LONGER HISTORY OR THE RIGHT HISTORY? CANDIDATE-FOCAL HISTORY COMPRESSION FOR LONG-SEQUENCE RECOMMENDATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-sequence recommendation seeks to improve ranking by leveraging longer user histories, under the premise that more historical interactions provide richer evidence for recommendation. However, more history is not always better: historical evidence is often redundant, and, crucially, what is useful often depends on the candidate being ranked. This observation motivates us to reframe long-sequence recommendation from processing longer histories to *candidate-focal history compression*, *i.e.*, preserving the right historical evidence for each candidate. This reframing raises two coupled challenges: a shared compressed history may discard candidate-specific evidence (**C1**), while independently constructing a history for every candidate multiplies redundant computation (**C2**). To study this quality–efficiency tradeoff, we develop a *quality–write–state tradeoff framework* which characterizes the first-order contribution of each recurrent write to the ranking objective, and then quantifies when candidates can share sparse histories with bounded error. Building on this theoretical frame, we introduce DELTAREC, which distills history contributions into a lightweight selector to guide both candidate-focal history writing and utility-based state sharing. Across three datasets and four backbones, we show that DELTAREC **(1)** improves ranking metrics by up to 71.0% (20.1% on average) across all datasets over the corresponding backbones; **(2)** achieves a $17.48\times$ P50 prefill speedup over per-candidate state construction; and **(3)** extends the empirical quality–efficiency Pareto frontier compared to other baselines, achieving higher ranking quality at lower prefill cost than full-history execution. Together, these results support selecting and sharing the right history rather than blindly extending it. We release our codebase.

1 INTRODUCTION

Long-sequence recommendation has extensively studied the scaling capability with longer user histories (Geng et al., 2022; Rajput et al., 2023; Zhai et al., 2024; Deng et al., 2025). They successfully show that longer histories can improve ranking quality (Zhai et al., 2024), while these methods implemented with dense self-attention make this scaling computationally expensive, with quadratic cost in sequence length. Linear recurrent attention, such as gated delta-rule attention (GDR) (Yang et al., 2025), addresses this bottleneck by replacing pairwise token interactions with updates to a compact recurrent state, reducing sequence processing from quadratic to linear complexity (Yang et al., 2024; Team et al., 2025). Recent ranking systems, such as LinRec and FuXi-Linear (Liu et al., 2023; Ye et al., 2026), also leverage linear attention to efficiently scale to longer user histories.

Yet the ability to efficiently process longer histories does not imply that all historical interactions are useful for ranking. This raises a fundamental question: *does recommendation benefit from longer history, or from the right history?* We investigate this question in GDR, where historical interactions are recurrently incorporated into a state through *writes*. Figure 1(a) shows that retaining only a carefully selected 25% of historical writes (Target CWI) achieves 107% of the NDCG@10 of full-length GDR, suggesting that fewer but more relevant writes can outperform the full history. More importantly, Figure 1(b) shows that the *right history* is candidate-specific: an interaction that strongly contributes to one candidate may have little effect on another, while similar candidates can exhibit

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

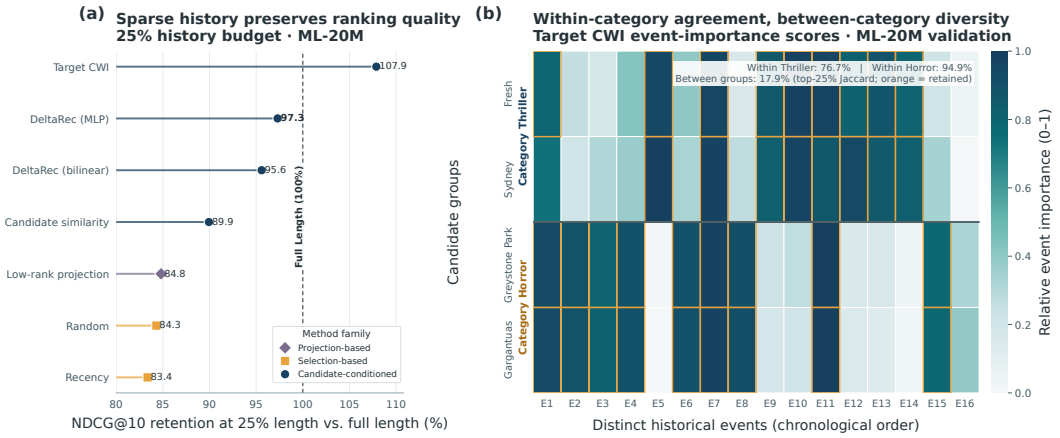


Figure 1: **Long-history redundancy is candidate-dependent.** (a) Informed selection preserves most full-history GDR quality with 25% of the history. (b) Historical importance varies across candidates, with similar candidates showing similar utility patterns.

similar write-utility patterns. These observations are consistent with prior findings that filtering irrelevant behaviors can improve recommendation accuracy (Qin et al., 2020; Pi et al., 2020) and that candidate-focal behavior selection can better preserve relevant user evidence (Pi et al., 2020; Chang et al., 2023). Together, we shift history scaling from simply processing more interactions to identifying which interactions matter to each candidate and when candidate-specific histories can be shared efficiently.

However, candidate-specific histories create a tension between quality and efficiency: preserving the right histories for each candidate may require constructing multiple recurrent states. The key question is therefore: **How can we preserve the right history for each candidate while sharing state construction whenever possible?**

To answer this question, we identify two coupled challenges that balance quality and efficiency. **C1: Preserving candidate-specific ranking quality.** At one extreme, all candidates reuse a single compressed history. Although this representation is cheap to reuse, it may omit historical evidence that is important for particular candidates, as illustrated in Figure 1(b). The challenge is therefore to determine *which historical interactions must be preserved for each candidate* so that reducing the history does not discard the evidence needed for recommendation. **C2: Controlling candidate-specific computation.** At the opposite extreme, each candidate can use its own history; independently selecting and encoding a history for each candidate preserves most important events but can require a separate recurrent-state construction for every candidate. Shorter histories reduce the writes per state, but not the number of states to construct. The challenge is to determine *when candidates can share the same history selection* without sacrificing candidate-specific utility.

We introduce a *quality–write–state tradeoff framework* to balance the two challenges. For **C1**, we first identify which historical interactions are most important to each candidate. Theorem 1 shows that a write is important when its propagated effect on the history representation aligns with the direction of candidate-focal ranking objective. This gives each write a candidate-focal utility and, under a fixed write budget, naturally prioritizes the interactions with the largest utility. For **C2**, we then ask when candidates can safely share the same recurrent state. Theorem 2 shows that candidates relying on similar historical evidence can share state construction with bounded error, characterized by their write-utility discrepancy and a finite-deletion remainder. Thus, candidates with similar write-utility profiles can share state construction with bounded quality error. Together, the two results connect which writes to retain and which candidates can share them, balancing recommendation quality against both write and state computation.

Building on this framework, we introduce DELTAREC, which uses candidate-focal write utility to jointly determine *what history to retain* and *which candidates can share recurrent states*. A lightweight Counterfactual Write Influence (CWI)-distilled selector predicts these utilities to guide sparse history selection and candidate grouping, reducing repeated history-state construction. Across three datasets and four backbones, DELTAREC achieves relative ranking gains of up to 71.0% (20.1% on average) over the corresponding backbones at a 25% history budget. On KuaiRand-1K

with 2k history length, the proposed state sharing yields $17.48\times$–$27.11\times$ P50 prefill speedups over per-candidate state construction. More importantly, **DELTAREC improves the empirical quality–efficiency Pareto frontier rather than merely trading ranking quality for speed**. These results support selecting and sharing the right history rather than simply extending it.

- **Problem reframing.** We reframe long-sequence recommendation from using longer histories to identifying the *right history* for each candidate, revealing a tension between candidate-specific utility and recurrent computation.
- **Theoretical factorization.** We introduce a quality–write–state tradeoff framework that characterizes which writes matter to each candidate and when candidates can safely share recurrent-state construction.
- **Efficient system implementation.** We instantiate the framework with DELTAREC, which jointly uses candidate-focal write utility for sparse history selection and candidate grouping, reducing K state constructions to $G \ll K$.
- **Effectiveness and efficiency validation.** Experiments across three datasets and four backbones demonstrate improved ranking quality at a 25% history budget, up to $27.11\times$ P50 prefill speedup over per-candidate state construction, and a better empirical quality–efficiency Pareto frontier.

2 PRELIMINARIES

Problem statement. Let $\mathcal{X}$ and $\mathcal{V}$ denote the event and item spaces, respectively. For a user, let $L \in \mathbb{N}$ denote the history length, and define the chronological history $H = (x_1, \ldots, x_L) \in \mathcal{X}^L$, where $x_i \in \mathcal{X}$ is the event at position $i \in \{1, \ldots, L\}$. Let $\mathcal{C} \subseteq \mathcal{V}$ be a finite retrieved candidate set containing K items. Our goal is to recommend candidates in $\mathcal{C}$ more accurately while reducing the computation required to construct recurrent states from long user histories.

