

# Request-Only Optimization for Recommendation Systems

Liang Guo, Wei Li, Lucy Liao \*, Huihui Cheng, Rui Zhang, Yu Shi, Yueming Wang, Yanzun Huang, Keke Zhai, Pengchao Wang, Timothy Shi, Xuan Cao, Shengzhi Wang, Renqin Cai, Zhaojie Gong, Omkar Vichare, Rui Jian, Leon Gao, Shiyang Deng, Xingyu Liu, Xiong Zhang, Fu Li \*, Wenlei Xie \*, Bin Wen \*, Rui Li, Xing Liu, Jiaqi Zhai \*  
Meta Platforms, Inc., USA

## Abstract

Deep Learning Recommendation Models (DLRMs) represent one of the largest machine learning applications on the planet. Industry-scale DLRMs are trained with petabytes of recommendation data to serve billions of users every day. To utilize the rich user signals in the long user history, DLRMs have been scaled up to unprecedented complexity, up to trillions of floating-point operations (TFLOPs) per example. This scale, coupled with the huge amount of training data, necessitates new storage and training algorithms to efficiently improve the quality of these complex recommendation systems.

In this paper, we present a Request-Only Optimizations (ROO) training and modeling paradigm. ROO simultaneously improves the storage and training efficiency as well as the model quality of recommendation systems. We holistically approach this challenge through co-designing data (i.e., request-only data), infrastructure (i.e., request-only based data processing pipeline), and model architecture (i.e., request-only neural architectures). Our ROO training and modeling paradigm treats a user request as a unit of the training data. Compared with the established practice of treating a user impression as a unit, our new design achieves native feature deduplication in data logging, consequently saving data storage. Second, by de-duplicating computations and communications across multiple impressions in a request, this new paradigm enables highly scaled-up neural network architectures to better capture user interest signals, such as Generative Recommenders (GRs) and other request-only friendly architectures.

Our proposed ROO training and modeling paradigm has been deployed to three major recommendation products, each with billions of active users. ROO data format allows for increasing the training sample volume by 43% to 150% across these three products. ROO training optimization yields a substantial increase in training throughput – up to 570% for the retrieval and early-stage ranking models, and between 32% and 100% for the late-stage ranking models. Combined with ROO-based neural architectures like Hierarchical Sequential Transduction Units (HSTU), ROO scales model FLOPs by 7x utilizing the same amount of training compute, enabling significant offline and online metric wins. Our studies

offer practical and scalable design solutions for engineers seeking to build efficient and effective large-scale recommendation systems.

## CCS Concepts

• **Computing methodologies** → **Artificial intelligence; Neural networks**; • **Information systems** → **Recommender systems**.

## Keywords

Recommender Systems; User Interest modeling

## ACM Reference Format:

Liang Guo, Wei Li, Lucy Liao [1], Huihui Cheng, Rui Zhang, Yu Shi, Yueming Wang, Yanzun Huang, Keke Zhai, Pengchao Wang, Timothy Shi, Xuan Cao, Shengzhi Wang, Renqin Cai, Zhaojie Gong, Omkar Vichare, Rui Jian, Leon Gao, Shiyang Deng, Xingyu Liu, Xiong Zhang, Fu Li [1], Wenlei Xie [1], Bin Wen [1], Rui Li, Xing Liu, Jiaqi Zhai [1]. 2018. Request-Only Optimization for Recommendation Systems. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 12 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 Introduction

Deep Learning Recommendation Models (DLRMs) [6, 28] are the backbone of recommendation systems serving billions of users daily. The scale at which these models operate creates substantial computational and infrastructure challenges across the entire pipeline – from data generation and pre-processing to GPU training and inference [11, 14, 20]. These challenges necessitate new solutions that reduce costs while maintaining or improving model quality.

A fundamental inefficiency in current recommendation systems lies in their data collection and processing pipelines. As illustrated in Figure 1, traditional systems follow a two-phase approach: an impression phase where multiple content items are sent in response to a user request, followed by a feedback phase where user actions are collected and joined with features to create training samples.

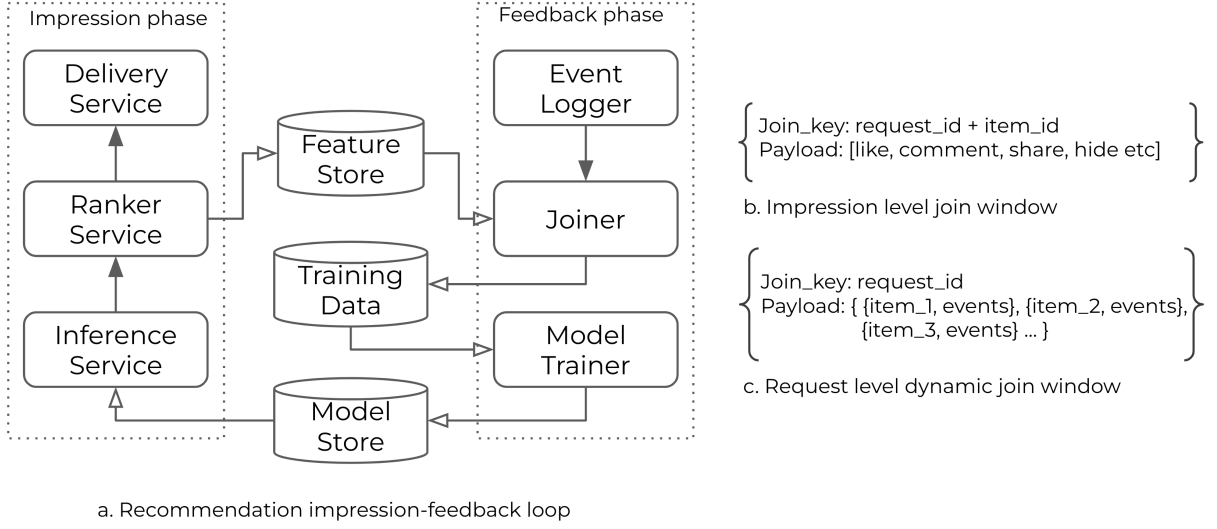
The critical issue emerges at the impression-level event-feature joiner (Figure 1.b), where extensive feature duplication occurs. For each user request that generates multiple impressions, identical user features are redundantly stored across all impression-level training samples. Figure 2 reveals that industrial recommendation products typically generate 4-7 impression samples per request. This redundancy cascades throughout the system – consuming storage, network bandwidth, and GPU computation resources.

This inefficiency is particularly pronounced for modern recommendation models that leverage long user history sequences. These sparse user features constitute the majority of features in training samples, as demonstrated in previous work [28]. The growing adoption of sequential modeling techniques further exacerbates

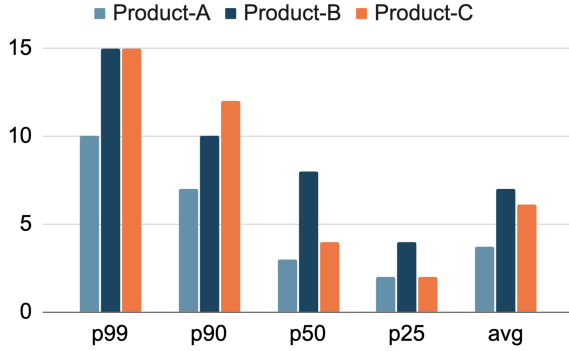
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components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

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 978-1-4503-XXXX-X/2018/06  
<https://doi.org/XXXXXXX.XXXXXXX>

<sup>1</sup>Work done while at Meta.



