Make It Long, Keep It Fast: End-to-End 10k-Sequence Modeling at Billion Scale on Douyin

Lin Guan*, Jia-Qi Yang*, Zhishan Zhao*, Beichuan Zhang*, Bo Sun, Xuanyuan Luo, Jinan Ni, Xiaowen Li, Yuhang Qi, Zhifang Fan, Hangyu Wang, Qiwei Chen†, Yi Cheng, Feng Zhang, Xiao Yang
ByteDance
{yangjiaqi.yjq,zhaozhishan,zhangbeichuan.123,zhangyang.2583,xuanyuanluo,nijinan,lixiaowen.911,qiyuhang,wanghangyu.123, chengyi.23,feng.zhang,wuqi.shaw}@bytedance.com, {guanlin.13,fanzhifangfzf,chenqiwei05}@gmail.com

Abstract

Short-video recommenders such as Douyin must exploit extremely long user histories without breaking latency or cost budgets. We present an end-to-end system that scales long-sequence modeling to *10k*-length histories in production. First, we introduce *Stacked Target-to-History Cross Attention (STCA)*, which replaces history self-attention with stacked cross-attention from the target to the history, reducing complexity from quadratic to *linear* in sequence length and enabling efficient end-to-end training. Second, we propose *Request Level Batching (RLB)*, a user-centric batching scheme that aggregates multiple targets for the same user/request to share the user-side encoding, substantially lowering sequence-related storage, communication, and compute without changing the learning objective. Third, we design a *length-extrapolative* training strategy—train on shorter windows, infer on much longer ones—so the model generalizes to *10k* histories without additional training cost. Across offline and online experiments, we observe predictable, monotonic gains as we scale history length and model capacity, mirroring the *scaling law* behavior observed in large language models. Deployed at full traffic on Douyin, our system delivers significant improvements on key engagement metrics while meeting production latency, demonstrating a practical path to scaling end-to-end long-sequence recommendation to the 10k regime.

CCS Concepts

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

Keywords

Recommender systems, long-sequence modeling, scaling laws

ACM Reference Format:
Lin Guan*, Jia-Qi Yang*, Zhishan Zhao*, Beichuan Zhang*, Bo Sun, Xuanyuan Luo, Jinan Ni, Xiaowen Li, Yuhang Qi, Zhifang Fan, Hangyu Wang,, Qiwei Chen†, Yi Cheng, Feng Zhang, Xiao Yang. 2025. Make It Long, Keep It Fast: End-to-End 10k-Sequence Modeling at Billion Scale on Douyin. In . ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

* Equal Contribution. † Corresponding author.

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Deep neural networks have become the backbone of modern recommender systems, powering applications in e-commerce, news feeds, and short-video platforms[**? ? ? ?**]. A key reason is their ability to leverage user behavior sequences, as past interactions provide essential signals for inferring preferences[**? ?**]. In short-video recommendation such as Douyin[**?**], user histories are often thousands of videos long. If effectively utilized, these long sequences can substantially improve ranking performance[**? ? ?**].

The importance of modeling longer sequences relates to the scaling law of deep learning: performance tends to improve predictably with more data, parameters, and compute[**? ?**]. Unlike NLP/CV where scaling often comes from enlarging datasets, recommendation is constrained by user-generated data[**? ?**]. A natural way to expose more information is to use longer histories[**? ? ?**]. However, most systems adopt a two-stage paradigm[**? ? ? ?**]: retrieve a small set similar to the target and feed the truncated sequence to the ranker. While efficient, this breaks end-to-end optimization and discards valuable interactions. *Our empirical results (Fig. 1) show that when the architecture and system support true long-sequence training and inference, model quality scales smoothly with both sequence length and sequence-module capacity, echoing scaling-law behavior in other modalities.*

To truly unlock scaling for large recommendation models, end-to-end training over long sequences must coexist with strict online latency and cost budgets[**? ?**]. Designs must allocate computation selectively, and longer histories amplify storage, communication, and compute in distributed training[**? ? ?**]. To this end, we combine: (i) a *target-centric*, single-query cross-attention model (STCA) with per-layer cost linear in sequence length; (ii) *Request Level Batching (RLB)*, which amortizes user-side encoding across multiple targets within a request and *can be extended to share across multiple requests for the same user/session*, offering further efficiency gains; and (iii) a "train sparsely/infer densely" regimen: we train on sequences with an *average length of around 2k* tokens, yet *extrapolate to 10k* at serving, preserving end-to-end modeling without increasing training compute. Together these components enable predictable gains with increasing length and capacity, *consistent with scaling-law behavior* (Figure 1).

Our contributions. We make three contributions toward practical end-to-end long-sequence recommendation:

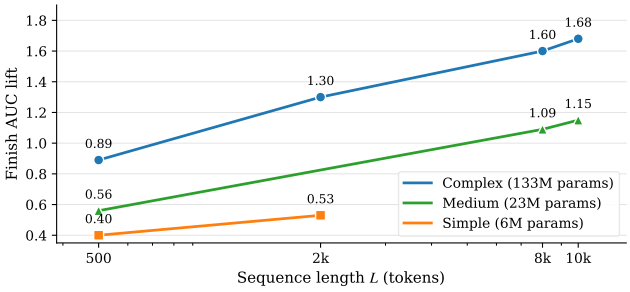

Figure 1: Scaling with sequence length and model capacity. Finish AUC lift (percentage points) for the sequence module (STCA) as we increase user sequence length ($500 \rightarrow 10$k tokens) and sequence-module capacity (Simple: 6M, Medium: 23M, Complex: 133M parameters).

- **Sequence-efficient architecture.** We propose *Stacked Target-to-History Cross Attention* (STCA), which prioritizes cross-attention between the target item and the history while omitting history self-attention, reducing complexity from $O(L^2)$ to $O(L)$ in sequence length L. Stacking multiple layers captures higher-order dependencies via target-conditioned fusion.
- **Scalable training via user-centric batching.** We introduce *Request Level Batching (RLB)*, aggregating samples from the same user to share one user-side encoding across multiple targets. Beyond a single request, RLB naturally extends to *multi-request* sharing for the same user/session, further cutting memory, communication, and computation (up to 8× reduction reported under request-level sharing) while remaining an unbiased estimator of empirical risk.
- **Train sparsely, infer densely.** We adopt a length-extrapolative regimen that trains on sequences with an average length of ∼2k but serves on sequences up to 10k, decoupling training cost from deployment-time context length and realizing long-sequence gains without additional training compute.