Recurrent linear attention for history encoding. Recurrent linear attention (Yang et al., 2024; 2025; Bulatov et al., 2022) has been widely developed to reduce the quadratic computation of conventional attention for ultra-long sequences (Team et al., 2026; Qiu et al., 2026). In recommendation systems, it provides an efficient backbone for encoding long user histories (Ye et al., 2026; Liu et al., 2023). In this paper, we focus on the representative work GDR (Yang et al., 2025).

Starting from an initial state $\mathbf{S}_0 \in \mathbb{R}^{d_k \times d_v}$, GDR recurrently updates the state after writing event x_i:

$$F_i(\mathbf{S}) = \gamma_i \mathbf{S} + \beta_i \mathbf{k}_i \left(\mathbf{v}_i - \gamma_i \mathbf{S}^\top \mathbf{k}_i\right)^\top, \qquad \mathbf{S}_i = F_i(\mathbf{S}_{i-1}), \tag{1}$$

where $\mathbf{S}_i \in \mathbb{R}^{d_k \times d_v}$ is the recurrent state after event x_i, $\mathbf{k}_i \in \mathbb{R}^{d_k}$ and $\mathbf{v}_i \in \mathbb{R}^{d_v}$ are its key and value vectors, respectively, and γ_i and β_i are the decay and write-strength gates. The decay gate γ_i controls forgetting, while β_i controls how strongly the current event updates the state. We refer to the complete transition $\mathbf{S}_{i-1} \to \mathbf{S}_i$ as a *recurrent write*; skipping it leaves the state unchanged.

Extreme case of C1: shared histories can lose candidate-specific evidence. A single state $\mathbf{S}$ from a compressed history is shared across all K candidates, although different candidates may rely on different events. Thus, **C1** asks *which recurrent writes preserve candidate-specific utility*.

Extreme case of C2: candidate-specific histories duplicate state construction. At the other extreme, building a separate history for each of K candidates requires up to K recurrent states and KB retained writes. We therefore separate history compression, $\rho = B/L$, from state sharing, $G \in [1, K]$, where $G = 1$ denotes full sharing and $G = K$ fully candidate-specific construction. Efficient serving must control both *which writes are retained* and *how many states are constructed*.

3 A QUALITY–WRITE–STATE TRADEOFF FRAMEWORK

The two challenges above expose a fundamental quality–efficiency tension. A single compressed history is cheap to reuse but may discard evidence needed by individual candidates (**C1**), whereas fully candidate-specific histories preserve such evidence but multiply recurrent state construction (**C2**). We organize this tension along three axes. We first explicitly identify the utility of each recurrent write for ranking quality (*Quality*), then use this utility to allocate a limited recurrent-write budget (*Write*), and finally characterize when candidate-specific sparse histories can share state construction (*State*).

3.1 QUALITY: WHAT MATTERS TO CANDIDATE RANKING

Define candidate-focal recommendation utility. Preserving more history does not necessarily preserve more ranking-relevant information. An event may substantially alter the recurrent state while contributing little to a particular candidate. For example, when ranking a soccer jersey, a write of furniture purchase event may induce a large state change but be less informative than prior interactions with soccer-related items. To quantify this distinction, we evaluate each write by its effect on the standard softmax ranking objective for a focal candidate.

Let $\mathbf{q}_H = R_{\boldsymbol{\theta}}(\mathbf{S}_L) \in \mathbb{R}^d$ denote the history-query representation, where $R_{\boldsymbol{\theta}} : \mathbb{R}^{d_k \times d_v} \to \mathbb{R}^d$ is the learned ranking readout parameterized by $\boldsymbol{\theta}$. Let $\mathbf{e}_a \in \mathbb{R}^d$ denote the embedding of any candidate $a \in \mathcal{C}$. For temperature $\tau > 0$, the recommendation objective is defined as

$$\ell_c(H, \mathcal{C}) = -\log \frac{\exp(s_c(H)/\tau)}{\sum_{a \in \mathcal{C}} \exp(s_a(H)/\tau)}, \quad s_a(H) = \mathbf{q}_H^\top \mathbf{e}_a. \tag{2}$$

Here, $c \in \mathcal{C}$ denotes an arbitrary focal candidate whose support from the history is being evaluated. Lower ℓ_c indicates stronger relative support for c against the same retrieved candidate set.

Identify utility by counterfactual write influence. A write that is important to the overall history representation need not be equally useful for every candidate. In other words, global history importance does not imply candidate-specific ranking utility. We therefore evaluate each write counterfactually with respect to a focal candidate: how would the ranking support for candidate c change if this write were suppressed?

To formalize this counterfactual intervention, we introduce a gate m_i on each complete state transition. Let $\tilde{\mathbf{S}}_i(\mathbf{m})$ denote the recurrent state under gate vector $\mathbf{m}$. The gated transition is

$$\tilde{\mathbf{S}}_i(\mathbf{m}) = \tilde{\mathbf{S}}_{i-1}(\mathbf{m}) + m_i \left[F_i\Big(\tilde{\mathbf{S}}_{i-1}(\mathbf{m})\Big) - \tilde{\mathbf{S}}_{i-1}(\mathbf{m}) \right], \qquad m_i \in [0, 1], \tag{3}$$

where $\mathbf{m} = (m_1, \ldots, m_L)^\top \in [0, 1]^L$. At the endpoints, $m_i = 1$ executes the original transition F_i in Eq. (1), whereas $m_i = 0$ skips the complete write induced by x_i. Accordingly, we define $\mathbf{q}_H(\mathbf{m}) = R_{\boldsymbol{\theta}}(\tilde{\mathbf{S}}_L(\mathbf{m}))$ and let $\ell_c(\mathbf{m}; H, \mathcal{C})$ denote the ranking loss in Eq. (2) evaluated under the gate vector.

Inspired by influence-based analysis (Koh & Liang, 2017; Chen et al., 2026), we define the *Counterfactual Write Influence* (CWI) of event x_i for candidate c as

$$\omega_i(c) := -\left. \frac{\partial \ell_c(\mathbf{m}; H, \mathcal{C})}{\partial m_i} \right|_{\mathbf{m}=\mathbf{1}}. \tag{4}$$

A positive $\omega_i(c)$ indicates that locally suppressing the write increases the focal loss, while a larger value indicates stronger utility for ranking candidate c. CWI therefore measures the first-order contribution of each write around the full-history trajectory.

Theorem 1 (Candidate-factorized write influence)**.** *Fix H, $\mathcal{C}$, and a differentiable frozen ranking model. At the full history $\mathbf{m} = \mathbf{1}$, let $\mathbf{J}_i := \left. \frac{\partial \mathbf{q}_H(\mathbf{m})}{\partial m_i} \right|_{\mathbf{m}=\mathbf{1}} \in \mathbb{R}^d$, $p_a := \frac{\exp(s_a(H)/\tau)}{\sum_{b \in \mathcal{C}} \exp(s_b(H)/\tau)}$, where p_a is the corresponding softmax probability of candidate $a \in \mathcal{C}$. Define the probability-weighted candidate embedding as $\bar{\mathbf{e}}_p := \sum_{a \in \mathcal{C}} p_a \mathbf{e}_a$. Then the CWI of write i for candidate c factorizes as*

$$\omega_i(c) = \frac{1}{\tau} \underbrace{\mathbf{J}_i^\top}_{\text{propagated write effect}} \underbrace{(\mathbf{e}_c - \bar{\mathbf{e}}_p)}_{\text{candidate ranking direction}}. \tag{5}$$

Theorem 1 reveals the key mechanism behind challenge **C1**: write utility is determined not only by how strongly an event changes the history representation, but by how that propagated change aligns with the ranking direction of the focal candidate. The same write can therefore be highly useful for one candidate yet nearly irrelevant to another, and changing the focal candidate can even reverse the utility ordering of two writes (Corollary 1). Consequently, a single candidate-agnostic importance ordering cannot generally preserve the writes most relevant to every ranking decision, motivating candidate-focal write selection. Proofs are in Appendix B.

3.2 WRITE: UTILITY-GUIDED SPARSE WRITING

The previous subsection establishes CWI as a candidate-specific measure of write utility for ranking quality. We now use this utility to determine which recurrent writes should be retained.