**Figure 1: A traditional training data pipeline incurs excessive feature duplication at the source of data generation, i.e. impression-level event-feature joiner (b), wasting significant computational resources throughout the system. A request-level joiner (this work) replaces the traditional joiner and captures all impressions belonging to the same request in one single sample (c).**



**Figure 2: Number of items (i.e., number of impression-level samples) within requests across three major recommendation products on a billion-user-scale platform.**

this challenge, as these approaches rely heavily on the very user sequence features being redundantly processed [21, 25, 29, 41].

While Request-Only Optimization (ROO) has been recently adopted for model inference [39], extending this concept to training presents significant challenges. Existing ROO implementations often act as wrappers around impression-level data at inference time, increasing system complexity. Potential training-time optimization is complicated by the widespread use of impression-based data formats throughout client and server infrastructures, and the incremental arrival of impression-based events during the feedback phase.

Our work presents a novel training data format, Request-Only Optimization (ROO), that captures all impression-level samples corresponding to the same request in a single training sample. This data format distinctly separates request-only (RO) data from non-request-only, or impression-level (NRO) data. This clean separation enables user sequence tensors to be processed exactly once across

all impressions from the same request or app session, eliminating redundant computation of user-side features.

Building on this foundation, we develop ROO training methodologies and ROO-friendly model architectures that capitalize on this data structure. By processing user-side information at the request level rather than the impression level, we reduce the computational workload from  $B_{NRO}$  to  $B_{RO}$  examples (where  $B_{NRO}$  represents the number of impression items in a batch and  $B_{RO}$  represents the number of requests). With an average of 4-7 impressions per request ( $B_{NRO}/B_{RO} = 4 - 7$ ) and user-side features dominating the feature space (Figure 2), our approach yields theoretical speedups of 300-600%.

Our ROO training paradigm delivers substantial benefits for retrieval and early-stage ranking models, which utilize minimal cross-features. These models demonstrate up to 6x improvements in training throughput with minimal code changes. For late-stage ranking models, we leverage ROO amortization to apply user feature compression, resulting in significant quality gains without increased computational costs. Most importantly, the dramatic reduction in GPU training costs has enabled the adoption of previously prohibitive modeling technologies such as Generative Recommenders (GRs) [41]. This represents a fundamental shift in what becomes computationally feasible in production recommendation systems.

Overall, our paper makes the following key contributions:

- A new Request-Only Optimization (ROO) training data format that eliminates pervasive feature duplication at the source (Section 2), increasing training sample volumes by 43% to 150% across different recommendation products utilizing the same storage capacity.
- A new ROO training paradigm that maximizes DLRM system efficiency through computation redundancy elimination

and at the same time, unifying the data format used across training and inference tech stacks.

- Novel neural model architectures including UserArch, ROO sequential models, and HSTU that fully leverage the cost amortization benefits of ROO training (Section 3).
- Extensive evaluation of ROO-based data formats and architectures throughout feature preprocessing, training, and inference across multiple ranking/retrieval stages for billion-user scale products, with experimental validation demonstrating significant efficiency gains and quality improvements in production (Section 4).

## 2 End-to-End ROO Design

In this work, we take a first-principled approach and address the fundamental problem at the root level of data for DLRM training and inference. The end-to-end Request-Only Optimization training paradigm benefits from a cohesive co-design with data, system, and model architecture. The well-designed data format eliminates the feature duplication at the source and enables us to hatch a systematic implementation throughout storage, GPU resource usage, training, and inference. The synergy between ROO and advanced modeling techniques, such as Generative Recommenders [41], strikes a great balance between model complexity and efficiency.

### 2.1 Request-Level Training Data

We redesign the training sample format as request level data format. In the recommendation feedback phase, a request level joiner is used to buffer user-item interaction events as grouped by unique request ids (Figure 1.c). When the join window closes, it creates a compact ROO training example that has one copy of user features in the RO part and an array of item features in the NRO part (Table 2). Besides the request level joiner, there is no additional infrastructure change in the training data pipeline. This is because the design eliminates user feature redundancy at the root data level and furthermore it fosters efficiency optimizations in the upstream model feature preprocessing and training pipeline.

**2.1.1 Request-Level Data Ingestion** The ROO training data schema (Table 2) allows RO and NRO features to be organized separately in feature flattening storage format [44] that enables better columnar storage compression ratio. ROO training samples are typically read in a mini-batch by data preprocessing (DPP) workers. Features are read from feature flattened tables into columnar major in-memory format for further feature preprocessing and tensor transformation. In each mini-batch, RO features are transformed into a 2D tensor representing RO user float features, along with RO user ID-list features structured as a PyTorch KeyedJaggedTensor [19]. NRO features are similarly processed, with flattening along the candidate dimension to form an 2D NRO item float feature tensor and an NRO item ID-list KeyedJaggedTensor. The batch sizes for RO and NRO features, denoted as  $B_{RO}$  and  $B_{NRO}$  respectively, are inherently distinct. To avoid redundant duplication of user features for batch size alignment, a NRO tensor is used to specify the number of impressions per ROO training sample, thereby minimizing data copying and transformation of duplicated user features. Furthermore, tensor operations such as dense normalization, sequence merging, deduplication, and masking are optimized to enhance computational

efficiency and performance. The efficiency of the ROO data schema extends from storage to DPP workers and greatly scales training data ingestion for DLRM and GR model training needs.

**2.1.2 ROO Data Activity-to-Serving Latency** In recommendation products, activity-to-serving (ATS) latency is used to measure personalization models' freshness, and activity-to-training latency is a critical part of it. In theory, adopting request-level joining could potentially increase activity-to-training latency because it needs to wait for multiple items within the request join window. In production, request-level joiners use either fixed-time policy or dynamic training trigger to close the join window before the fixed-time window closes. Through measurements, we establish that the average time gap between the earliest user-item interaction event arriving at the request join window buffer and the last item of the request is within 16 minutes. In practice, ROO training samples' data landing latency is about the half the fixed-time join window and satisfies model ATS requirements.

**2.1.3 ROO Expansion for Backward Compatibility** Despite its significant storage efficiency advantage, request-level training sample schema is incompatible with the traditional DLRMs trained with impression-level schema. In a typical industrial environment, hundreds of models across multiple ranking stages collectively serve billions of users using a product, and not all models have to adopt the ROO training paradigm due to return-on-investment (ROI) considerations. We present a practical way to minimize the migration cost for these models and facilitate the adoption of ROO data schema. We introduce ROO expansion data adapter in the feature preprocessing layer. For each mini-batch, the data adapter expands a ROO sample into multiple in-memory impression samples, with user-side features duplicated to the in-memory data frames. Trainer side rebatching is leveraged to keep the consistent training batch size. This equips ROO data schema with backward compatibility and eliminates unnecessary model rewrite efforts. The expansion functionality simplifies the migration process; major products fully migrated all retrieval, early stage ranking and late stage ranking models to ROO data within a 9-month time frame.

### 2.2 Unifying Model Training and Inference Request-Level Optimization

The ROO feature preprocessing ensures the same model input format for both training and inference stacks. This opens up engineering opportunities to simplify the DLRM system with significant performance improvements.