These innovations provide a practical framework for scaling along the sequence dimension under production constraints. Deployed in a large-scale short-video platform, our methods deliver significant online gains across multiple business metrics.

2 Background and Notations

Notation. We use the following symbols throughout. The user interaction history is $\mathcal{H} = \{(v_i, a_i)\}_{i=1}^{L}$ of length L, where v_j is the feature vector of the j-th historical video and a_j is its interaction type; t denotes the candidate (target) video to be ranked. Embeddings are $\mathbf{x}_j$ for (v_j, a_j) and $\mathbf{x}_t$ for t. Let d be the embedding dimensionality, r the expansion ratio in the *SwiGLUFFN* width, h the number of attention heads, and M the number of stacked cross-attention layers. The model outputs $\hat{y} \in [0, 1]$, the predicted finish probability for t, and $y \in \{0, 1\}$ is the ground-truth finish label.

We consider large-scale short-video recommendation (e.g., Douyin / TikTok), where the system ranks a set of candidate videos for a user. In practice, the final score may combine multiple objectives (finish, click, etc.); for clarity we focus on predicting the *finish rate*.

User signals are dominated by the interaction history. Let

$$\mathcal{H} = \{(v_1, a_1), \ldots, (v_L, a_L)\},$$

where v_j is the feature vector of the j-th historical video and a_j encodes the interaction type. The candidate to rank is t. Features (IDs, multimodal content, creator attributes, *etc.*) are embedded as $\mathbf{x}_j$ for (v_j, a_j) and $\mathbf{x}_t$ for t.

Given input $(\mathcal{H}, t)$, the ranking model outputs a scalar $\hat{y} \in [0, 1]$, the estimated probability that the user finishes t conditioned on $\mathcal{H}$. This notation will be used throughout when introducing the model and training strategies.

3 Method

We develop an end-to-end framework for long-sequence modeling in short-video ranking that couples a sequence-efficient architecture, a user-centric batching strategy, and a length-extrapolative training regimen as depicted in Figure 2.

3.1 Stacked Target Cross Attention (STCA)

In ranking, the principal signal for predicting the user's response to t arises from *direct* interactions between t and the user's historical behaviors, whereas second-order relations among historical items are comparatively less informative. Yet Transformer-style self-attention over $[t; \mathcal{H}]$ incurs $O(L^2)$ cost in the history length and therefore constrains L.

We make an explicit capacity–cost trade-off: we *de-emphasize* explicit history–history interactions and instead use a *single-query target-to-history cross attention* (STCA). By treating the target as the sole query, the per-layer complexity becomes linear in L ($O(Ldh)$), avoiding length-L key/value intermediates while focusing compute exactly on target–history relevance. This substantial FLOPs and memory reduction enables training and serving with ultra-long histories (e.g., 10k-level) within the same compute envelope, thereby improving *scale-up*: at matched computation, STCA can process much longer context and achieves higher accuracy than $O(L^2)$ self-attention that must operate at shorter L.

3.1.1 Input encoding. Each historical element (v_j, a_j) is embedded as $\mathbf{x}_j \in \mathbb{R}^d$ (video, action-type, position fused), and $X = [\mathbf{x}_1, \ldots, \mathbf{x}_L] \in \mathbb{R}^{L \times d}$. The target video is embedded as $\mathbf{x}_t \in \mathbb{R}^d$. We use a dimension-preserving *SwiGLUFFN* block (SwiGLU + linear projection), followed by LayerNorm:

$$\mathrm{SwiGLUFFN}(\mathbf{x}) = \Big((\mathbf{x}W_u) \odot \big(\mathbf{x}W_v \odot \mathrm{sigmoid}(\mathbf{x}W_v)\big)\Big) W_o, \quad (1)$$

where $W_u, W_v \in \mathbb{R}^{d \times rd}$, $W_o \in \mathbb{R}^{rd \times d}$, $r \geq 1$, and $\odot$ is element-wise product (biases omitted). We apply the block row-wise to matrices and then normalize:

$$\widetilde{X}^{(i)} = \mathrm{LN}\big(\mathrm{SwiGLUFFN}^{(i)}(X)\big) \in \mathbb{R}^{L \times d}, \quad (2)$$

$$\mathbf{q}^{(1)} = \mathrm{LN}\big(\mathrm{SwiGLUFFN}^{(1)}(\mathbf{x}_t)\big) \in \mathbb{R}^{d}, \quad (3)$$

where $\mathrm{LN}(\cdot)$ denotes LayerNorm.

Multi-head target-to-history cross attention. At layer i, given $\mathbf{q}^{(i)}$ and $\widetilde{X}^{(i)}$, we compute h-head cross attention. Let $d_h = d/h$ and

$$W_Q^{(i,r)}, W_K^{(i,r)}, W_V^{(i,r)} \in \mathbb{R}^{d \times d_h}, \quad W_O^{(i)} \in \mathbb{R}^{d \times d}.$$