For a retained index set $\mathcal{I} \subseteq [L]$, where $[L] := \{1, \ldots, L\}$, let $\mathbf{m}_\mathcal{I} \in \{0,1\}^L$ denote the corresponding intervention vector, with $(\mathbf{m}_\mathcal{I})_i = 1$ if and only if $i \in \mathcal{I}$. Given a write budget $B \in \{1, \ldots, L\}$, the ideal candidate-focal sparse history is obtained by

$$\mathcal{I}_c^\star \in \underset{\substack{\mathcal{I}\subseteq[L] \\ |\mathcal{I}|=B}}{\arg\min}\, \ell_c(\mathbf{m}_\mathcal{I}; H, \mathcal{C}). \tag{6}$$

Directly solving Eq. (6) requires combinatorial subset search. Instead, we approximate the objective around the full-history trajectory $\mathbf{m} = \mathbf{1}$. Using the CWI definition (Eq. (4)), a first-order expansion gives $\ell_c(\mathbf{m}_\mathcal{I}; H, \mathcal{C}) \approx \ell_c(\mathbf{1}; H, \mathcal{C}) + \sum_{i \notin \mathcal{I}} \omega_i(c)$. Because the first term is constant across subsets, minimizing the linearized objective is equivalent to retaining the B writes with the largest CWI.

This candidate-focal Top-B selection improves write quality, but is not by itself efficient: for K candidates, independently constructing their sparse histories can require up to K recurrent states and KB retained writes. The remaining challenge is therefore to share state construction across candidates without losing their candidate-specific utility.

3.3 STATE: UTILITY-BASED STATE SHARING

Candidate-focal sparse writing addresses challenge **C1**, but applying it independently to every candidate exposes challenge **C2**: K candidates may induce up to K distinct sparse histories and therefore require K separately constructed recurrent states.

To address this, we share state construction among candidates that require similar historical evidence. Let $\{\mathcal{G}_g\}_{g=1}^G$ be a partition of the candidate set $\mathcal{C}$, where $\mathcal{G}_g$ is the g-th candidate group and $1 \le G \le K$. For each event x_i, let $\Omega_i(\mathcal{G}_g) \in \mathbb{R}$ denote the group's shared approximation to its members' candidate-specific write utilities. Under write budget B, group $\mathcal{G}_g$ retains $\widehat{\mathcal{I}}_g := \text{Top-B}_{i\in[L]}\, \Omega_i(\mathcal{G}_g)$. All candidates in $\mathcal{G}_g$ therefore share the sparse history indexed by $\widehat{\mathcal{I}}_g$, allowing its recurrent state to be constructed once and reused within the group. For each candidate $c \in \mathcal{G}_g$, define the group utility-approximation error as $\epsilon_g(c) := \max_{i\in[L]} |\Omega_i(\mathcal{G}_g) - \omega_i(c)|$.

Theorem 2 (Utility-preserving state sharing)**.** *For any $c \in \mathcal{G}_g$, suppose the gate-loss Hessian $\nabla^2_\mathbf{m} \ell_c(\mathbf{m}; H, \mathcal{C})$ has operator norm at most $\kappa_c \ge 0$ along the relevant deletion paths. Then the candidate-focal loss incurred by using the group-shared sparse history satisfies*

$$\ell_c(\mathbf{m}_{\widehat{\mathcal{I}}_g}; H, \mathcal{C}) - \ell_c(\mathbf{m}_{\mathcal{I}_c^\star}; H, \mathcal{C}) \le \underbrace{2B\,\epsilon_g(c)}_{\textit{sharing approximation}} + \underbrace{\kappa_c(L-B)}_{\textit{finite-deletion remainder}}. \tag{7}$$

Theorem 2 characterizes when candidate-specific sparse histories can share state construction with bounded error. When the group-level utility profile closely approximates the candidate-specific profiles, *i.e.*, $\epsilon_g(c)$ is small, sharing incurs only a bounded increase in the candidate-focal loss. For a fixed write budget B, reducing the number of groups G reduces the number of recurrent states that must be constructed, while more aggressive sharing can increase $\epsilon_g(c)$. The theorem therefore exposes the tradeoff between state-sharing efficiency and candidate-focal ranking quality. The proof and sufficient conditions for controlling $\epsilon_g(c)$ are provided in Appendix B.3.

Together, Theorem 1 and Theorem 2 establish the *quality–write–state tradeoff*: identify which writes preserve candidate-focal ranking quality, allocate a limited write budget according to their utility, and share state construction when candidate-specific utility profiles remain sufficiently compatible. This progressively reduces recurrent computation from $O(KL)$ writes of candidate-specific full histories, to $O(KB)$ with candidate-specific sparse writing, and further to $O(GB)$ with utility-based state sharing, where $B \ll L$ and $G \ll K$.

4 DELTAREC: SELECT WRITES, SHARE STATES

Building on Section 3, DELTAREC implements utility-guided selection (Section 3.2) and state sharing (Section 3.3) through three stages (Figure 2). Offline training distills CWI into a selector and

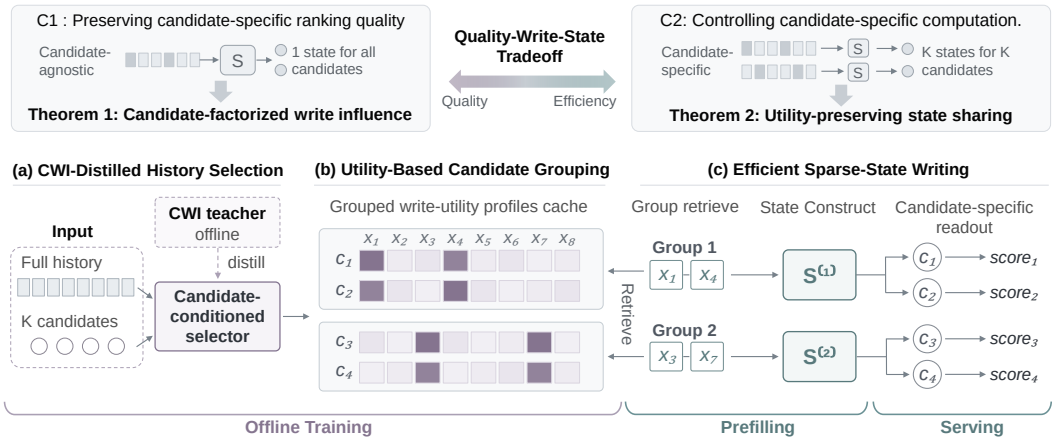


Figure 2: **DELTAREC overview.** Training learns a CWI-guided selector and adapts the ranking model. Preparation caches utility-based candidate groups and retained event indices. Prefill constructs group states; serving reuses them for candidate-focal scoring.

adapts the ranking model to sparse histories. Cache preparation groups candidates by predicted write utility and records their retained event indices and group assignments. Given this metadata, prefill constructs one state per active group, and serving reuses these states for candidate-focal scoring.

4.1 CWI-DISTILLED HISTORY SELECTION

Model training and CWI distillation. To operationalize the *Quality–Write* connection (Theorem 1), DELTAREC learns to predict candidate-focal write utility instead of differentiating through the full history at inference time (Chen et al., 2026). We first obtain warm-up checkpoints of a full-history GDR model and compute trajectory-level CWI targets using Eq. (4), which serve as supervision for the selector. We then distill these utility signals into a candidate-conditioned selector, $\widehat{\omega}_i(c) = s_{\boldsymbol{\phi}}(\mathbf{z}_{i,c})$, where $\mathbf{z}_{i,c}$ contains i-th event feature and candidate c.

Cache preparation and sparse fine-tuning. After distillation, we freeze the selector and use it to produce the candidate-focal utility profile $\widehat{\boldsymbol{\omega}}(c) = (\widehat{\omega}_1(c), \ldots, \widehat{\omega}_L(c))^\top$. These profiles guide history selection within fixed utility groups, and cache preparation stores the retained indices. We then fine-tune the recommendation model on the resulting utility-selected sparse histories, adapting it to the compressed inputs used at inference time. Appendix C.1 details the training protocol, and Algorithm 1 summarizes the complete offline training procedure.

4.2 UTILITY-BASED CANDIDATE GROUPING

The *State* principle in Theorem 2 motivates sharing among candidates with compatible write utilities. We form G utility groups offline by clustering training-data CWI. During cache preparation, DELTAREC maps each request's candidates to these fixed groups rather than clustering them again. For each nonempty group $\mathcal{G}_g$, an aggregator A forms the shared utility $\Omega_i(\mathcal{G}_g) = A(\{\widehat{\omega}_i(c) : c \in \mathcal{G}_g\})$, which determines the retained indices $\widehat{\mathcal{I}}_g = \text{Top-B}_{i\in[L]}\, \Omega_i(\mathcal{G}_g)$.

Thus, candidates share a sparse history based on CWI-derived utility rather than candidate embedding similarity alone. We cache the assignments $g(c)$ and retained index sets $\widehat{\mathcal{I}}_g$ for the corresponding history and candidate set; recurrent states are constructed later during prefill. More details about budget rule, offline grouping, and request-time group assignment are specified in Appendix C.2.

4.3 EFFICIENT SPARSE-STATE EXECUTION

Figure 3 illustrates execution from cached selection and grouping metadata through two stages.