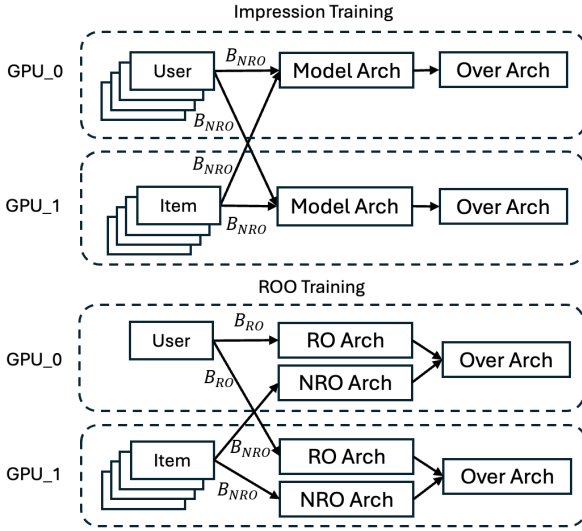
In a trainer mini-batch, the user-side RO and item-side NRO feature tensors have different batch sizes. The RO tensor batch size  $B_{RO}$  is the number of ROO training samples, while the NRO tensor batch size  $B_{NRO}$  is the sum of impression items across all request level samples in the trainer mini-batch. This allows trainers to leverage TorchRec's variable-length batch size feature [19] for model parallel between RO and NRO computation. The system efficiency gains primarily come from the user-side RO computation deduplication. User-side RO feature embedding lookup and associated all2all communication overhead are reduced to exactly once for each request-level sample in the mini-batch. Compared to impression level training, as illustrated in Figure 3, the computation and

request-id	conversions	dense features	id-list features	id-score-list features
int	list<int>	map<int, float>	map<int, float>	map<int, map<int, float>>

Table 1: Impression-level training sample schema

request-id	conversions	user dense features	user id-list features	item dense features	item id-list features
int	list<list<int>>	map<int, float>	map<int, list<int>>	map<int, list<float>>	map<int, list<list<int>>

Table 2: Request level training sample schema



**Figure 3: Compared to impression training, ROO training deduplicates RO-side embedding lookup, embedding all2all communication, RO-only model arch’s computation FLOPs from  $B_{NRO}$  batch size to  $B_{RO}$  batch size. In the example, user and item embedding tables are sharded across GPU-0 and GPU-1 accordingly;  $B_{NRO}$  is 4 and  $B_{RO}$  is 1.**

communication complexity is reduced from  $O(B_{NRO})$  to  $O(B_{RO})$ . In our extensive experiments in DLRM systems (Section 4), the RO model arch reduced overall FLOPs significantly as a result of the ROO-aware training paradigm.

Traditional DLRM training and inference had different model input feature data formats that led to diverging code stacks. At the model inference side, request level features were explicitly deduplicated through coordinated inference server-client operations to save network and GPU resources. These request level optimizations were premature because they only partially mitigated the problems but added layers of complexity without providing a universal solution.

Similar to ROO training, ROO inference feature preprocessing allows DLRM inference to easily distinguish RO and NRO features within the inference request and therefore passing RO feature to recommendation models for deferred user-side feature fanout inside the model. It removes unintuitive client-side user feature broadcast and server-side deduplication. Effectively, the ROO training

paradigm unifies DLRM training and inference stacks, while dramatically simplifying the DLRM system with scalability improvements and engineering cost reduction.

### 3 ROO Model Architectures

Co-designing with the ROO data format and training paradigm, we investigate a novel family of DLRMs that have highly scaled-up complexity on the RO part of the model, where the cost of scaling up is largely amortized by the computation deduplication in ROO.

#### 3.1 Simple RO Scale-Up: User Tower in Two-Tower Like Models

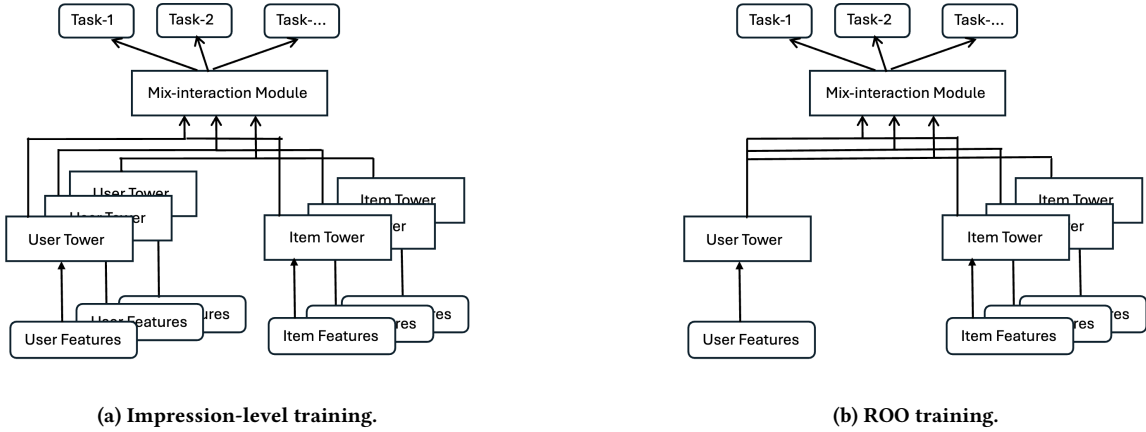
In the funnel of typical industrial recommendation systems, the retrieval model and the early-stage ranking (ESR) model [10], despite having different optimization goals, usually consist of a user tower that learns user representations, an item tower that learns item representations, and an optional user-item mix-interaction module fusing the user representations with item representations to capture the user-item interactions. We illustrate one such example in Figure 4.

As the user tower only processes the user features and the impressions in a request have the same user features (RO features from Section 2.1), the user tower computation for impressions in a request can be deduplicated, and the user representation is learned for a request and shared by these impressions.

#### 3.2 RO-side Scale Up with UserArch

Traditional deep learning recommendation models (DLRMs) have been shown to scale effectively with an increasing number of features, leading to improved performance. However, this comes at the cost of increased complexity in the interaction arch, e.g. DCNv2 [36] and DHEN [42]. In the context of recommendation systems, features can be broadly categorized into user features, candidate features, and cross features that capture the relationships between the user and candidate. Notably in recommendation systems, user features dominate the feature landscape, accounting for the majority of the features in typical ranking models for recommendation services. In late-stage ranking (LSR) models, the interaction between user and item features occurs early in the neural network, limiting the benefits of ROO training primarily to embedding table lookups or inter-GPU communications for common DLRM models.

To fully exploit the advantages of ROO training, we designed UserArch to efficiently process and integrate user-side features.