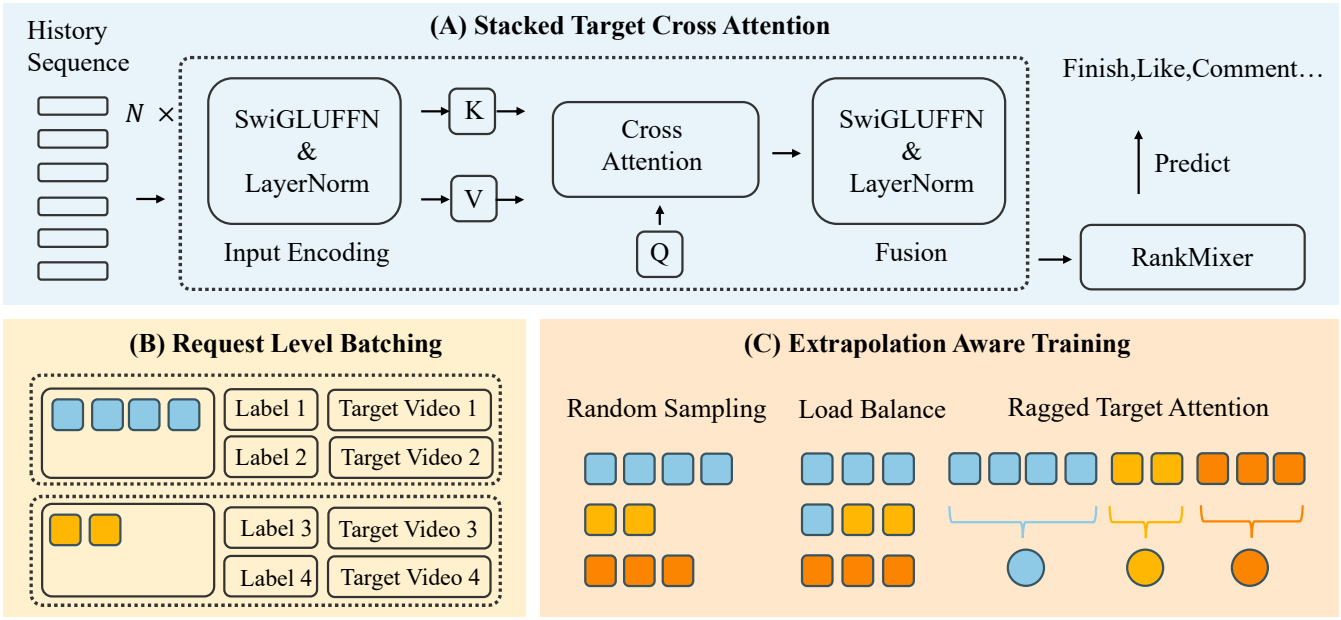

Figure 2: Overview of our long-history ranking stack. (A) Stacked Target Cross Attention: single-query cross attention from the target to the full history with layer-wise fusion, enabling linear scaling in history length and end-to-end optimization. (B) Request Level Batching: compute the user/history encoding once per request and reuse it across multiple targets to reduce bandwidth and compute. (C) Extrapolation Aware Training: sample shorter histories during training and serve longer histories at inference (Train Sparsely, Infer Densely).

For head $r \in \{1, \dots, h\}$,

$$\alpha^{(i,r)} = \mathrm{softmax}\left(\frac{\mathbf{q}^{(i)} W_Q^{(i,r)} \left(\widetilde{X}^{(i)} W_K^{(i,r)}\right)^\top}{\sqrt{d_h}}\right) \in \mathbb{R}^{1\times L}, \quad (4)$$

$$\mathbf{o}^{(i,r)} = \alpha^{(i,r)} \left(\widetilde{X}^{(i)} W_V^{(i,r)}\right) \in \mathbb{R}^{1\times d_h}. \quad (5)$$

Concatenate heads and project:

$$\mathbf{o}^{(i)} = \left[\mathbf{o}^{(i,1)} \parallel \cdots \parallel \mathbf{o}^{(i,h)}\right] W_O^{(i)} \in \mathbb{R}^d. \quad (6)$$

Per layer, cost is $O(Ldh)$ for a single target query—linear in L—versus $O(L^2 dh)$ for self-attention over $[t; \mathcal{H}]$.

3.1.2 Stacking and target-conditioned fusion. We *stack* M cross-attention layers and update the query via target-conditioned fusion. To keep dimensions consistent (input d), we compress the growing concatenation with a learnable projection:

$$\mathbf{q}^{(i+1)} = \mathrm{SwiGLUFFN}^{(i+1)}\left(\left[\mathbf{o}^{(1)} \parallel \cdots \parallel \mathbf{o}^{(i)} \parallel \mathbf{x}_t\right] W_C^{(i+1)}\right), \quad (7)$$

where $W_C^{(i+1)} \in \mathbb{R}^{(i+1)d\times d}$. After M layers, we collect the summaries as

$$\mathbf{Z}_{\mathcal{H}} = \begin{bmatrix} \mathbf{o}^{(1)} \\ \vdots \\ \mathbf{o}^{(M)} \end{bmatrix} \in \mathbb{R}^{M\times d}. \quad (8)$$

3.1.3 Prediction head and objective. We form the final target-aware token by compressing all layer summaries with the target:

$$\mathbf{z} = \mathrm{SwiGLUFFN}\left(\left[\mathbf{o}^{(1)} \parallel \cdots \parallel \mathbf{o}^{(M)} \parallel \mathbf{x}_t\right] W_Z\right), \quad W_Z \in \mathbb{R}^{(M+1)d\times d}. \quad (9)$$

Let

$$\mathcal{X}_{\mathrm{mix}} = \mathrm{concat}\left(\mathbf{z}, \{\mathbf{u}_k\}_{k=1}^K, \{\mathbf{c}_\ell\}_{\ell=1}^C\right),$$

where $\{\mathbf{u}_k\}$ are auxiliary *user-side* tokens (e.g., profile/context features) and $\{\mathbf{c}_\ell\}$ are *candidate-side* tokens associated with the same target t (e.g., content/creator modalities). Then RankMixer[?] produces

$$\mathbf{h} = \mathrm{RankMixer}\left(\mathcal{X}_{\mathrm{mix}}; \Theta\right), \qquad \hat{y} = \mathrm{sigmoid}\left(\mathbf{w}^\top \mathbf{h} + b\right). \quad (10)$$

We optimize binary cross-entropy:

$$\mathcal{L}_{\mathrm{BCE}} = -y \log \hat{y} - (1-y) \log(1-\hat{y}). \quad (11)$$

3.1.4 Computation optimization for single-query cross attention. With exactly one query per layer, let $X \in \mathbb{R}^{L\times d}$, $q \in \mathbb{R}^{1\times d}$, and $d_h{=}d/h$. The standard form

$$\mathrm{Attn}(q, X) = \mathrm{softmax}\left(\frac{(qW_Q)(XW_K)^\top}{\sqrt{d_h}}\right) \cdot (XW_V), \quad (12)$$

projects all L tokens twice. Reordering removes the length-L projections:

$$u = (qW_Q) W_K^\top \in \mathbb{R}^{1\times d}, \quad \alpha = \mathrm{softmax}\left(\frac{uX^\top}{\sqrt{d_h}}\right) \in \mathbb{R}^{1\times L},$$

where $o = (\alpha X)\, W_V \in \mathbb{R}^{1\times d_h}$, $W_Q, W_K, W_V \in \mathbb{R}^{d\times d_h}$. Equivalently,

$$\mathrm{Attn}(q, X) = \Big(\underbrace{\mathrm{softmax}\big(\tfrac{((qW_Q)W_K^\top)X^\top}{\sqrt{d_h}}\big)}_{\alpha\in\mathbb{R}^{1\times L}} X \Big) W_V = (\alpha X) W_V. \tag{13}$$

Per head the cost is $O(d\, d_h) + O(Ld) + O(d\, d_h)$; across h heads it totals $O(Ldh + d^2)$ while avoiding any $L\times d_h$ intermediates.