Prefill. Given a history H and its cached retained indices $\widehat{\mathcal{I}}$, DELTAREC gathers the selected events in chronological order and processes the sparse histories of nonempty groups to produce $\mathbf{S}^{(g)} = \text{GDR}(H_{\widehat{\mathcal{I}}_g})$, where $H_{\widehat{\mathcal{I}}_g}$ denotes the retained history of group g. Packed execution processes these histories together while keeping recurrent states independent.

Serving. For each candidate c, DELTAREC retrieves its group index $g(c)$ and computes $\text{Score}_c = f_s\big(R_{\boldsymbol{\theta}}(\mathbf{S}^{(g(c))}), \mathbf{e}_c\big)$. Here, $R_{\boldsymbol{\theta}}$ extracts the history-query representation from the group state, and f_s combines it with the candidate representation to produce a score. Candidates thus reuse recurrent states without sharing their final scores.

With B retained writes per group, prefill constructs at most G states using at most GB recurrent writes, rather than KB for independently executed candidate-specific sparse histories. Serving produces K candidate scores without further recurrent updates. Packed execution and cache handling are detailed in Appendix C.3; Algorithm 2 summarizes prefill and serving.

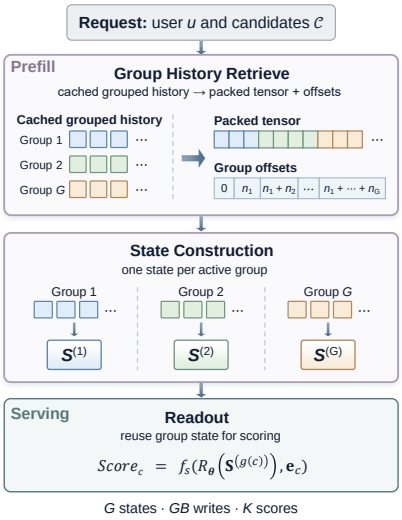


Figure 3: Sparse-state execution.

5 EXPERIMENTS

We evaluate DELTAREC through four research questions: **RQ1 (Quality comparison):** Can DELTAREC preserve or improve ranking quality compared with state-of-the-art (SOTA) recommendation models? **RQ2 (Prefill and serving efficiency):** prefill and serving latencies are critical in production, we then ask how efficiently can DELTAREC construct reusable history states and serve candidates from cached states? **RQ3 (Model analysis):** How do the proposed components and key parameters affect ranking quality and efficiency? **RQ4 (Quality–efficiency tradeoff):** How well does DELTAREC balance ranking quality and computational cost?

Experimental settings. We describe our experimental settings as follows: **For datasets and metrics**, we evaluate on ML-20M, Amazon Books, and KuaiRand-1K. The candidate pool of ML-20M and Amazon Books are constructed via a fixed top-100 candidate set from a well-trained HSTU; KuaiRand-1K uses its provided 32-candidate pool. We report HR@10 and NDCG@10 on ML-20M and Amazon Books, and macro-GAUC averaged over eight tasks on KuaiRand-1K. **For baselines**, we compare with three classes of SOTA methods, including conventional attention SOTA HSTU (Zhai et al., 2024), linear attention SOTAs, LinRec and FuXi-Linear (Liu et al., 2023; Ye et al., 2026), and an attention sparsification SOTA BlossomRec (Ma et al., 2026). We further plus DELTAREC on the four backends for comparison. **For training details,** all models are trained on two NVIDIA A100 GPUs using their published implementations and dataset-specific settings. For DELTAREC, we replace the attention layer with GDR (Eq. (3)) while keeping other backbone architecture unchanged. Full experimental details are in Appendix D.

5.1 QUALITY COMPARISON (RQ1)

We evaluate ranking quality on all three datasets and instantiate DELTAREC with multiple recommendation backbones. This comparison asks whether candidate-focal history compression merely reduces computation, or can also preserve and improve the ranking quality of the underlying model.

(1) DELTAREC selection improves the underlying backbones with $\rho = 0.25$ serving budget. Table 1 shows improvements for all four backbones. Table 1 shows average relative improvements of **27.5%**, **22.2%**, and **0.9%** on ML-20M, Amazon Books, and KuaiRand-1K, respectively, averaging all backbones. All four backbones improve on both metrics for ML-20M and Amazon Books, with gains reaching **71.0%** in NDCG@10. **(2) DELTAREC selection adds value beyond efficient attention and existing history compression.** The gains extend beyond HSTU to the linear-attention backbones LinRec and FuXi-Linear, indicating that more efficient history processing still benefits from selective retention. Even for BlossomRec, which already compresses history, DELTAREC further improves NDCG@10 by **14.4%** on ML-20M and **33.9%** on Amazon Books. These results support our central premise: selecting the right historical evidence for each candidate complements, rather than replaces, improvements in how efficiently a backbone processes history. More comparisons are provided in Appendix E.1.

5.2 PREFILL AND CACHED-SERVING EFFICIENCY (RQ2)

We next evaluate system efficiency on a fixed KuaiRand-1K workload for prefill and serving stages. We compare one shared state, one state per candidate, and group-shared states to isolate how the

Table 1: Ranking quality on three recommendation datasets with $\rho = 0.25$ history budget. Parentheses report relative changes from the corresponding backbone.

	ML-20M		Amazon Books		KuaiRand-1K
Method	HR@10	NDCG@10	HR@10	NDCG@10	Macro-GAUC
LinRec (Liu et al., 2023)	0.2348	0.1288	0.0448	0.0252	0.5248
DELTAREC(LinRec)	**0.2428** (+3.4%)	**0.1339** (+4.0%)	**0.0559** (+24.8%)	**0.0343** (+36.1%)	**0.5256** (+0.2%)
HSTU (Zhai et al., 2024)	0.1496	0.0753	0.0441	0.0238	0.5348
DELTAREC (HSTU)	**0.2364** (+58.0%)	**0.1288** (+71.0%)	**0.0537** (+21.8%)	**0.0255** (+7.1%)	**0.5419** (+1.3%)
FuXi-Linear (Ye et al., 2026)	0.1857	0.1008	0.0350	0.0185	0.5222
DELTAREC (FuXi-Linear)	**0.2393** (+28.8%)	**0.1286** (+27.6%)	**0.0409** (+16.9%)	**0.0228** (+23.2%)	**0.5361** (+2.7%)
BlossomRec (Ma et al., 2026)	0.2114	0.1135	0.0426	0.0218	**0.5387**
DELTAREC (BlossomRec)	**0.2380** (+12.6%)	**0.1298** (+14.4%)	**0.0485** (+13.8%)	**0.0292** (+33.9%)	0.5365 (-0.4%)

Table 2: Prefill and cached-serving efficiency on KuaiRand-1K. Upper part is HSTU and its backend DELTAREC, bottom part is FuXi-Linear and its backend. Overall denotes prefill + serving mean. L, K, G, and ρ denote history length, candidate numbers, group numbers, and history retention ratio.

	GAUC	Prefill (ms / batch)				Serving (ms / batch)				Overall
Method		Mean	P50	P95	Big-O	Mean	P50	P95	Big-O	Mean
HSTU	0.5348	**8.724**	**8.672**	**9.099**	$\mathcal{O}(L^2)$	6.610	6.598	6.774	$\mathcal{O}(KL)$	15.334
DELTAREC (1-state)	0.5288	9.190	9.123	9.438	$\mathcal{O}(L)$	3.825	3.820	3.884	$\mathcal{O}(K)$	**13.015**
DELTAREC (K-state)	0.5364	182.579	182.399	183.312	$\mathcal{O}(K\rho L)$	4.093	4.051	4.430	$\mathcal{O}(K)$	186.672
DELTAREC (G-state)	**0.5419**	10.519	10.434	11.203	$\mathcal{O}(G\rho L)$	**3.596**	**3.783**	**3.859**	$\mathcal{O}(K)$	14.115
FuXi-Linear	0.5222	15.475	15.429	15.682	$\mathcal{O}(L)$	3.093	3.072	3.210	$\mathcal{O}(K)$	18.568
DELTAREC (1-state)	0.5283	**13.409**	**13.361**	**13.806**	$\mathcal{O}(L)$	**0.253**	**0.249**	**0.286**	$\mathcal{O}(K)$	**13.662**
DELTAREC (K-state)	0.5359	989.229	989.124	990.695	$\mathcal{O}(K\rho L)$	0.321	0.318	0.343	$\mathcal{O}(K)$	989.550
DELTAREC (G-state)	**0.5361**	36.560	36.479	37.071	$\mathcal{O}(G\rho L)$	0.287	0.287	0.301	$\mathcal{O}(K)$	36.847

number of recurrent states affects cost. The detailed workload, stage boundaries, hardware, and measurement protocol are provided in Appendix D.2.