**Figure 4: The two-tower like models in the recommendation system with lightweight interaction between the two towers, e.g. retrieval models or early-stage ranking model. Figure (a) shows the models in the traditional impression-level training mode, figure (b) shows the models in the ROO training mode.**

Specifically, we employ a simple cross-feature interaction architecture, Linear Compress Embedding (LCE) [24, 34, 42], to compress the dimensionality of user features before they enter the post-ROO architecture. Let  $X \in R^{B \times d_{in} \times n_{in}}$  be the tensor of user features, where  $B$  is the batch size,  $d_{in}$  is the embedding dimensions of each input features, and  $n_{in}$  is the number of input user features. The LCE module would first compress the number of embeddings to the output number of embeddings  $n_{out}$ :

$$f(X) = b + g(X) \odot W \quad (1)$$

where  $b \in R^{1 \times n_{out}}$ ,  $g(X)$  will then reshape and permute  $X$  to  $R^{B \times d_{in} \times n_{in}}$ , and  $W \in R^{n_{in} \times n_{out}}$ . Then another linear layer will project all the compressed embeddings to an output dimension  $d_{out}$ :

$$f'(f(X)) = b' + g'(f(X)) \odot W' \quad (2)$$

where  $b' \in R^{1 \times d_{out}}$ ,  $g'(f(X))$  will the reshape and permute  $f(X)$  to  $R^{B \times n_{out} \times d_{in}}$ , and  $W' \in R^{d_{in} \times d_{out}}$ . Note that similar to the sequential modeling case, the computational cost of UserArch itself can also be amortized by ROO.

This compression enables the inclusion of more user-side features without significantly increasing the size of the post-ROO architecture, thereby enhancing user-centric modeling while maintaining model efficiency. By amortizing the cost of RO-side feature generation through ROO, our UserArch design allows for the efficient incorporation of additional user-side features, ultimately improving model performance and scalability. As illustrated in Figure 8, in LSR models, the UserArch outputs are fed into the post-ROO architecture, adding only ROO amortized extra computation to the model before the interaction architecture of traditional DLRM for ranking.

### 3.3 ROO-based Autoregressive Modeling

Most importantly, ROO addresses challenges associated with recent trends to move to autoregressive model training in RecSys.

ROO enables scalable autoregressive modeling, with a commonly used variant, generative recommenders (GRs) [1, 15, 18, 41, 43] as

follows. Denote a sequence of  $n$  past impressions in user history  $RO_0, RO_1, \dots, RO_{n-1}$  ( $RO_i \in \mathbb{X}$ ) ordered chronologically, and a sequence of  $m$  target items in the request  $NRO_0, NRO_1, \dots, NRO_{m-1}$  ( $NRO_i \in \mathbb{X}$ ), where  $\mathbb{X}$  denotes the set of all items served in the product. With each item associated with an action feature  $a$  and various contextual features including but not limited to surface type, timestamp, denoted as  $c$ , sequential modeling in standard retrieval and ranking models can be formulated as shown in Figure 5b.

**Amortized training cost with ROO.** In a traditional impression-based setup, the total computational requirement for self-attention based sequential transduction architectures, such as Transformers [35], scales as  $m \cdot (n^2 d + n d^2)$  for all impressions in the request. In our proposed ROO setup, the cost goes down to  $(n+m)^2 d + (n+m) d^2$ , where  $n$  represents the length of the user history, which is typically in the order of thousands, and  $m$  represents the request size, which is typically in the order of tens. In a moderate scenario where  $n = 1000, m = 10, d = 256$ , the saving for computational cost would be 9.82x for the sequential encoder module.

**HSTU.** For autoregressive modeling-related experiments in this paper, we apply HSTU [41] – a variant of transformer designed for industrial-scale recommendation systems with large, non-stationary vocabularies – over the user sequence for aggregating and capturing nuanced interests and interactions. We scale up HSTU leveraging the opportunity in deduplicating computation that depends only on the RO part of the ROO data.

## 4 Experiments

### 4.1 End-to-End System Efficiency Savings

ROO training data captures all impressions of the same request in one single request-level training sample and hereby eliminates duplications at the fundamental data level. Through data and system co-design, we eliminated computational redundancy throughout training data pipeline, intra-datacenter network transfer, feature preprocessing, GPU training, and model inference. As a result, ROO training achieved multi-fold efficiency benefits.

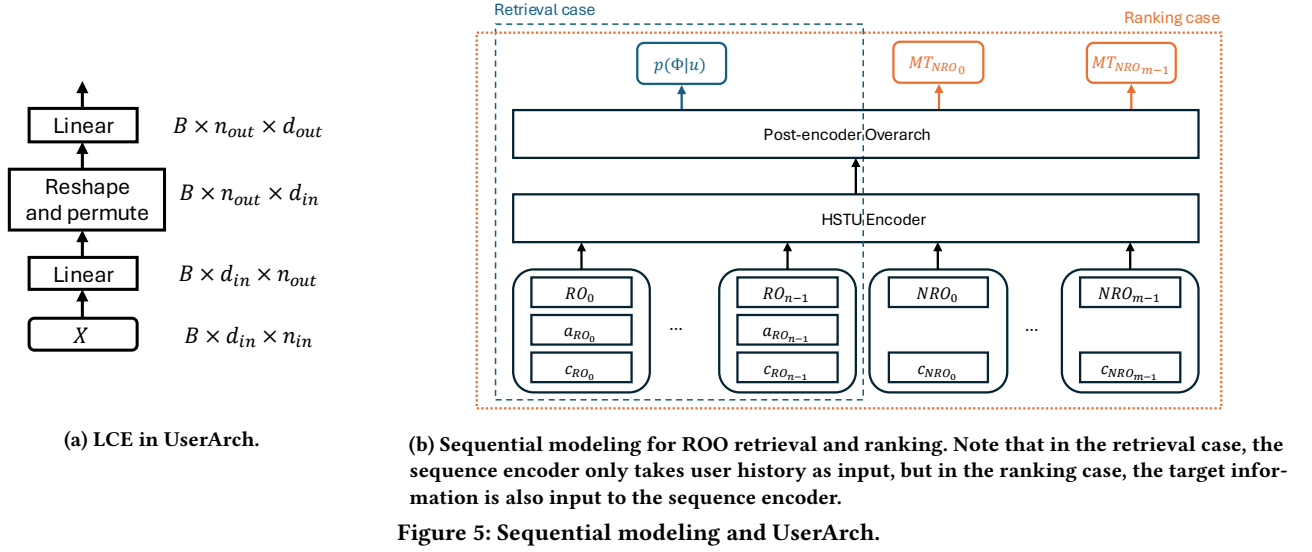


Figure 5: Sequential modeling and UserArch.

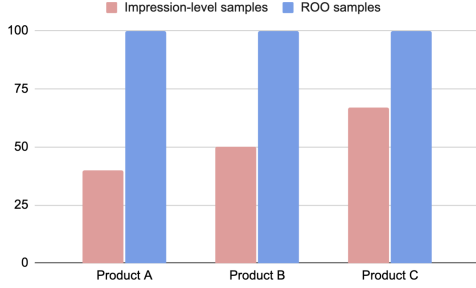


Figure 6: Training sample volume increase across different recommendation products, eliminating data downsampling.

ROO data allows logging more training samples and avoids down-sampling with the same storage capacity. As shown in Figure 6, we observed 43% to 150% increases in training sample volumes across a few recommendation products.

The adoption of ROO training optimization yields substantial benefits, including significant reductions in GPU training costs and notable increases in training throughputs. This breakthrough enables the exploration of previously cost-prohibitive modeling technologies, such as UserArch and GRs/HSTU, which were formerly hindered by computational constraints. To demonstrate the efficacy of ROO training, we conducted experiments on multiple models across various recommendation stages without modifying the model architecture. The results, presented in Figure 7, show an increase in training throughput of 48% to 570% in retrieval models, 125% to 266% in early stage ranking models, and 32% to 100% in late stage ranking models, achieved through our proposed ROO optimization technique.