The *naïve* path spends $\approx 4Ldd_h$ FLOPs on (XW_K, XW_V) and materializes two $L\times d_h$ tensors per head; our *reordered* path replaces them with a single weighted reduction αX of cost $2Ld$ FLOPs and no $L\times d_h$ intermediates. Thus the length-dependent FLOPs shrink by a factor of $\approx 2d_h = 2d/h$; e.g., with d=256, h=8 (d_h=32) this yields $\sim$64× fewer L-dependent FLOPs.

3.2 Request Level Batching (RLB)

STCA makes the *per-target* sequence cost linear in history length ($O(L)$), which enables long contexts. However, in real logs each user typically contributes multiple targets within the same request/session. If we still train on independent triplets (u, v, y), the same long history $\mathcal{H}$ is serialized, transferred (CPU→GPU), and re-encoded *repeatedly*, so storage and bandwidth—not FLOPs—become the bottleneck as L grows. RLB is therefore the *system-side complement* to STCA: it removes redundant user-side work so that the linear-in-L encoder can scale to multi-k sequences under production budgets.

3.2.1 Basic idea and unbiasedness. RLB aggregates m samples from the same user into a user micro-batch $\mathcal{B}_u = \{(u, v_k, y_k)\}_{k=1}^m$. Let $\Phi_{\text{user}}(\mathcal{H})$ denote the user/history path (shared across targets). With RLB we compute $\Phi_{\text{user}}(\mathcal{H})$ *once* and reuse it for all $\{v_k\}_{k=1}^m$. The per-user loss and overall objective are

$$\mathcal{L}_u = \frac{1}{m}\sum_{k=1}^{m} \mathcal{L}_{\text{BCE}}\big(\hat{y}(u, v_k), y_k\big), \qquad \mathcal{L} = \frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}} \mathcal{L}_u. \tag{14}$$

Unbiasedness. Writing the conventional objective as a user-average of instance-averages shows that replacing the inner average by a without-replacement average over m targets leaves the expectation unchanged (linearity of expectation). Hence Eq. (14) is an unbiased estimator of empirical risk; RLB changes *only* the computation layout, not the learning objective.

3.2.2 Systems view. RLB turns repeated user-side work into a *compute-once, reuse-m-times* pattern, yielding concrete gains along the IO – memory – kernel–distributed stack:

- **Host↔Device I/O.** Let the per-request payload split into user / history bytes U and per-target bytes T. Triplet batching transfers $m(U+T)$, whereas RLB transfers $U+mT$, saving $\approx (1 - \frac{1}{m})U$ bytes. For m=8 this idealized saving is 87.5% on the user payload, and our end-to-end measurements (including non-sequence features) show 77–84% bandwidth reduction.
- **Peak memory and activations.** The history encoder's intermediate states ("K/V-like" tensors from $\widetilde{X}^{(i)}$) are *shared* across m targets instead of replicated, shrinking the per-target footprint from $O(L)$ to $O(L/m)$. This lowers peak activation memory and enables much longer L under the same GPU RAM; empirically, the maximum trainable length increases by $\sim 8\times$ at fixed hardware.
- **Kernel efficiency.** Reusing user encodings lets us batch target-side matmuls into larger **GEMM**s with higher occupancy and fewer kernel launches. Combined with ragged/compacted sequences, this reduces padding, increases arithmetic intensity, and improves samples/s; we observe a 2.2× single-GPU throughput speedup.
- **Distributed training.** Fewer distinct user encodings per step reduce gradient aggregation and PS/allreduce traffic. In practice, Parameter-Server CPU usage and inter-module bandwidth both drop by **50**% while maintaining accuracy).

Overall, RLB removes redundant movement and recomputation of $\mathcal{H}$, so the linear-in-L STCA encoder remains the dominant—and now amortized—work, making long-sequence training practical at production scale.

3.3 Extrapolation: Train Sparsely, Infer Densely

STCA makes the *per-target* sequence cost linear in L and RLB amortizes the user path across m targets, together enabling long-context *serving* under strict latency and bandwidth budgets. However, *training* on uniformly long histories still scales linearly with the number of tokens processed per batch, quickly exhausting memory and throughput as L grows. To decouple training cost from deployment-time context length, we introduce a length–extrapolation regimen that *trains sparsely* (low average tokens per batch) yet *infers densely* (long histories at test time). Throughout this section we fix the deployment target at $L_{\text{infer}} = \mathbf{10k}$ and the training average at $L_{\text{train}}^{\text{avg}} = \mathbf{2k}$, yielding an extrapolation ratio $\rho_{\text{extra}} = \frac{L_{\text{infer}}}{L_{\text{train}}^{\text{avg}}} = \mathbf{5}$.

We therefore follow the *Stochastic Length* (SL) training paradigm during training, each input sequence is randomly truncated to a length $L_{\text{train}} \in [L_{\text{train}}^{\min}, L_{\text{train}}^{\max}]$, where $L_{\text{train}}^{\max} \le L_{\text{infer}}$, and L_{infer} is the maximum sequence length used at inference. We quantify the resulting efficiency via *sequence sparsity* (SS), defined as

$$\text{SS} = \frac{\mathbb{E}[L_{\text{train}}]}{L_{\text{train}}^{\max}} = \frac{L_{\text{train}}^{\text{avg}}}{L_{\text{train}}^{\max}}, \tag{15}$$

which reflects the average computational cost relative to the maximum training length.

Under the Stack Cross-Attention (STCA) architecture, this stochastic training strategy raises two key challenges:

(1) **Batch-Level Load Balancing**: Variable-length sequences cause GPU workload imbalance, as batch processing time is dictated by the longest sequence—undermining potential FLOPs savings.
(2) **Subsequence Selection**: An effective selection strategy must minimize training sequence length (i.e., maximize SS) without degrading model accuracy.

3.3.1 Subsequence Selection Strategy. The subsequence selection process involves two steps: (1) sampling a stochastic training length L_{train}, and (2) selecting which elements from the full history to retain.

Stochastic Length Sampling. We sample a normalized length ratio s from a Beta distribution, chosen for its bounded support and flexibility. Specifically,

$$s \sim \text{Beta}(\alpha, \beta), \quad s \in (0, 1),$$

and compute the raw training length as

$$L_{\text{train}}^{\text{raw}} = L_{\text{train}}^{\min} + s \cdot (L_{\text{train}}^{\max} - L_{\text{train}}^{\min}). \tag{16}$$

To align with hardware acceleration requirements (e.g., tensor core alignment), $L_{\text{train}}^{\text{raw}}$ is rounded to the nearest multiple of 8 to obtain L_{train}.