Table 2 shows the comparisons with two DELTAREC backbone. We conclude that, **(1) state sharing can largely increase efficiency**: for both backbones, the 1-state and G-state variants are more efficient than the K-state variant. With the HSTU backend, grouping reduces P50 prefill latency from 182.399 to 10.434 ms/batch relative to K-state execution, a $17.48\times$ speedup. With FuXi-Linear, it reduces P50 prefill latency from 989.124 to 36.479 ms/batch, a $27.11\times$ speedup. These results show that shortening each history alone is insufficient: sharing state construction is essential to avoid repeated candidate-specific computation. **(2) The grouped design preserves fast serving without sacrificing quality.** The G-state variant achieves the highest GAUC within each backend: 0.5419 with HSTU and 0.5361 with FuXi-Linear. Meanwhile, grouping reduces overall latency from 186.672 to 14.115 ms/batch with HSTU and from 989.550 to 36.847 ms/batch with FuXi-Linear, compared with K-state execution. The consistent gains over candidate-specific K-state construction show that the benefit of state sharing is not specific to one architecture.

5.3 MODEL ANALYSIS (RQ3)

We first ablate the history-selection strategy to examine whether candidate-focal utility is necessary for identifying useful interactions. *Original* DELTAREC *(HSTU)* uses the CWI-distilled, candidate-focal selector. *w/o history selection (full-length GDR)* retains the entire history. *w/o CWI guidance (candidate similarity)* selects interactions by their similarity to the candidate instead of CWI utility. The two *w/o candidate aware* variants use candidate-independent selection: *Recency-only* retains the most recent interactions, while *Rank-only* retains interactions based on candidate-independent ranking. We then vary the number of shared states G and retention ratio ρ to study how the two controllable computation budgets affect ranking quality and prefill cost. Details are in Appendix D.3.

Table 3 illustrates the effect of different history-selection policies. We can get the following results. **(1) The advantage of history compression is dataset-dependant:** without history selection, DELTAREC may increase 12.7% on ML-20M while decrease 25% and 2.4% on Amazon Books and KuaiRand. **(2) Candidate-aware with CWI guidance is influential:** simply compressing with

Table 3: Ablation of history-selection policies. We show HR@10 for ML-20M and Amazon Books, and macro-GAUC for KuaiRand-1K as metrics due to space limits.

Variants	ML-20M	Amazon Books	KuaiRand-1K
Original DELTAREC(HSTU)	0.2364	**0.0537**	**0.5419**
w/o history selection (full-length GDR)	**0.2664** (+12.7%)	0.0403 (-25.0%)	0.5288 (-2.4%)
w/o CWI guidance (candidate similarity)	0.2130 (-9.9%)	0.0375 (-30.2%)	0.5346 (-1.3%)
w/o candidate aware (Recency-only)	0.2272 (-3.9%)	0.0363 (-32.4%)	0.5348 (-1.3%)
w/o candidate aware (Rank-only)	0.1409 (-40.4%)	0.0352 (-34.5%)	0.5222 (-3.6%)

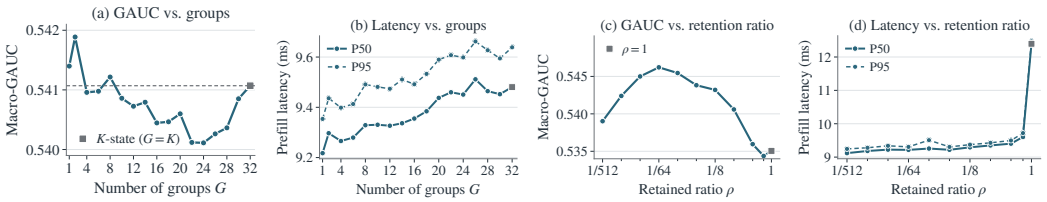


Figure 4: **Quality–efficiency sensitivity to state sharing and history retention.** (a,b) vary the number of groups at $\rho = 0.25$; (c,d) vary the retained-history ratio.

candidate similarity (w/o CWI guidance) decreases performance, while it is better than the other candidate-agnostic selection (rank-only), and recent histories are also important.

Figure 4 shows how state sharing and history retention affect quality and prefill latency. **(1) more states do not necessarily improve ranking**, as Macro-GAUC peaks at $G = 2$, while prefill latency generally increases with G. **(2) more history is not always better**, as Macro-GAUC peaks at $\rho = 1/64$ rather than full retention. Full retention yields lower quality than this peak and sharply increases prefill latency, showing that smaller, but well-selected histories can improve both quality and efficiency. More generalizability analysis is in Appendix E.3 and E.4.

5.4 QUALITY–EFFICIENCY TRADEOFF (RQ4)

Finally, we jointly compare ranking quality and P50 prefill latency across retention ratios in Figure 5. Unlike RQ3, which studies each control parameter separately, this experiment asks whether history compression creates a true quality–efficiency tradeoff or can improve both objectives simultaneously.

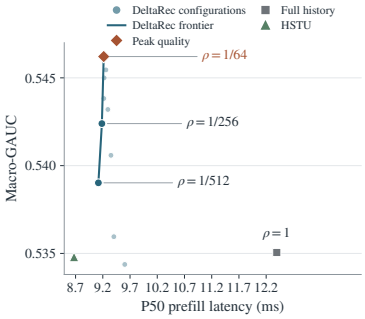


Figure 5: Pareto frontier across ρ.

(1) DELTAREC moves the operating point toward a better Pareto frontier, rather than trading quality for speed. Several compressed configurations simultaneously achieve higher Macro-GAUC and lower prefill latency than the full-history endpoint. The best observed quality appears around $\rho = 1/64$, reaching approximately 0.546 Macro-GAUC with low prefill latency. HSTU has lower prefill latency but also lower Macro-GAUC. **(2) The result reinforces the "right history" view of long-sequence recommendation.** Full-history ($\rho = 1$) is neither the highest-quality nor the most efficient operating point in Figure 5. Effective long-sequence recommendation therefore depends on allocating computation to useful historical evidence, rather than blindly processing the maximum available context. More results can be found at Appendix E.2

6 CONCLUSION

In this work, we investigate long-sequence recommendation from the perspective of *which* historical evidence should be preserved, rather than simply how much history can be processed. We introduce the *quality–write–state tradeoff* theoretical framework, where CWI identifies candidate-focal write utility, sparse writing allocates a limited writing budget to useful events, and state sharing reduces redundant candidate-specific computation. Building on this framework, DELTAREC combines CWI-distilled selection with utility-based candidate grouping for efficient recurrent execution. Experiments show that DELTAREC can substantially improve the recommendation results across four backbones, and extends the observed quality–efficiency Pareto frontier. These results suggest that effective long-sequence recommendation depends not only on scaling history length, but on allocating computation to the right history and the right candidate-specific states.

AI USE STATEMENT

Generative AI tools were used to assist with dataset preprocessing, reimplementing baseline methods, and proofreading the manuscript.

REPRODUCIBILITY STATEMENT

The appendices specify the details of the method, data protocol, metrics, and training details. We also open-source our code to easily reproduce the results with an anonymous codebase. All checkpoints and artifacts will be published after the review process due to large file sizes.

REFERENCES

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *NeurIPS*, 35: 11079–11091, 2022.

Jianxin Chang, Chenbin Zhang, Zhiyi Fu, Xiaoxue Zang, Lin Guan, Jing Lu, Yiqun Hui, Dewei Leng, Yanan Niu, Yang Song, et al. 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*, pp. 3785–3794, 2023.

Jianpeng Chen, Wangzhi Zhan, Haohui Wang, Dongqi Fu, and Dawei Zhou. Duetda: Decomposed and dynamic data attribution with model-state gating for accelerated scientific endeavors. In *Proceedings of the 32nd ACM SIGKDD V.2*, KDD '26, pp. 10589–10600, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3818877.

Benjamin Cohen-Wang, Harshay Shah, Kristian Georgiev, and Aleksander Mkadry. Contextcite: Attributing model generation to context. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 95764–95807. Curran Associates, Inc., 2024. doi: 10.52202/079017-3035.

Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965*, 2025.

Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (RLP): A unified pretrain, personalized prompt & predict paradigm (P5). In *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 299–315, 2022. doi: 10.1145/3523227.3546767. URL https://doi.org/10.1145/3523227.3546767.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *ICML*, pp. 5156–5165. PMLR, 2020.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, pp. 1885–1894. PMLR, 2017.

Langming Liu, Liu Cai, Chi Zhang, Xiangyu Zhao, Jingtong Gao, Wanyu Wang, Yifu Lv, Wenqi Fan, Yiqi Wang, Ming He, et al. Linrec: Linear attention mechanism for long-term sequential recommender systems. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pp. 289–299, 2023.

Mengyang Ma, Xiaopeng Li, Wanyu Wang, Zhaocheng Du, Jingtong Gao, Pengyue Jia, Yuyang Ye, Yiqi Wang, Yunpeng Weng, Weihong Luo, et al. Blossomrec: Block-level fused sparse attention mechanism for sequential recommendations. In *Proceedings of the ACM Web Conference 2026*, pp. 6389–6399, 2026.

Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 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*, pp. 2685–2692, 2020.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In *Advances in neural information processing systems*, volume 33, pp. 19920–19930, 2020.