## 4.2 Model Quality Improvements

We next conduct a range of experiments in the retrieval, early-stage ranking (ESR), and late-stage ranking (LSR) models from three billion-user scale recommendation products, hereinafter referred

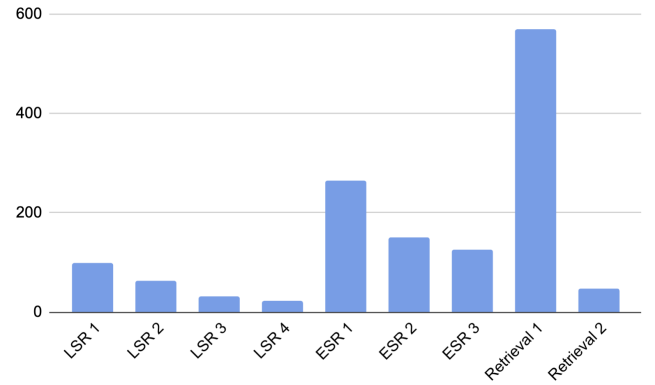


Figure 7: Training throughput increase (in percentage) on multiple models across the recommendation stages.

to as product-A, -B, and -C. These experiments validate the practicality of the ROO paradigm and the related model architecture innovations in production environments. We also provide detailed ablation studies and online A/B test results to showcase the impact from ROO in production.

In the following subsections that discuss offline studies of early-stage and late-stage ranking models (ESR, LSR), we use normalized entropy (NE) [16] as evaluation metrics, which indicates better accuracy if the value is reduced. We use Recall@K for retrieval models, where better models have better recall metrics.

## 4.3 Retrieval and Early-Stage Ranking Models

Through reducing training costs per item example by deduplicating user tower computation costs across items in a logged request, ROO has enabled the development of more complex user architectures for retrieval and early-stage ranking models.



**Table 3: Relative FLOPs and offline/online metric comparisons (Recall@100 for the main C-/E-Task) for retrieval models.**

Model		FLOPs per example	C-Task Recall@100	E-Task Recall@100	Topline Consumption
Retrieval	HSTU (Impression)	6.8x	+3.53%	+2.35%	–
	HSTU (ROO)	0.99x	+3.87%	+2.41%	+4.77%
	Baseline	1.0x	+0.0%	+0.0%	+0.0%

**4.3.1 Retrieval Models** Table 3 illustrates the value of ROO optimization for model scale-up. Here we apply a dual encoder-based retrieval model [6, 17, 48] to illustrate relative training cost and model quality comparisons when the same model is migrated to ROO, although our approach can be easily extended to recent work including multi-embeddings [23], generative retrieval [12, 32], and learned similarities [7, 40].

Without the ROO data format, model scaling can contribute to improved offline results at the expense of 6.8x more computation costs per training example. With ROO, the same improvements in offline recall can be attained with no increase in training cost per item example as we now effectively amortize computational cost across multiple impressions. We collected online A/B test metrics were collected from a large-scale experiment on product-B. We observed a 4.77% increase in the main online consumption metric after launching the HSTU architecture on ROO.

**4.3.2 Early-stage Ranking (ESR) Models** With the ROO paradigm, we further integrated HSTU into the user tower of the ESR models. As shown in Table 4, HSTU improved the recommendation quality as measured both offline NE and post launch topline impact. Specifically, this design leads to significant boost of the main consumption metric by 2.43% and the main topline metric by 0.04% to product-B, where 0.24% and 0.03% are significant, respectively.

## 4.4 Late-Stage Ranking (LSR) Models

We finally conduct extensive experiments on four LSR models across three recommendation products.

**LSR on product-A.** In this experiment, we conducted an offline ablation analysis on a late-stage ranking model for product-A. We incorporated two ROO architectures into the model: UserArch and HSTU. On this model, we deployed UserArch to production first, followed by HSTU, which was tested and rolled out subsequently. From both offline and online experiments, HSTU can add notable incremental improvement to the model on top of UserArch. Table 4 summarizes the offline evaluation NE improvement and post-launch online metrics lift. Note that a 0.1% NE improvement on both engagement and consumption tasks is considered substantial for this particular model, and 0.03% lift in the topline metric, 0.63% lift in engagement metric, and 0.06% lift in consumption metric are deemed significant for this product.

**LSR on product-B.** We conducted a comprehensive evaluation of UserArch and HSTU on two distinct models, LSR-1 and LSR-2, within the recommendation service for product-B. In this product, 0.03% lift in the topline metric, 0.63% lift in engagement metric, and 0.24% lift in consumption metric is considered as significant. Our offline and online experiments revealed that these ROO-related technologies yielded notable improvements in normalized entropy

(NE) and post-launch online metrics in both models as shown in Table 4.

**LSR on product-C.** We tested HSTU on a LSR model for product-C, yielding significant improvements in NE. Specifically, we observed 0.81% and 0.79% NE reductions on engagement and consumption tasks, respectively, which are larger than the 0.1% significant level. Following the successful deployment of this model to production, we measured substantial gains in online metrics, including a 0.25% lift in topline metric and a 2.85% increase in engagement metric as shown in Table 4. Normally, 0.03% lift in topline metric, 0.19% in engagement metric and 0.04% in consumption metric are considered significant for this product.

## 5 Related Work

**Data de-duplication.** Normalization [4] is a process to facilitate redundancy removal and data integrity checking in relational databases. The impression-level training sample schema (Table 1) violates the second normal form (i.e. 2NF) [5], which prohibits a non-key column being a fact about only a subset of the key columns [22]. To achieve the 2NF and data de-duplication on the training sample schema, the RO side features should be extracted out to another table with request id being the key. Request-level schema (Table 2), in contrast, exploited the RO feature equivalence within a request by grouping training samples from the same request into one sample. Despite the aforementioned differences, the timelessness idea of normalization inspired the need to remove data redundancy and the design of ROO schema.

In the field of DLRM, data de-duplication has also been investigated [9, 39, 47]. In particular, part of our work in the inference system is reminiscent of the work done by You et al. [39], which described the duplication as “single-user-multiple-candidates” and introduced a “structured feature” to exploit the de-duplication to reduce cost during inference time. However, these studies generally focused on one part of the system (e.g. inference time). Our work provides a comprehensive analysis of feature value de-duplication throughout the end-to-end data, training and inference pipeline.