Given a target average training length $L_{\text{train}}^{\text{avg}}$, the expectation constraint

$$\mathbb{E}[L_{\text{train}}^{\text{raw}}] = L_{\text{train}}^{\min} + (L_{\text{train}}^{\max} - L_{\text{train}}^{\min}) \cdot \frac{\alpha}{\alpha + \beta} = L_{\text{train}}^{\text{avg}}$$

implies

$$\beta = \alpha \cdot \frac{L_{\text{train}}^{\max} - L_{\text{train}}^{\text{avg}}}{L_{\text{train}}^{\text{avg}} - L_{\text{train}}^{\min}}. \tag{17}$$

Hence, s is sampled from

$$s \sim \text{Beta}\left(\alpha,\ \alpha \cdot \frac{L_{\text{train}}^{\max} - L_{\text{train}}^{\text{avg}}}{L_{\text{train}}^{\text{avg}} - L_{\text{train}}^{\min}}\right).$$

The average length $L_{\text{train}}^{\text{avg}}$ directly controls sequence sparsity via $\text{SS} = L_{\text{train}}^{\text{avg}} / L_{\text{train}}^{\max}$. *Experimental results show that adjusting the shape parameter $\alpha > 0$ to make the beta distribution as close as possible to a U-shaped distribution yields better performance.*

Element Selection Policy. Given L_{train}, we select the corresponding number of items from the user's full history (truncated to L_{infer} at inference). *Empirical results show that retaining the most recent L_{train} interactions—i.e., the temporal suffix—consistently yields optimal accuracy.*

3.3.2 Batch-Level Load Balancing. To mitigate GPU load imbalance caused by variable-length sequences, we introduce a *batch-level load-balancing operator* that maintains a fixed total token budget of $B \cdot L_{\text{train}}^{\text{avg}}$ per batch (with batch size B), while preserving the stochastic training distribution. This operator works in two phases:

(1) **Global Length Allocation**: Sequences are stochastically truncated such that the total sequence length across the batch remains close to $B \cdot L_{\text{train}}^{\text{avg}}$. This ensures balanced GPU utilization and allows the computational graph to be built using the average—not maximum—sequence length.
(2) **Sequence Compaction**: Sequences are compacted so that each has length exactly $L_{\text{train}}^{\text{avg}}$. Excess tokens from longer sequences are redistributed to shorter ones, eliminating padding and reducing memory and I/O overhead (see Figure 2).

To efficiently process these compacted variable-length sequences, we design a *Ragged Target Attention* mechanism, supported by a highly optimized **GEMM** kernel [**?**]. Unlike standard cross-attention, our method avoids padding by using an auxiliary `index` tensor that marks segment boundaries. Keys (K) and values (V) are flattened into 2D matrices, and each query (Q) attends only to its corresponding segments in K and V.

Table 1: Offline results on Douyin. All values are in %.

Model	Finish		Skip		Head	
	ΔAUC↑	ΔNLL↓	ΔAUC↑	ΔNLL↓	ΔAUC↑	ΔNLL↓
Baseline	0.00	0.00	0.00	0.00	0.00	0.00
DIN	+0.19	−0.17	+0.23	−0.18	+0.19	−0.21
Trans	+0.25	−0.46	+0.27	−0.36	+0.38	−0.27
HSTU	+0.31	−0.86	+0.52	−0.62	+0.36	−0.43
Ours	**+0.49**	**−1.16**	**+0.71**	**−1.14**	**+0.39**	**−1.41**

3.4 Summary

STCA provides a target-centric, stackable cross-attention mechanism that scales linearly with sequence length; RLB shares user computation across multiple targets to curb I/O and compute overhead; and the extrapolation regimen trains on predominantly short sequences while realizing long-sequence gains at inference. Together, these components deliver an end-to-end solution for long-sequence ranking under strict latency budgets.

4 Experiments

4.1 Offline Comparison on Douyin

4.1.1 Setup. We evaluate sequence encoders on the *Douyin* offline dataset for three objectives—**finish** (completion), **skip** (quick skip), and **head** (creator-page click)—reporting *AUC* (higher is better) and *NLL* (lower is better). To keep the comparison conservative, *all baselines are augmented with TWIN(10k)[**?**]* (a retrieval-style block built from a 10k-length behavior search), whereas *our method removes TWIN(10k)* and relies purely on end-to-end long-history modeling.[1] Importantly, we compare under *roughly matched compute*: per-sample sequence FLOPs and step-time are aligned across methods. For quadratic-cost encoders (Transformer, HSTU), we *reduce depth/width* to keep their total compute comparable to STCA, ensuring fair comparisons.

We compare *Single-layer target attention*, *DIN*, *Transformer* (self-attention), and *HSTU* against our **STCA** (stacked target→history cross attention) with **RLB** and **Train Sparsely / Infer Densely** ("Ext"). Table cells report *% changes* vs. a production baseline: *RankMixer + Single-layer Target Attention + TWIN(10k)*; positive ΔAUC and negative ΔNLL are better.

4.1.2 Results and discussion. In Table 1, **STCA+RLB+Ext** achieves the strongest improvements on all tasks despite using *no* TWIN(10k): **+0.49**/**-1.16** (finish), **+0.71**/**-1.14** (skip), and **+0.39**/**-1.41** (head). Baselines also improve under the matched-compute setting—DIN up to **+0.23**/**-0.21**, Transformer up to **+0.38**/**-0.46**, HSTU up to **+0.52**/**-0.86**—yet our method consistently delivers the largest lifts, especially in NLL (e.g., head **-1.41**).

The gains align with our design: retrieval features pre-select and lose information and end-to-end gradients; **STCA** performs *exact* softmax attention over the full history with $O(L)$ cost per target, **RLB** removes redundant user encodings across targets, and **Train Sparsely / Infer Densely** exposes a calibrated tail of long

[1] All models share identical non-sequence features, optimizers, and data splits; only the sequence encoder and the use of TWIN(10k) differ.

contexts to enable multi-thousand-token inference without full-length training.

Takeaway. Under identical non-sequence features and *matched compute*, **STCA+RLB+Ext** improves both ranking and calibration while simplifying the stack (no TWIN(10k)), providing a practical path to accurate, deployable long-sequence modeling on Douyin.

4.2 Ablation: Accuracy vs. Model Complexity

Setup and metrics. We evaluate the end-to-end stack **STCA → RankMixer** with **RLB** (m=8) and single-query cross-attention (Sec. 3). Unless noted, training and evaluation use L=512; we report *finish AUC* lift (%) over a strong **RankMixer**-only baseline trained with identical non-sequence features and optimization.