Jiarui Qin, Weinan Zhang, Xin Wu, Jiarui Jin, Yuchen Fang, and Yong Yu. User behavior retrieval for click-through rate prediction. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, pp. 2347–2356, 2020.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, et al. On the design of qwen3. 8-next architecture: Evaluation, efficiency, and training stability. *arXiv preprint arXiv:2608.30320*, 2026.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10299–10315, 2023.

Zihua Si, Lin Guan, ZhongXiang Sun, Xiaoxue Zang, Jing Lu, Yiqun Hui, Xingchao Cao, Zeyu Yang, Yichen Zheng, Dewei Leng, et al. 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*, pp. 4890–4897, 2024.

Zhongxiang Sun, Kepu Zhang, Haoyu Wang, Xiao Zhang, and Jun Xu. Effective in-context example selection through data compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 871–877, 2024.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Khanh Hiep Tran, Azin Ghazimatin, and Rishiraj Saha Roy. Counterfactual explanations for neural recommenders. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1627–1631, 2021.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *NeurIPS*, volume 37, pp. 115491–115522, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *ICLR*, volume 2025, pp. 29687–29707, 2025.

Yufei Ye, Wei Guo, Hao Wang, Luankang Zhang, Heng Chang, Hong Zhu, Yuyang Ye, Yong Liu, Defu Lian, and Enhong Chen. Fuxi-linear: Unleashing the power of linear attention in long-term time-aware sequential recommendation, 2026.

Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Jiayuan He, Yinghai Lu, and Yu Shi. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Proceedings of the 41st ICML*, volume 235 of *Proceedings of Machine Learning Research*, pp. 58484–58509, 21–27 Jul 2024.

Yuren Zhang, Enhong Chen, Binbin Jin, Hao Wang, Min Hou, Wei Huang, and Runlong Yu. Clustering based behavior sampling with long sequential data for ctr prediction. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2195–2200, 2022.

A Related Works

A.1 Linear Attention for Recommender Systems

Linear attention reduces the quadratic cost of self-attention by replacing pairwise token interactions with associative computation, and can be implemented recurrently with a compact state (Katharopoulos et al., 2020). In recommendation, LinRec (Liu et al., 2023) uses normalized linear attention for long histories, while FuXi-Linear (Ye et al., 2026) incorporates semantic, temporal, and positional modeling while retaining linear complexity. These methods mainly improve how a long history is efficiently encoded. In contrast, DeltaRec asks whether every historical interaction needs to be written into the recurrent state for every candidate. Built on gated delta-rule attention (Yang et al., 2025), DeltaRec selects candidate-relevant state updates and allows candidates with similar write requirements to share state construction. Therefore, our focus is not only on reducing the complexity of the attention operator, but also on reducing the amount of history and the number of recurrent states that need to be processed.

A.2 History Compression in Recommender Systems

History compression reduces the amount of historical information processed for recommendation. UBR4CTR (Qin et al., 2020) and SIM (Pi et al., 2020) first retrieve behaviors that are relevant to a target item and then perform more detailed interest modeling, while TWIN (Chang et al., 2023) improves the consistency between behavior retrieval and target-aware attention. Other methods compress histories through clustering. UBCS (Zhang et al., 2022) clusters candidate items to accelerate behavior sampling, and TWIN V2 (Si et al., 2024) groups historical behaviors for cluster-aware target attention. BlossomRec (Ma et al., 2026) combines block-level long-term selection with sparse attention over recent behaviors. These methods primarily reduce the history processed for each prediction. DeltaRec not only implies compression for recurrent writing in linear attention, but also differs from these methods by considering history selection and recurrent-state construction together. It estimates how useful each historical write is for a candidate, uses this signal to retain important events, and further groups candidates that require similar writes so that they can share a recurrent state. This directly connects the retained-history budget and the number of constructed states to candidate-focal ranking quality.

A.3 Influence and Attribution for Context Compression

Influence and attribution methods study how individual examples or context elements affect model predictions. Classical influence functions estimate how changing the weight of a training example affects the learned model and its predictions (Koh & Liang, 2017), while TracIn (Pruthi et al., 2020) estimates training-example influence using gradient similarity across training checkpoints. In recommendation, ACCENT (Tran et al., 2021) identifies historical user actions whose removal changes the recommended item. For language-model contexts, Sun et al. (2024) use relevance retrieval and meta-gradient-based influence estimates to select demonstrations, while ContextCite (Cohen-Wang et al., 2024) measures the contribution of context sources through ablation and uses it for context pruning. Our CWI considers a different question: how much does a historical event matter when its recurrent state update is suppressed in a frozen ranking model? It provides a local first-order estimate of the effect of each write on the candidate-focal ranking loss. DeltaRec then distills this signal into a lightweight selector for history compression and uses differences in write utility to determine when candidates can safely share recurrent states. In this way, influence estimation is used not only to identify important history, but also to guide efficient recurrent execution.

B CWI Derivations and the Scope of the Guarantees

The results below make explicit the derivations underlying the supplied formulation. They concern a frozen differentiable model, fixed history features, a fixed candidate set, and the complete-transition intervention. They do not assume that the empirical curvature or approximation errors are small.

B.1 Factorization and Candidate-Focal Event Ordering

Proof of Theorem 1. The candidate-focal softmax loss can be written as

$$\ell_c(q) = -\frac{q^\top e_c}{\tau} + \log \sum_{a \in \mathcal{C}} \exp(q^\top e_a/\tau).$$

Hence $\nabla_q \ell_c = (\bar{e}_p - e_c)/\tau$. Differentiation through the recurrent query gives $\partial \ell_c / \partial m_i = J_i^\top (\bar{e}_p - e_c)/\tau$. Negating this derivative gives Eq. (5). □

Corollary 1 (Candidate-specific event ordering)**.** *Under the assumptions of Theorem 1, for $c, c' \in \mathcal{C}$ and $i, j \in [L]$,*

$$\begin{aligned} \omega_i(c) - \omega_i(c') &= \frac{1}{\tau} J_i^\top (e_c - e_{c'}), \\ [\omega_i(c) - \omega_j(c)] - [\omega_i(c') - \omega_j(c')] &= \frac{1}{\tau} (J_i - J_j)^\top (e_c - e_{c'}). \end{aligned} \tag{8}$$

Proof. The common competitor term $\bar{e}_p$ cancels when subtracting focal candidates. Subtracting the resulting identities for i and j gives the second equality. □

A nonzero difference between pairwise margins is not itself an order reversal. Reversal requires those margins to have opposite signs. The corollary permits such reversals and identifies their dependence; whether they occur frequently enough to matter is an empirical question.

B.2 Finite Deletion and Exact-Budget Selection

Let $\mathbf{u}_i$ denote the ith coordinate basis vector, to distinguish it from candidate embedding e_c. Define $\Delta_i(c) = \ell_c(\mathbf{1} - \mathbf{u}_i) - \ell_c(\mathbf{1})$. If $|d^2 \ell_c(\mathbf{1} - t\mathbf{u}_i)/dt^2| \le \kappa_{i,c}$ for $t \in [0, 1]$, Taylor's theorem gives

$$\Delta_i(c) = \omega_i(c) + \mathcal{R}_i(c), \qquad |\mathcal{R}_i(c)| \le \frac{\kappa_{i,c}}{2}. \tag{9}$$

The sign convention follows because deleting a write changes its gate in the negative direction.

For a retained set $\mathcal{I}$ of size B, put $h = \mathbf{m}_\mathcal{I} - \mathbf{1}$. If the Hessian operator norm is at most κ_c along the corresponding line segment, the integral remainder satisfies

$$\begin{aligned} \ell_c(\mathbf{m}_\mathcal{I}) - \ell_c(\mathbf{1}) &= \sum_{i \notin \mathcal{I}} \omega_i(c) + \mathcal{R}_c(\mathcal{I}), \\ |\mathcal{R}_c(\mathcal{I})| &\le \frac{\kappa_c}{2} \|h\|_2^2 = \frac{\kappa_c}{2}(L - B). \end{aligned} \tag{10}$$

Minimizing the linear term is equivalent to maximizing the sum of retained CWI values. Thus $\mathcal{I}_c^{\mathrm{CWI}} = \text{Top-B}_i\, \omega_i(c)$ solves the linearized problem. If the curvature condition holds along all feasible deletion paths, comparison with $\mathcal{I}_c^\star$ yields

$$\ell_c(\mathbf{m}_{\mathcal{I}_c^{\mathrm{CWI}}}) - \ell_c(\mathbf{m}_{\mathcal{I}_c^\star}) \le \kappa_c (L - B). \tag{11}$$

The bound increases with the number of omitted writes. In particular, first-order optimality does not establish reliable 25%-budget selection without controlling the remainder or measuring fidelity.