A previous design RecD [45] addressed the feature duplication problem by implementing a suite of data pipeline optimizations to group impression samples of the same session so that columnar storage compression ratio could be maximized. It introduced an inverse index to reconstruct the impression training samples for feature preprocessing, sparse data distribution, and embedding lookups during model training. Our work differs from RecD with a holistic approach to ensure systematic efficiency across storage, training, and inference. ROO data schema deduplicates feature at the root source of the training data pipeline, as a result it does not entail data pipeline infrastructure changes nor does it increase

**Table 4: Normalized entropy (NE) improvement in offline experiments of ranking models from multiple product surfaces and online A/B test metrics improvements after the models are deployed to production. The significant level of each metric across different models are marked in separate rows. In the LSR model on product-A, UserArch and HSTU are tested and deployed sequentially so that their metrics are listed incrementally. E.g. the two modules brings 0.41% and 0.53% NE improvement (reduction) on E-task respectively. On product-B, two LSR models, 1 and 2, were tested.**

Model		E-Task NE	C-Task NE	Topline	Consumption	Engagement
LSR on product-A	UserArch	-0.41%	-0.42%	+0.04%	+0.08%	-
	+HSTU	-0.53%	-0.57%	+0.12%	+0.18%	+4.8%
	<i>Significant delta on product-A</i>	0.1%	0.1%	0.03%	0.06%	0.63%
ESR on product-B	HSTU	-0.31%	-0.32%	+0.04%	+2.43%	+4.9%
	LSR-1 on product-B	-0.86%	-0.78%	+0.07%	+5.56%	+7.0%
	LSR-2 on product-B	-0.60%	-0.64%	+0.04%	+2.51%	+6.9%
	<i>Significant delta on product-B</i>	0.1%	0.1%	0.03%	0.24%	0.63%
LSR on product-C	HSTU	-0.81%	-0.79%	+0.25%	+1.1%	+2.9%
	<i>Significant delta on product-C</i>	0.1%	0.1%	0.03%	0.04%	0.19%

data landing latency. Its feature preprocessing integrates seamlessly with commonly used tensor data structures (e.g. KeyedJaggedTensors in TorchRec [19]) without any custom data structures like InverseKeyedJaggedTensors, minimizing potential subsequent model rewrite. In appendix, we explain ROO data with more implementation details.

**Efficient scaling of DLRM.** To scale DLRMs, an effective line of thinking is to examine the recommendation life-cycle from an end-to-end point of view, seeking model-system co-design and global optimization opportunities. Efficient application of various hardware (e.g. accelerators, scheduler, SSDs) in different components of the end-to-end recommendation system has been studied in [13, 27, 37]. In particular, some work adapted the ML strategy to enable more efficient usages of the accelerator resources [38, 40]. In addition, leveraging offline/batch compute that has access to more compute and memory resources is also observed in many systems [29, 30].

Optimizing the high computation cost in attention-based methods for long user behavior sequence modeling has attracted a lot of interests. Many work studied cost effective techniques to select sub-user behavior sequences that are relevant to candidates [2, 8, 46]. Following up on DIN’s [46] target attention work, Pi et al. [31] employed a two-stage approach to select target aware subsequences in the general search stage and perform finer grained modeling in the exact search stage. On top of the new two-stage paradigm, Chang, et al. [3] and the more recent work by Lv, et al. [26] adopted a similar two-stage architecture but supported by pre-computing and caching strategies for online inference, which enhanced system performance. Si et al. [33]’s work pre-computed a hierarchical clustering of the long user sequence offline, and allowed subsequence selection based on clusters during inference. Unlike pairwise attention approaches, HSTU [41] leveraged efficiently-designed self-attention sequential transduction units to capture the complex interactions between target and history sequences.

Being orthogonal to the innovative methods above, the ROO paradigm offers a generic solution from the angle of systematic de-duplication and amortization. This approach underscores the

importance of model and system co-design in jointly optimizing data, training and inference processes under the DLRM context.

## 6 Conclusion

In this paper, we introduced the Request-Only Optimization (ROO) training and modeling paradigm, a holistic approach that co-designs data, system, and model architectures to unlock unprecedented efficiency and scalability in deep learning recommendation models. By addressing the feature duplication problem at its source through our innovative ROO training data schema design, we achieved significant gains in system efficiency and unified request-only optimization across training and inference. Our results demonstrate substantial improvements in training throughput, with 2x to 6x efficiency gains across retrieval and ranking models. Moreover, our ROO model architectures have consistently delivered enhanced offline and online A/B test metrics, driving impact across multiple applications serving billions of users worldwide. The widespread adoption of the ROO training and modeling paradigm in major DLRM products processing billion-scale user traffic is a testament to its effectiveness and versatility. We hope that the lessons we share in this paper are useful to industry practitioners seeking to optimize their own deep learning recommendation systems.

## 7 Acknowledgment

This work would not be possible without work from the following contributors (alphabetical order): Bugra Akyildiz, Yuanfei Bi, Duo Chen, Li Chen, Xi Chen, Xianjie Chen, Rex Cheung, Patrick Cullen, Delia David, Fei Ding, Rajeev Dixit, Lu Fang, Md Mustafijur Rahman Faysal, Bo Feng, Liz Guo, Luyi Guo, Daisy Shi He, Michael He, Austin Jenkins, Jie Jiao, Basri Kahveci, Manos Karpapothakis, Keyu Lai, Dai Li, Shen Li, Hao Lin, Weiran Liu, Yinghai Lu, Michael Ly, Matt Ma, Yun Mao, Franco Mo, Min Ni, Jongsoo Park, Jing Qian, Alex Singh, Jason Song, Bingjun Sun, Ruohan Sun, Yimin Tan, Alex Vu, Shanyue Wang, Wenyu Wang, Zellux Wang, Ziqi Wang, Evan Welch, Yue Weng, Yanhong Wu, Ke Xu, Zheng Yan, Zimeng Yang, James Zhang, Jingjing Zhang, Lu Zhang, Qunshu Zhang, Vivian Zhang, Wei Zhang, Yue Zhang, Zhiyun Zhang, Xin Zhuang.