Where the gains come from. Ablations in Table 2 indicate that adding a token-wise FFN on the sequence path and deepening STCA from 2L→4L provides the largest single boost (+0.18%). Upgrading FFNs to *SwiGLU* yields a further +0.11%. Enlarging sparse ID embeddings (+0.08%) and introducing time-delta side information (+0.08%) both contribute meaningfully. Increasing attention heads from 8 → 16 offers a modest but positive gain (+0.05%). Finally, the *query fusion* mechanism (Eq. 7) adds +0.06% by reinjecting lower-layer summaries into higher layers for target-conditioned reasoning.

Compute–quality scaling. Figure 3 couples compute (FLOPs) with quality (NLL) under matched depth/width. Two points stand out:

- **Linear vs. quadratic in** L**.** STCA's sequence-side FLOPs grow linearly; Transformer's grow quadratically. From L=500 → 10k, STCA rises 1.06 → 21.06 GFLOPs (∼19.9×), while Transformer rises 2.08 → 236.26 GFLOPs (∼113.6×).
- **Better frontier at long** L**.** At similar NLL (≈0.396), STCA runs at L=10k with 21.06 GFLOPs, whereas Transformer needs L=8k and 156.24 GFLOPs (∼7.4× higher).

These results align with our broader findings: **STCA** removes history self-attention and focuses compute on target↔history relevance (near-linear scaling), **RLB** amortizes the user path (and extends across requests/sessions), and **train sparsely / infer densely** trains at ∼2k but serves up to 10k. Combined, they provide a practical scale-up path: STCA sustains longer contexts at similar or lower FLOPs while preserving or improving ranking quality.

4.3 Request-Level Batching (RLB)

Setup. RLB realizes user-centric batching at the *request* boundary: for each request containing one user history and multiple targets, we transmit and encode the *shared* user/context payload once, reuse it for all targets, and aggregate gradients at the request level before synchronization. Unless otherwise noted, the stack is STCA → RankMixer with RLB enabled (Sec. 3), trained on the same data, optimizer, and batch size as the point-wise baseline. Reported bandwidth figures include *all* non-sequence features (profile/context, content, creator, etc.), i.e., end-to-end payloads.

Bandwidth footprint. RLB substantially cuts inter-module traffic by amortizing the $O(L)$ user/history payload across targets: the measured reduction is **77%** at history length L=512 and **84%** at L=2k (both including all existing features). The larger saving at longer

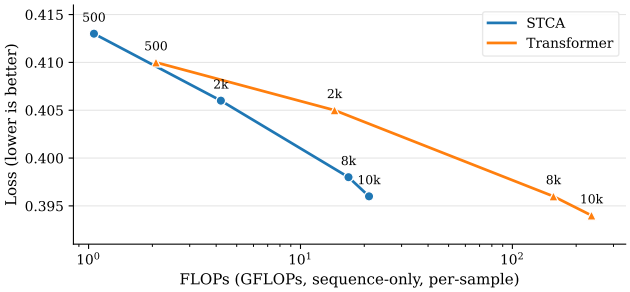

Figure 3: Compute–quality scaling of STCA vs Transformer. X-axis: per-sample *sequence-only* forward FLOPs (log scale). Y-axis: NLL (lower is better). Markers show $L \in \{500, 2k, 8k, 10k\}$**. Both use 4 layers with** d=256, h=8, r=4**.**

L follows directly from avoiding per-target retransmission of the same history.

Throughput and scalability. Relative to a point-wise baseline (1×), RLB delivers a **2.2**× end-to-end training throughput gain. With further kernel optimizations—specialized batched matmul for the reordered single-query attention, high-throughput *SwiGLU*, and optimized LayerNorm—the speedup reaches **5.1**×. Under the same infrastructure, amortizing per-target activations and payload also raises the *maximum trainable sequence length* by roughly **8**× (workloads that were memory/IO bound at length L become trainable at ≈ $8L$).

Parameter server and inter-module costs. At synchronization and feature-serving boundaries, RLB further lowers overhead: we observe a **50%** reduction in Parameter Server (PS) CPU usage during training and a **50%** reduction in data↔training communication bandwidth; training-side PS CPU load likewise drops by **50%**.

Discussion. These gains complement STCA's architectural efficiency. STCA removes the quadratic dependence on history length via single-query cross attention ($O(L)$), while RLB eliminates *redundant* movement and re-encoding of the same user/history across targets. Together they yield higher samples-per-second, lower PS contention, and much larger feasible sequence lengths, enabling long-context models within fixed latency and hardware budgets.

4.4 Extrapolation: Train Sparsely, Infer Densely

Experimental Setup. We evaluate the proposed extrapolation framework using STCA encoder with single-query optimization. The user token **z** is fed into *RankMixer* (Sec. 3) with RLT training (m=8). During training, history length L_{train} is randomized following Sec. 3.3.1. All results report finish AUC improvements versus a fixed 2k-token baseline. Unless otherwise specified, **inference uses** $L_{\textbf{infer}}$ = **10**k.

The impact of the maximum length (Table 3). Progressively increasing $L_{\text{train}}^{\max}$ from 2k to 10k yields a corresponding improvement in the AUC lift, which rises from a marginal +0.03% to a more substantial +0.21%. These results confirm that to ensure robust generalization to long contexts, the training regime must expose the

Table 2: Ablation of the 4L complex STCA at 512 tokens.

Component	Δ AUC	Notes
Enlarge sparse ID embedding: 128 → 320	+0.08%	Video-ID embedding width. Use smaller init range and LR to avoid instability.
Add sequence-side FFN; depth 2 → 4	+0.18%	Token-wise FFN on the history path; then double STCA depth from 2L to 4L.
FFN → SwiGLU	+0.11%	Upgrade FFNs to SwiGLU; widen hidden by ×2.
Attention heads: 8 → 16	+0.05%	More heads improve target-conditioned selectivity with modest cost.
Query fusion	+0.06%	At layer i+1, concatenate $[\mathbf{o}^{(1)} \dots \mathbf{o}^{(i)}, \mathbf{x}_t]$ then apply SwiGLU (Eq. 7).
Time-delta side info	+0.08%	Add per-token feature: request time minus item timestamp (recency prior).

Table 3: Effect of maximum training length.