B.3 Selector Error and the Cost of Sharing

Proof of Theorem 2. Fix $c \in \mathcal{G}_g$ and abbreviate $\epsilon_g(c) = \max_i |\Omega_i(\mathcal{G}_g) - \omega_i(c)|$. By optimality of the top-B set under Ω,

$$\begin{aligned} \sum_{i \in \mathcal{I}_c^\star} \omega_i(c) - \sum_{i \in \widehat{\mathcal{I}}_g} \omega_i(c) &\le \sum_{i \in \mathcal{I}_c^\star} \Omega_i(\mathcal{G}_g) - \sum_{i \in \widehat{\mathcal{I}}_g} \Omega_i(\mathcal{G}_g) + 2B\epsilon_g(c) \\ &\le 2B\epsilon_g(c). \end{aligned}$$

The difference between omitted-utility sums equals this difference between retained-utility sums. Adding the two Taylor remainder bounds gives Eq. (7).

Algorithm 1 DELTAREC training and cache preparation

Require: Training contexts $\mathcal{D}$; preparation requests $\mathcal{P}$, including sparse training requests; checkpoint count T; retention ratio ρ; group count G
Ensure: Trained selector, fixed group information, prepared index cache, and fine-tuned ranking model
Full-history warm-up and CWI supervision
Obtain T warm-up checkpoints, reusing compatible checkpoints when available
Compute checkpoint-specific targets $\{\omega_i^{(t)}(c)\}$ on common sampled training contexts from $\mathcal{D}$ using Eq. (4)
Selector distillation
Fix selector features to a snapshot of the final warm-up checkpoint
while selector training is not complete **do**
    Sample checkpoint t with probability $\pi_t = t/\sum_{j=1}^{T} j$
    Sample examples with CWI targets from checkpoint t
    Update $r_{\boldsymbol{\phi}}$ by minimizing $\mathcal{L}_{\mathrm{sel}}$
end while
Set $s_{\boldsymbol{\phi}} = \sinh \circ r_{\boldsymbol{\phi}}$ and freeze the selector and its feature snapshot
Index-cache preparation
Obtain G utility groups from offline clustering of training-data CWI information
Fix the group information and its assignment rule a_{grp}
for each $(H, \mathcal{C}) \in \mathcal{P}$ **do**
    $n \leftarrow |H|$; obtain budget B from the experiment settings
    Assign $g(c) \leftarrow a_{\mathrm{grp}}(H, c)$ for each $c \in \mathcal{C}$
    Set $\mathcal{G}_g \leftarrow \{c \in \mathcal{C} : g(c) = g\}$ and $\mathcal{A} \leftarrow \{g : \mathcal{G}_g \neq \emptyset\}$
    Predict $\{\widehat{\boldsymbol{\omega}}(c) : c \in \mathcal{C}\}$ using $s_{\boldsymbol{\phi}}$
    for each $g \in \mathcal{A}$ **do**
        $\Omega_i(\mathcal{G}_g) \leftarrow |\mathcal{G}_g|^{-1} \sum_{c \in \mathcal{G}_g} \widehat{\omega}_i(c)$ for each event i
        $\widehat{\mathcal{I}}_g \leftarrow \mathrm{Top}_{B,\, i \in [n]}\, \Omega_i(\mathcal{G}_g)$
        Restore retained indices to chronological order
    end for
    Cache $g(c)$ and $\{\widehat{\mathcal{I}}_g\}_{g \in \mathcal{A}}$ under the matching index-cache key
end for
Sparse fine-tuning
Initialize the ranking model from the final warm-up checkpoint
Fine-tune the ranking model on sparse histories defined by the index cache
Keep the selector, its feature snapshot, and the group assignment rule fixed

For the embedding-diameter condition in the supplied analysis, assume $|\widehat{\omega}_i(a) - \omega_i(a)| \le \epsilon_{\mathrm{sel}}$ for every event and group member, and $\|J_i\|_2 \le M$. Let $d_g = \max_{a,b \in \mathcal{G}_g} \|e_a - e_b\|_2$. The factorization implies

$$|\omega_i(a) - \omega_i(c)| \le \frac{M d_g}{\tau}.$$

Consequently every member prediction lies within $\epsilon_{\mathrm{sel}} + M d_g/\tau$ of $\omega_i(c)$. If the aggregator lies between the smallest and largest member prediction, so does its output. Therefore

$$\epsilon_g(c) \le \epsilon_{\mathrm{sel}} + \frac{M d_g}{\tau}, \qquad \ell_c(\mathbf{m}_{\widehat{\mathcal{I}}_g}) - \ell_c(\mathbf{m}_{\mathcal{I}_c^\star}) \le 2B\left(\epsilon_{\mathrm{sel}} + \frac{M d_g}{\tau}\right) + \kappa_c(L - B). \tag{12}$$

□

This is a sufficient condition in ranking-embedding space. It does not prove that an unspecified predicted-profile clustering algorithm finds such groups, or that its groups minimize ranking loss. Likewise, relative-order distillation does not by itself establish the assumed uniform score error.

C IMPLEMENTATION DETAILS

C.1 CWI TARGET GENERATION AND SELECTOR TRAINING

Training stages and supervision. The offline pipeline consists of full-history warm-up, CWI target generation, selector distillation, index-cache preparation, and sparse fine-tuning, in this order. Compatible warm-up checkpoints can be reused. For CWI supervision, we evaluate each frozen

Algorithm 2 DELTAREC prefill and candidate-specific serving

Require: History H; candidate set $\mathcal{C}$; matching index cache containing $g(c)$ and retained indices for each nonempty group; fine-tuned GDR backbone; readout $R_{\boldsymbol{\theta}}$; scorer f_s
Ensure: Scores $\{\mathrm{Score}_c : c \in \mathcal{C}\}$
Prefill or cached state reuse
$\mathcal{A} \leftarrow \{g(c) : c \in \mathcal{C}\}$
if a matching state-cache entry is available **then**
  Load group states $\{\mathbf{S}^{(g)}\}_{g\in\mathcal{A}}$ and required readout tensors
else
  Gather $\{H_{\widehat{\mathcal{I}}_g}\}_{g\in\mathcal{A}}$ in chronological order with original positions
  $(\mathbf{X}_{\mathrm{pack}}, \mathbf{o}) \leftarrow \textsc{Pack}(\{H_{\widehat{\mathcal{I}}_g}\}_{g\in\mathcal{A}})$ ▷ $\mathbf{o}$: group offsets
  $\{\mathbf{S}^{(g)}\}_{g\in\mathcal{A}} \leftarrow \mathrm{GDR}(\mathbf{X}_{\mathrm{pack}}, \mathbf{o})$
  Retain required readout tensors and optionally cache the prefill outputs under the matching state-cache key
end if
Serving
for each candidate $c \in \mathcal{C}$ **do**
  $\mathrm{Score}_c \leftarrow f_s\Big(R_{\boldsymbol{\theta}}(\mathbf{S}^{(g(c))}), \mathbf{e}_c\Big)$
end for
return $\{\mathrm{Score}_c : c \in \mathcal{C}\}$

teacher on the same sampled training contexts and apply the intervention in Eq. (3) to the complete recurrent transition, including both decay and writing. The teacher parameters remain fixed while gradients are computed with respect to the intervention gates.

For HSTU, contexts follow the original training stream's history and target construction. The positive next item is the focal candidate; sampled negatives participate in the ranking loss but are not separately used as focal candidates. The selector receives event and candidate features, not their positive/negative labels.

Trajectory-Late sampling. Let $t \in \{1, \ldots, T\}$ index warm-up checkpoints in training order, and let $\omega_i^{(t)}(c)$ denote the CWI target computed using checkpoint t. Targets from different checkpoints remain separate supervision examples rather than being averaged. During selector training, checkpoints are sampled with probability $\pi_t = t/\sum_{j=1}^{T} j$. The two-checkpoint configuration uses $(\pi_1, \pi_2) = (1/3, 2/3)$; the four-checkpoint configuration uses $(0.1, 0.2, 0.3, 0.4)$. This gives more weight to later teacher states while retaining supervision from earlier stages.

Selector features and objective. The selector uses a frozen feature snapshot from the final warm-up checkpoint. Let $\mathbf{e}_i^{\mathrm{sel}}$ and $\mathbf{e}_c^{\mathrm{sel}}$ denote its event and candidate embeddings. The input is $\mathbf{z}_{i,c} = [\mathbf{e}_i^{\mathrm{sel}}, \mathbf{e}_c^{\mathrm{sel}}, \mathbf{e}_i^{\mathrm{sel}} \odot \mathbf{e}_c^{\mathrm{sel}}]$. These fixed selector embeddings are distinct from the ranking model embeddings updated during sparse fine-tuning.

An MLP $r_{\boldsymbol{\phi}}$ with one SiLU hidden layer of width $\max(8, d)$ and a scalar output predicts transformed CWI targets:

$$\mathcal{L}_{\mathrm{sel}}(i, c, t) = \mathrm{SmoothL1}\left(r_{\boldsymbol{\phi}}(\mathbf{z}_{i,c}), \mathrm{asinh}\Big(\omega_i^{(t)}(c)\Big)\right). \tag{13}$$

We optimize the MLP with AdamW. The utility predictor in Section 4.1 includes the inverse transformation, $s_{\boldsymbol{\phi}}(\mathbf{z}) = \sinh(r_{\boldsymbol{\phi}}(\mathbf{z}))$. Selection and grouping therefore use predictions in the original CWI scale.