## References

- [1] Fedor Borisov, Lars Hertel, Ganesh Parameswaran, Gaurav Srivastava, Sudarshan Srinivasa Ramanujam, Borja Ochoa, Peng Du, Andrei Akterskii, Neil Daftary, Shao Tang, Daqi Sun, Qiang Charles Xiao, Deepesh Nathani, Mohit Kothari, Yun Dai, and Aman Gupta. 2025. From Features to Transformers: Redefining Ranking for Scalable Impact. *arXiv:2502.03417 [cs.LG]* <https://arxiv.org/abs/2502.03417>
- [2] Yue Cao, Xiaojiang Zhou, Jiaqi Feng, Peihao Huang, Yao Xiao, Dayao Chen, and Sheng Chen. 2022. Sampling is all you need on modeling long-term user behaviors for CTR prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 2974–2983.
- [3] Jianxin Chang, Chenbin Zhang, Zhiyi Fu, Xiaoxue Zang, Lin Guan, Jing Lu, Yiqun Hui, Dewei Leng, Yanan Niu, Yang Song, et al. 2023. TWIN: Two-stage interest network for lifelong user behavior modeling in CTR prediction at kuaishou. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3785–3794.
- [4] Edgar F Codd. 1970. A relational model of data for large shared data banks. *Commun. ACM* 13, 6 (1970), 377–387.
- [5] Edgar F Codd. 1972. Further normalization of the data base relational model. *Data base systems* 6 (1972), 33–64.
- [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 (RecSys '16)*. 191–198.
- [7] Bailu Ding and Jiaqi Zhai. 2025. Retrieval with Learned Similarities. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) (WWW '25). Association for Computing Machinery, New York, NY, USA, 1626–1637. doi:10.1145/3696410.3714822
- [8] Zhichao Feng, Junjie Xie, Kaiyuan Li, Yu Qin, Pengfei Wang, Qianzhong Li, Bin Yin, Xiang Li, Wei Lin, and Shangguang Wang. 2024. Context-based Fast Recommendation Strategy for Long User Behavior Sequence in Meituan Waimai. In *Companion Proceedings of the ACM on Web Conference 2024*. 355–363.
- [9] Kun Gai, Xiaoqiang Zhu, Han Li, Kai Liu, and Zhe Wang. 2017. Learning piecewise linear models from large scale data for ad click prediction. *arXiv preprint arXiv:1704.05194* (2017).
- [10] Luke Gallagher, Ruey-Cheng Chen, Roi Blanco, and J. Shane Culpepper. 2019. Joint Optimization of Cascade Ranking Models. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (Melbourne VIC, Australia) (WSDM '19). Association for Computing Machinery, New York, NY, USA, 15–23. doi:10.1145/3289600.3290986
- [11] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Rifati, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference* (Sydney, NSW, Australia) (ACM SIGCOMM '24). Association for Computing Machinery, New York, NY, USA, 57–70. doi:10.1145/3651890.3672233
- [12] Weihao Gao, Xiangjun Fan, Chong Wang, Jiankai Sun, Kai Jia, Wenzhi Xiao, Ruofan Ding, Xingyan Bin, Hui Yang, and Xiaobing Liu. 2021. Learning An End-to-End Structure for Retrieval in Large-Scale Recommendations. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Virtual Event, Queensland, Australia) (CIKM '21). Association for Computing Machinery, New York, NY, USA, 524–533. doi:10.1145/3459637.3482362
- [13] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S Lee, David Brooks, and Carole-Jean Wu. 2020. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 982–995.
- [14] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottle, Kim Hazelwood, Mark Hempstead, Bill Jia, et al. 2020. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 488–501.
- [15] Ruidong Han, Bin Yin, Shangyu Chen, He Jiang, Fei Jiang, Xiang Li, Chi Ma, Mincong Huang, Xiaoguang Li, Chunzhen Jing, Yueming Han, Menglei Zhou, Lei Yu, Chuan Liu, and Wei Lin. 2025. MTGR: Industrial-Scale Generative Recommendation Framework in Meituan. *arXiv:2505.18654 [cs.IR]* <https://arxiv.org/abs/2505.18654>
- [16] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*. 1–9.
- [17] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based Retrieval in Facebook Search. *CoRR* abs/2006.11632 (2020). *arXiv:2006.11632* <https://arxiv.org/abs/2006.11632>
- [18] Yanhua Huang, Yuqi Chen, Xiong Cao, Rui Yang, Mingliang Qi, Yinghao Zhu, Qingchang Han, Yaowei Liu, Zhaoyi Liu, Xuefeng Yao, Yuting Jia, Leilei Ma, Yinqi Zhang, Taoyu Zhu, Liujie Zhang, Lei Chen, Weihang Chen, Min Zhu, Ruiwen Xu, and Lei Zhang. 2025. Towards Large-scale Generative Ranking. *arXiv:2505.04180 [cs.IR]* <https://arxiv.org/abs/2505.04180>
- [19] Dmytro Ivchenko, Dennis Van Der Staay, Colin Taylor, Xing Liu, Will Feng, Rahul Kindi, Anirudh Sudarshan, and Shahin Sefati. 2022. Torchrec: a pytorch domain library for recommendation systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 482–483.
- [20] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.
- [21] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.
- [22] William Kent. 1983. A simple guide to five normal forms in relational database theory. *Commun. ACM* 26, 2 (1983), 120–125.
- [23] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Pipei Huang, Huan Zhao, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-Interest Network with Dynamic Routing for Recommendation at Tmall. *arXiv:1904.08030 [cs.IR]* <https://arxiv.org/abs/1904.08030>
- [24] Muyang Li, Xiangyu Zhao, Chuan Lyu, Minghao Zhao, Runze Wu, and Ruocheng Guo. 2022. MLP4Rec: A Pure MLP Architecture for Sequential Recommendations. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, Lud De Raedt (Ed.). International Joint Conferences on Artificial Intelligence Organization, 2138–2144. doi:10.24963/ijcai.2022/297 Main Track.
- [25] Chi Liu, Jiangxia Cao, Rui Huang, Kai Zheng, Qiang Luo, Kun Gai, and Guorui Zhou. 2024. Kuaiformer: Transformer-Based Retrieval at Kuaishou. *arXiv preprint arXiv:2411.10057* (2024).
- [26] Xiao Lv, Jiangxia Cao, Shijie Guan, Xiaoyou Zhou, Zhiguang Qi, Yaqiang Zang, Ming Li, Ben Wang, Kun Gai, and Guorui Zhou. 2025. MARM: Unlocking the Future of Recommendation Systems through Memory Augmentation and Scalable Complexity. *arXiv:2411.09425 [cs.IR]* <https://arxiv.org/abs/2411.09425>
- [27] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. 2022. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 993–1011.
- [28] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Allison G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).
- [29] 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.
- [30] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on long sequential user behavior modeling for click-through rate prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2671–2679.
- [31] 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.
- [32] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. 2023. Recommender Systems with Generative Retrieval. In *Thirty-seventh Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=BJ0fQUU32w>
- [33] Zihua Si, Lin Guan, ZhongXiang Sun, Xiaoxue Zang, Jing Lu, Yiqun Hui, Xingchao Cao, Zeyu Yang, Yichen Zheng, Dewei Leng, et al. 2024. Twin v2: Scaling ultra-long user behavior sequence modeling for enhanced ctr prediction at kuaishou. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*. 4890–4897.
- [34] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. 2021. MLP-mixer: an all-MLP architecture for vision. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 1857, 12 pages.
- [35] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
- [36] 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.
- [37] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. RecSSD: near data processing for solid

- state drive based recommendation inference. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 717–729.
- [38] Jiajing Xu, Andrew Zhai, and Charles Rosenberg. 2022. Rethinking personalized ranking at Pinterest: An end-to-end approach. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 502–505.
- [39] Xin You, Hailong Yang, Siqi Wang, Tao Peng, Chen Ding, Xinyuan Li, Bangduo Chen, Zhongzhi Luan, Tongxuan Liu, Yong Li, and Depei Qian. 2024. Exploiting Structured Feature and Runtime Isolation for High-Performant Recommendation Serving. *IEEE Trans. Comput.* 73, 11 (2024), 2474–2487. doi:10.1109/TC.2024.3449749
- [40] Jiaqi Zhai, Zhaojie Gong, Yueming Wang, Xiao Sun, Zheng Yan, Fu Li, and Xing Liu. 2023. Revisiting Neural Retrieval on Accelerators. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 5520–5531.
- [41] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. 2024. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. *arXiv preprint arXiv:2402.17152* (2024).
- [42] 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).
- [43] Luankang Zhang, Kenan Song, Yi Quan Lee, Wei Guo, Hao Wang, Yawen Li, Huifeng Guo, Yong Liu, Defu Lian, and Enhong Chen. 2025. Killing Two Birds with One Stone: Unifying Retrieval and Ranking with a Single Generative Recommendation Model. arXiv:2504.16454 [cs.IR] <https://arxiv.org/abs/2504.16454>
- [44] Mark Zhao, Niket Agarwal, Aarti Basant, Buğra Gedik, Satadru Pan, Mustafa Ozdal, Rakesh Komuravelli, Jerry Pan, Tianshu Bao, Haowei Lu, et al. 2022. Understanding data storage and ingestion for large-scale deep recommendation model training: Industrial product. In *Proceedings of the 49th annual international symposium on computer architecture*. 1042–1057.
- [45] Mark Zhao, Dhruv Choudhary, Devashish Tyagi, Ajay Somani, Max Kaplan, Sung-Han Lin, Sarunya Pumma, Jongsoo Park, Aarti Basant, Niket Agarwal, et al. 2023. RecD: Deduplication for end-to-end deep learning recommendation model training infrastructure. *Proceedings of Machine Learning and Systems* 5 (2023), 754–767.
- [46] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5941–5948.
- [47] Lixi Zhou, Jiaqing Chen, Amitabh Das, Hong Min, Lei Yu, Ming Zhao, and Jia Zou. 2022. Serving deep learning models with deduplication from relational databases. *arXiv preprint arXiv:2201.10442* (2022).
- [48] Chandler Zuo, Jonathan Castaldo, Hanqing Zhu, Haoyu Zhang, Ji Liu, Yangpeng Ou, and Xiao Kong. 2024. Inductive Modeling for Realtime Cold Start Recommendations. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) (*KDD '24*). Association for Computing Machinery, New York, NY, USA, 6400–6409. doi:10.1145/3637528.3671588