Metric	$L^{\max}_{\text{train}} = 2\text{k}$	$L^{\max}_{\text{train}} = 4\text{k}$	$L^{\max}_{\text{train}} = 10\text{k}$
Finish AUC lift	+0.03%	+0.09%	+0.21%

Table 4: Effect of average training length.

Metric	$L^{\text{avg}}_{\text{train}} = 1.0\text{k}$	$L^{\text{avg}}_{\text{train}} = 2.0\text{k}$	$L^{\text{avg}}_{\text{train}} = 2.5\text{k}$
Finish AUC lift	+0.09%	+0.21%	+0.22%
Sequence Sparsity	10.0%	20.0%	25.0%

Table 5: Beta distribution shape parameter analysis.

Metric	$\alpha = 0.02$	$\alpha = 0.5$	$\alpha = 10$
Finish AUC lift	+0.21%	+0.11%	+0.08%
Shape	U-shaped	Decreasing	Skewed

model to sequence lengths that closely match or approximate those encountered during inference.

Sequence Sparsity Optimization. Table 4 shows the efficiency-accuracy trade-off. Increasing $L^{\text{avg}}_{\text{train}}$ from 1.0k to 2.5k improves AUC lift from +0.09% to +0.22% while reducing sequence sparsity from 10% to 25%. The diminishing returns beyond $\bar{L} = 2.0\text{k}$ confirm that SS $\approx$ 20% provides the optimal balance. Compared to SL in HSTU (SS=57.6%) [**?**], we achieve better computational efficiency.

Subsequence Selection Strategy Validation. Retaining the most recent interactions (greedy) achieves +0.21% AUC improvement, while random sampling yields no gain, strongly supporting the importance of temporal locality.

Beta Distribution Shape Analysis. Table 5 validates our distribution design: the U-shaped distribution ($\alpha = 0.02$) achieves superior performance (+0.21%) compared to decreasing (+0.11%) and skewed (+0.08%) distributions, confirming that bimodal sampling optimizes the training curriculum.

Efficiency-Accuracy Trade-off. Our approach achieves +0.23% offline AUC improvement at 10k inference, capturing ∼ 80% of the full 10k training gain (+0.30%) at one-third computational cost. Online A/B tests confirm production viability (+0.17% finish AUC).

Discussion. The results comprehensively validate our methodology: (1) Stochastic training with U-shaped Beta distribution enables effective extrapolation; (2) Temporal suffix selection preserves sequential patterns; (3) Load-balancing ensures training efficiency. Under our default setting of $L^{\text{avg}}_{\text{train}} = \mathbf{2k}$, $L^{\max}_{\text{train}} = \mathbf{10k}$, and $L_{\text{infer}} = \mathbf{10k}$ ($\rho_{\text{extra}} = \mathbf{5}$). Compared to HSTU's SL (SS=57.6%), we reduce SS to 20% while maintaining accuracy, demonstrating superior computational efficiency.

4.5 Online A/B Results

Setup. We deployed **STCA + RLB + Extrapolation** (Sec. 3) for one month on *Douyin* and *Douyin Lite*, replacing TWIN(10k)-augmented retrieval features with our single-query target↦history encoder while keeping all other components unchanged. We report percentage lifts over control on **30-day Activeness**, **App Stay Time**, **Finish**, **Comment**, and **Like**, overall and by user-activity segment (Table 6).

Findings. Our method delivers *consistent and sizable* online gains across both products and all segments. Overall, **Finish** and **App Stay Time** improve together, and interactive signals (**Comment**, **Like**) rise markedly. Gains are strongest for *low/medium-activity* users, indicating better personalization under sparse/noisy recent behavior, while **30-day Activeness** also increases modestly yet consistently. The improvements stem from (i) **STCA** focusing compute on exact target↦history interactions over long contexts, (ii) **RLB** amortizing user encoding to keep serving costs within budget, and (iii) **Train Sparsely/Infer Densely** exposing a calibrated tail of long sequences during training. Together, these make end-to-end long-history modeling both accurate and deployable at scale.

5 Related Work and Discussion

5.1 Modeling User Behavior Sequences

Early systems modeled sequences with non-deep methods such as item-to-item CF and Markov transitions [**? ?**]. Deep learning then brought session-based RNNs and industrial two-stage stacks (e.g., YouTube) [**? ?**]. Attention/Transformer models became dominant: DIN/DIEN condition history on the candidate to emphasize target-aware selection [**? ?**], while SASRec/BERT4Rec provide strong self-attention baselines that model temporal dependencies through bidirectional or unidirectional attention [**? ?**]; BST further demonstrated online deployment in an industrial feed-ranking setting [**?**]. In parallel, multi-interest modeling explicitly captures diverse user intents and has become widely used in large-scale platforms [**?**].

Table 6: One-month online A/B lifts (%) over control on Douyin and Douyin Lite.

	Douyin					Douyin Lite				
Segment	**30-day Act.**	**Stay Time**	**Finish**	**Comment**	**Like**	**30-day Act.**	**Stay Time**	**Finish**	**Comment**	**Like**
Low	0.3659%	2.0070%	5.4987%	2.6848%	2.4367%	0.3575%	1.3888%	6.2808%	6.3385%	3.1623%
Medium	0.3788%	1.7065%	5.2062%	0.7028%	2.3779%	0.2832%	1.3172%	5.9688%	5.7934%	5.6372%
High	0.1396%	1.1262%	3.7973%	1.4012%	2.1455%	0.1604%	0.9872%	4.9922%	2.8496%	3.1752%
All Users	**0.1161%**	**0.9266%**	**3.3454%**	**1.5678%**	**1.8282%**	**0.1281%**	**0.8467%**	**4.2275%**	**2.6167%**	**2.3828%**

Recent production work emphasizes *long* histories and system compatibility: user-representation approaches (e.g., PinnerFormer) compress long-term behaviors into condensed memories, and real-time/batch fusion (e.g., TransAct) merges streaming signals with precomputed features for latency-aware serving [**? ? ?**]. To scale further, two-stage paradigms retrieve a target-relevant slice before fine modeling (SIM, UBR4CTR) [**? ?**], or engineer sparse memories/hierarchies (SAMN, TWIN/TWIN-V2) to bound the cost of processing very long logs [**? ? ?**]. These designs typically trade exact end-to-end gradients over the full sequence for efficiency via truncation, retrieval, or summarization. *Compared with these, our STCA performs single-query target→history cross attention end-to-end over the full sequence, achieving linear complexity in L and avoiding retrieval/truncation.* This yields a practical path to *very* long histories in production while maintaining differentiability through all observed interactions.

