducing serving time relative to a strong non-unified baseline.

We further observe a +0.7478% increase in user *Active Days* and a significant improvement of +13.59% in *cold-start product order/u*, highlighting the strong generalization capability of the proposed model.

5 Conclusion

We present OneTrans, a unified Transformer backbone for personalized ranking to replace the conventional *encode–then–interaction*. A unified tokenizer converts both sequential and non-sequential attributes into one token sequence, and a unified Transformer block jointly performs sequence modeling and feature interaction via shared parameters for homogeneous (sequential) tokens and token-specific parameters for heterogeneous (non-sequential) tokens. To make the unified stack efficient at scale, we adopt a pyramid schedule that progressively prunes sequential tokens and a cross-request KV Caching that reuses user-side computation; the design further benefits from LLM-style systems optimizations (e.g., FlashAttention, mixed precision). Across large-scale evaluations, OneTrans exhibits near log-linear performance gains as width/depth increase, and delivers statistically significant business lifts while maintaining production-grade latency. We believe this unified design offers a practical way to scale recommender systems while reusing the system optimizations that have powered recent LLM advances.

References

[1] 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. *arXiv preprint arXiv:2505.04421* (2025).

[2] Jianxin Chang, Chenbin Zhang, Yiqun Hui, Dewei Leng, Yanan Niu, Yang Song, and Kun Gai. 2023. Pepnet: Parameter and embedding personalized network for infusing with personalized prior information. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3795–3804.

[3] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, et al. 2010. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research* 11, 4 (2010).

[4] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior Sequence Transformer for E-commerce Recommendation in Alibaba. arXiv:1905.06874 [cs.IR] https://arxiv.org/abs/1905.06874

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. arXiv:1606.07792 [cs.LG] https://arxiv.org/abs/1606.07792

[6] 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.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems* 35 (2022), 16344–16359.

[8] Tri Dao, Aleksander Thomas, Anima Anandkumar, Matei Zaharia, and Christopher Re. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *arXiv preprint arXiv:2307.08691* (2023).

[9] 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:1905.06482 [cs.IR] https://arxiv.org/abs/1905.06482

[10] Audrunas Gruslys, Remi Munos, Ivo Daniel, Oriol Vinyals, and Koray Kavukcuoglu. 2016. Memory-Efficient Backpropagation through Time. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[11] 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:2311.05884 [cs.IR] https://arxiv.org/abs/2311.05884

[12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, Xiuqiang He, and Zhenhua Dong. 2018. DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction. arXiv:1804.04950 [cs.IR] https://arxiv.org/abs/1804.04950

[13] Junjie Huang, Jizheng Chen, Jianghao Lin, Jiarui Qin, Ziming Feng, Weinan Zhang, and Yong Yu. 2024. A comprehensive survey on retrieval methods in recommender systems. *arXiv preprint arXiv:2407.21022* (2024).

[14] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. arXiv:1808.09781 [cs.IR] https://arxiv.org/abs/1808.09781

[15] 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).

[16] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. 2017. Cascade ranking for operational e-commerce search. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1557–1565.

[17] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740* (2017).

[18] Paulius Micikevicius, Sharan Narang, Jonah Alben, Greg Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *International Conference on Learning Representations (ICLR)*.

[19] Nikil Pancha, Andrew Zhai, Jure Leskovec, and Charles Rosenberg. 2022. Pinnerformer: Sequence modeling for user representation at pinterest. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*. 3702–3712.

[20] 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*. 2685–2692.

[21] Jiarui Qin, Jiachen Zhu, Bo Chen, Zhirong Liu, Weiwen Liu, Ruiming Tang, Rui Zhang, Yong Yu, and Weinan Zhang. 2022. Rankflow: Joint optimization of multi-stage cascade ranking systems as flows. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 814–824.

[22] 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 (CIKM '19)*. ACM, 1161–1170. doi:10.1145/3357384.3357925

[23] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. arXiv:1904.06690 [cs.IR] https://arxiv.org/abs/1904.06690

[24] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. arXiv:1708.05123 [cs.LG] https://arxiv.org/abs/1708.05123

[25] 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 (WWW '21)*. ACM, 1785–1797. doi:10.1145/3442381.3450078

[26] 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.

[27] Zhichen Zeng, Xiaolong Liu, Mengyue Hang, Xiaoyi Liu, Qinghai Zhou, Chaofei Yang, Yiqun Liu, Yichen Ruan, Laming Chen, Yuxin Chen, et al. 2024. Interformer: Towards effective heterogeneous interaction learning for click-through rate prediction. *arXiv preprint arXiv:2411.09852* (2024).

[28] 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).

[29] Biao Zhang and Rico Sennrich. 2019. Root mean square layer normalization. *Advances in neural information processing systems* 32 (2019).

[30] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2018. Deep Interest Evolution Network for Click-Through Rate Prediction. arXiv:1809.03672 [stat.ML] https://arxiv.org/abs/1809.03672

[31] Guorui Zhou, Chengru Song, Xiaoqiang Zhu, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. arXiv:1706.06978 [stat.ML] https://arxiv.org/abs/1706.06978

[32] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1079–1088.

[33] 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, and Zuotao Liu. 2025. RankMixer: Scaling Up Ranking Models in Industrial Recommenders. arXiv:2507.15551 [cs.IR] https://arxiv.org/abs/2507.15551