**Algorithm 1** Request Level Join

---

```

1: Struct RequestJoinRecord {
2:   Int userId;
3:   Int requestId;
4:   Set<Int> impressions;           ▶ impression items
5:   Map<Int, List<Int>> conversions; ▶ item labels
6:   Map<Int, Float> roFloatFeatures;
7:   Map<Int, List<Int>> roIdListFeatures;
8:   Map<Int, Map<Int, Float>> nroFloatFeatures;
9:   Map<Int, Map<Int, List<Int>>> nroIdListFeatures;
10: }
11:
12: procedure REQUESTLEVELJOIN(userId, requestId, payload):
13:   currRequestId = getCurrentJoinerRequestId(userId)
14:   joinKey = JoinKey(userId, currRequestId)
15:
16:   if      userHasNewRequestId(requestId, currRequestId)
17:   ||      exceedsUserEngagementThreshold(joinKey)      ||
18:   joinWindowTimeUp(joinKey) then
19:     closeJoinWindow(joinKey) ▶ publish ROO sample
20:     shouldStartNewJoinWindow = true
21:   end if
22:
23:   if shouldStartNewJoinWindow then
24:     rec = RequestJoinRecord() ▶ new join record
25:     rec.impressions.add(payload.item)
26:     setRoFeatures(rec, userId, requestId)
27:     Joiner.set(JoinKey(userId, requestId), rec)
28:   else
29:     rec = Joiner.get(joinKey) ▶ update join record
30:     if rec.impressions.contains(payload.item) then
31:       rec.conversions[payload.item].add(payload.itemLabels)
32:     else
33:       rec.impressions.add(payload.item)
34:       setNroFeatures(rec, payload.item, requestId)
35:     end if
36:     Joiner.set(joinKey, rec)
37:   end if
38: end procedure

```

---

**A Notations**

We summarize key notations used in this paper in Table 5.

**B Request Level Join**

We provide pseudocode to explain request-level joiner and ROO training data. Compared to impression-level joiners such as RecD [45], the design and implementation details set ROO apart.

- The request level joiner supports different triggers to close a join window and publish a ROO training sample. In production, the average close time of the join window is 16 minutes and satisfies the model activity-to-serving requirement.
- The request level joiner keeps a single instance of user-side features, e.g., roIdListFeatures (line 7), for all impressions of the same request in the in-memory feature storage of the joiner. Consequently, it actually uses much less memory than impression-level joiners.
- Due to feature deduplication at the root data source of the training data pipeline, the ROO training data does not require additional data infra feature, nor does it add any data landing latency increase. In contrast, RecD requires extra ETL jobs to cluster samples by session, which adds data landing latency by 30 minutes in production and makes it inherently incompatible for online training and model responsiveness.
- The distinction of RO and NRO features in ROO data is leveraged by feature preprocessing to produce 2D tensor representations as PyTorch KeyedJaggedTensor. In contrast, RecD uses a custom InverseKeyedJaggedTensor to represent impression samples of the same request. The custom tensor transformation requires 1.5x more data preprocessing cpu overhead than ROO for the same GPU unit.

In one recommendation product surface, we replaced a RecD-trained LSR model with ROO-trained HSTU model and ROO training showed a 10% training QPS win even without any overarch optimization. On the same product surface, the ROO trained ESR model showed 50% training QPS win over the RecD version.

**C Ranking Model Architecture**

We illustrate a common DLRM-based ranking model baseline architecture discussed in this paper in Figure 8.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009

Table 5: Table of Notations.

Symbol	Description
$num\_candidate$	number of candidates in a request
$B_{NRO}$	Number of impression-level samples (i.e., number of items) in a training batch.
$B_{RO}$	Number of request-level samples (i.e., number of requests) in a training batch.
$RO_i$	$i$ -th RO-side impression in the user history list.
$NRO_i$	$i$ -th target item in the request.
$a$	Action feature.
$c$	Context features, for instance surface type, timestamp, etc.
$MT$	Multi-task prediction for ranking models.
$u$	User representation aggregated from user history sequence.
$\Phi$	Item served in product.
$X$	Input tensor of sparse feature embeddings.
$B$	Batch size. Note that this is invariant to ROO.
$d_{in}$	Input dimension of sparse feature embeddings.
$n_{in}$	Input number of sparse feature embeddings.
$d_{out}$	Output dimension of sparse feature embeddings.
$n_{out}$	Output number of sparse feature embeddings.

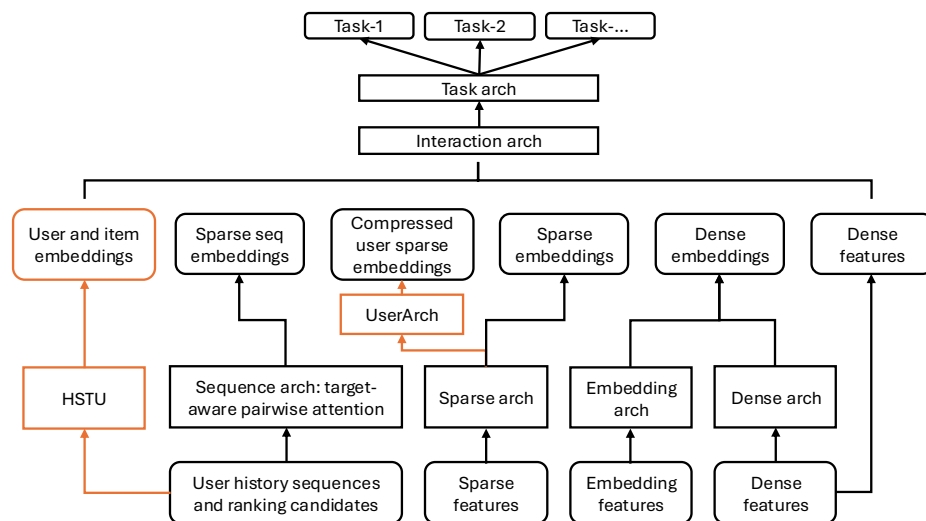


Figure 8: The late stage ranking model architecture. HSTU and user arch (highlighted) introduced request only computations to traditional DLRM architecture.