5.2 Organizing Training Samples

Classic training formulations include pointwise/pairwise learning (e.g., BPR) [**?**], large-batch CTR pipelines (DLRM, production systems) [**? ?**], and session-parallel batching for sequences [**?**]. At industrial scale, training is often I/O/memory bound rather than FLOP bound [**?**], motivating truncation or retrieval-first construction [**? ?**] and engineering for ultra-long contexts [**? ? ?**]. Modern stacks (e.g., TorchRec) provide jagged tensors and sharding to better utilize hardware and manage sparsity [**?**], and production systems mix real-time/batch signals to balance freshness and stability [**? ?**]. *Against this backdrop, our Request Level Batching (RLB)* reorganizes data at the user/request granularity: compute the user/history encoder once per request and reuse it across m targets, cutting the per-target user-path cost from $O(L)$ to $\approx O(L/m)$ while keeping the empirical-risk objective unbiased. Unlike truncation/retrieval, RLB preserves the full history and amortizes movement and recomputation, directly addressing the dominant I/O and activation bottlenecks at scale [**?**]. It is orthogonal to retrieval, kernel-level speedups, and sharding (e.g., jagged tensors in [**?**]) and aligns with request-level grouping in production, improving bandwidth, peak memory, and kernel efficiency without changing the loss.

5.3 Length Extrapolation

LLMs show that *"train sparsely, infer densely"* is feasible via positional designs (ALiBi, RoPE) and memory/attention patterns (Transformer - XL, Longformer, BigBird) that enable generalization beyond the training window [**? ? ? ? ?**]. *Our approach adapts this spirit to recommendation* by (i) replacing quadratic history self-attention with exact single-query cross attention (STCA), and (ii) reorganizing training via RLB to amortize user encoding. Unlike retrieval pipelines [**? ? ? ?**], we keep end-to-end gradients over the full history; unlike kernel/sparsity methods [**? ?**], we directly match the target-over-history interaction while aligning with production constraints [**? ? ? ?**].

Concretely, we use stochastic-length sampling during training to keep the *average* sequence length short while exposing a calibrated tail of longer contexts, and we serve at much longer histories at inference. This regimen leverages STCA's linear-in-L compute and RLB's amortization to make extrapolation practical in production, without truncation or surrogate memories.

6 Conclusion

We present an end-to-end recipe for long-sequence recommendation that is simultaneously architectural, system, and training efficient. Architecturally, STCA replaces history self-attention with single-query target→history cross-attention, yielding linear ($O(L)$) complexity in sequence length. System-wise, RLB reuses the user-side encoding at the request level, eliminating redundant transfers and computation. Training-wise, a "train sparsely / infer densely" regimen enables dense inference on long histories at modest training cost. Offline and online experiments show stable, substantial gains and scaling-law–like improvements as sequence length and sequence-module capacity grow. Key findings include: single-query attention is sufficient for short-video ranking while preserving $O(L)$ cost; stochastic-length sampling with a small-α Beta achieves roughly 80% of the 10k-window benefit at about one-third the training cost; parameter budget is best spent on the sequence path via SwiGLU, cross-layer query fusion, and time-delta features—whose impact increases with longer contexts; and system levers are decisive for deployability—at (L=2k), RLB reduces end-to-end bandwidth by up to 84%, halves PS CPU usage, delivers 2.2× throughput, and expands the maximum trainable sequence length by 8×.

In practice, a strong operating point couples a 4-layer STCA (with SwiGLU and query fusion) with time-delta features, RLB, and stochastic-length training (small α, *training mean* $\approx$ 2k) and *inference length* 10k (a 5× extrapolation ratio). This configuration provides monotonic improvements with both sequence length and sequence-module capacity while keeping training and serving within budget, offering a production-ready path to accurate long-sequence recommendation.

A Additional Implementation and System Details

A.1 Training Curriculum and Modular Integration

Directly training at long contexts can be unstable (e.g., L=2048). We therefore adopt a simple curriculum: (i) pretrain at L=512 to establish robust token-level filters and attention patterns; (ii) continue training at L=2048. During architecture iteration, we prototype at L=512 for faster convergence and lower resource use. For integration into a larger production stack, we first train the sequence sub-network to convergence, load its parameters into the composite model, and finally perform joint finetuning. This staging prevents vanishing gradients along the sequence path when the rest of the stack is already strong.

A.2 Generalization and Robustness of STCA

STCA generalizes well across sequence lengths because each history token is processed independently *conditioned on the target*, making the architecture naturally length-agnostic; stacking enables information aggregation to scale smoothly as L increases. By filtering history through the target at every layer, STCA is less sensitive to irrelevant or noisy behaviors common in real logs. This improves robustness to variable-length sequences and heterogeneous user patterns—properties that are crucial for industrial deployment at scale.

A.3 RLB vs. Common Sample Organizations

Instance-wise (triplet) batching re-encodes the full history $\mathcal{H}$ for every (u, v, y), driving dataset size and I/O as $O(mL)$. *Padding/bucketing by length* mitigates kernel divergence but still repeats user encoding. *History truncation / retrieval-first* shortens or selects subsequences, reducing cost but discarding information and breaking end-to-end gradients through the full history. *Embedding caching / clustering* compresses histories at the cost of approximation error. In contrast, **Request-Level Batching (RLB)** is lossless and end-to-end: it preserves the full $\mathcal{H}$, keeps the objective intact, and lowers the *per-target* user-path complexity from $O(L)$ to roughly $O(L/m)$. In practice we set m=8, yielding about 8× lower user-side bandwidth and encoder compute, higher GPU utilization, and longer feasible L.

A.4 Relation to Attention Optimizers and Fused Kernels

GQA/MQA reduce the number of distinct K/V projections to save memory/bandwidth, but their scoring over a length-L sequence remains $O(L^2 d)$. IO-efficient kernels (e.g., FlashAttention) reduce memory traffic yet keep quadratic compute; linear/low-rank variants reach $O(Ld)$ via approximations. **STCA** instead removes history self-interactions entirely and performs *exact* single-query target→history attention with $O(Ldh)$ per layer (further benefiting from the matmul reordering in Eq. (13)), while the downstream RankMixer operates on a small, length-independent token set. **RLB** complements this by amortizing the user path across m targets, effectively turning $O(L)$ into $O(L/m)$ per target. These techniques are compatible with head sharing and fused kernels, but our dominant savings arise from (i) architectural removal of the quadratic term and (ii) system-level amortization—achieved *without* retrieval or truncation.