Index-cache preparation. After distillation, we freeze the selector and its feature snapshot. We use fixed utility groups built offline from training-data CWI targets. For each history–candidate set, candidates are mapped to these existing groups rather than clustered again. The selector predicts candidate-specific utility profiles, which are then averaged within each nonempty group to select its shared sparse history. The index cache stores the candidate-to-group assignments and retained history indices, not recurrent states. It is prepared before sparse fine-tuning so that the selected histories can be reused without repeating utility prediction, group assignment, and selection.

Sparse fine-tuning. We initialize the ranking model from the final warm-up checkpoint and fine-tune it on the sparse histories defined by the prepared index cache. The selector, its feature snapshot, and the group assignment rule remain fixed, while the ranking model parameters are updated. The cached group assignments and retained indices are reused, but recurrent states are computed with the current ranking model during fine-tuning. This adapts the model to compressed histories without changing the selector's input features or the cached selections.

C.2 Write Budget and Utility-Based Grouping

Write budget. For Table 1, a history containing n valid events and nominal retention ratio ρ uses the retained count

$$B = \min\left(n, \max\left(\lceil \rho n \rceil, \min(32, n)\right)\right). \tag{14}$$

The 32-event minimum applies only to Table 1. Histories with at most 32 events are retained in full. Selection reserves a mandatory recent-event suffix $\mathcal{R}$ within the budget, with $|\mathcal{R}| \le B$, and fills the remaining positions by utility. Selected events are restored to chronological order and retain their original position indices.

Utility-based grouping. We build G fixed utility groups by clustering CWI information from the training data. This clustering is performed offline, and the resulting group information is fixed before request processing. Let a_{grp} denote the group assignment rule defined by this saved information. For a request with history H and candidate set $\mathcal{C}$, each candidate is mapped to an existing group as $g(c) = a_{\mathrm{grp}}(H, c)$. Request processing only applies this fixed rule; it does not run a new clustering procedure or compute new CWI targets.

The candidates assigned to group g in the current request form $\mathcal{G}_g = \{c \in \mathcal{C} : g(c) = g\}$. Let $\mathcal{A} = \{g : \mathcal{G}_g \neq \emptyset\}$ denote the groups used by this request. The selector predicts one utility profile per candidate, and the aggregator A in Section 4.2 is the mean utility of the candidates assigned to each nonempty group:

$$\Omega_i(\mathcal{G}_g) = \frac{1}{|\mathcal{G}_g|} \sum_{c \in \mathcal{G}_g} \widehat{\omega}_i(c), \qquad g \in \mathcal{A}. \tag{15}$$

Each nonempty group retains $\mathcal{R}$ and fills the remaining $B - |\mathcal{R}|$ positions with the largest $\Omega_i(\mathcal{G}_g)$ among events outside the suffix. Thus, the group definitions remain fixed, while group membership and the selected histories depend on the current request. The resulting candidate-to-group assignments $g(c)$ and retained indices $\{\widehat{\mathcal{I}}_g\}_{g \in \mathcal{A}}$ form the index-cache entry.

C.3 Sparse-State Execution and Caching

Packed prefill. Given a matching index-cache entry, prefill gathers the retained event features and packs the histories of the nonempty groups into a contiguous tensor. Group offsets mark the segments, allowing the fine-tuned GDR backbone to process retained events while maintaining independent states across groups. Groups with no candidates in the current request do not require state construction. The output contains the group recurrent states and any additional features required by the ranking readout. For multilayer adapters, this includes the per-layer states and, where required, features of the final retained event.

Candidate-specific scoring. Candidates reuse their assigned group state through $\mathrm{Score}_c = f_s(R_{\boldsymbol{\theta}}(\mathbf{S}^{(g(c))}), \mathbf{e}_c)$. The readout $R_{\boldsymbol{\theta}}$ and scoring function f_s are shared model components; f_s is not restricted to an inner product. For a request using $|\mathcal{A}|$ nonempty groups, recurrent state construction requires $|\mathcal{A}|B \le GB$ writes. This count does not include the complete cost of feature processing and candidate scoring.

Cache validity and state reuse. The index-cache key binds the history, exact candidate slate, frozen selector and its feature snapshot, fixed group information and assignment rule, and selection configuration. These inputs do not depend on the ranking-model parameters updated during sparse fine-tuning, so the index cache remains reusable as the ranking model changes. An index-cache hit avoids utility prediction, group assignment, and Top-B selection, but does not by itself skip prefill. An unmatched history or candidate slate requires a new index-cache entry using the existing

Table 4: Statistics of the datasets used in our experiments. History statistics are computed on validation instances with histories truncated to the latest 1,024 events.

Dataset	Users	Items	Interactions	Candidates	Mean Hist.	Median Hist.
ML-20M	138,493	26,744	20.0M	100	135.70	65
Amazon Books	811,227	695,762	∼11.7M	100	12.47	5
KuaiRand-1K	1,000	4,369,953	11.7M	32	1022.47	1024

groups, not new clustering. Changes to the selector, group information, or selection configuration also require rebuilding the affected index-cache entries.

The optional state cache stores the outputs of prefill, including recurrent states and required readout tensors. Its key additionally binds the ranking-model checkpoint and execution configuration. Unlike the index cache, a state-cache entry cannot be reused after the ranking model changes. State caches for serving are built with the fine-tuned ranking model. A matching state-cache hit skips recurrent state construction and directly supplies the cached outputs to serving, without changing the selected histories, group assignments, or scoring function.

D Experimental Details

D.1 Detailed Experimental Settings

D.1.1 More Details on Datasets

We evaluate on three public sequential recommendation benchmarks: ML-20M, Amazon Books, and KuaiRand-1K. For all datasets, interactions are ordered chronologically and models predict candidate items from their preceding user histories. For ML-20M and Amazon Books, we follow a leave-two-out evaluation protocol: the last interaction of each user is used for testing, the second-to-last interaction for validation, and all preceding interactions are available for training. Histories are truncated to the most recent 1,024 interactions in the main quality experiments.

ML-20M. ML-20M contains 20,000,263 timestamped ratings from 138,493 users. The processed interaction data contain 26,744 movies that constitute the effective recommendation vocabulary. Each event corresponds to a user's interaction with a movie, and the task is to rank the user's next interacted item based on the preceding interaction history.

Amazon Books. Amazon Books is constructed from the Amazon Reviews Books collection following the HSTU preprocessing protocol. Users and items occurring fewer than five times are removed, interactions are chronologically ordered, and categorical identifiers are remapped. The resulting benchmark contains 811,227 users and 695,762 items, with approximately 11.7M interactions. Each event corresponds to a user–book interaction, and the next interacted book is used as the prediction target. Compared with the other datasets, Amazon Books contains substantially shorter effective histories.

For ML-20M and Amazon Books, the original datasets do not provide a fixed candidate pool for our ranking setting. We therefore train an HSTU retrieval model and use it to generate the top-100 candidates for each validation and test instance while ensuring that the ground-truth item is included. The resulting candidate sets are generated once and frozen for all methods.

KuaiRand-1K. KuaiRand-1K is a sequential short-video recommendation dataset containing 1,000 users, 4,369,953 videos, and 11,713,045 logged interactions. Each event corresponds to a user–video exposure together with multiple behavioral feedback signals. We consider eight binary feedback tasks: click, like, follow, comment, forward, hate, long-view, and profile-enter. Unlike ML-20M and Amazon Books, KuaiRand-1K provides the candidate slate used by our evaluation pipeline; we preserve its 32-candidate evaluation setting instead of generating candidates using HSTU.

Table 4 summarizes the dataset statistics. History statistics are computed on validation instances after retaining at most the latest 1,024 events.

The corresponding 90th/95th percentiles of validation-history length are 332/518 for ML-20M, 22/39 for Amazon Books, and 1024/1024 for KuaiRand-1K. The three datasets therefore cover substantially different history regimes: Amazon Books is dominated by short and sparse histories, ML-20M contains moderately long histories, and KuaiRand-1K is strongly saturated at the 1,024-event context limit.

D.1.2 More Details on Metrics

For ML-20M and Amazon Books, we evaluate candidate ranking using HR@10 and NDCG@10. For KuaiRand-1K, we evaluate the eight behavioral prediction tasks using macro-GAUC.

Hit Rate (HR@10). HR@10 measures whether the held-out positive item occurs among the ten highest-scoring candidates. Let r_u denote the rank of the ground-truth item for instance u. We compute

$$\mathrm{HR@10} = \frac{1}{N}\sum_{u=1}^{N} \mathbb{1}[r_u \le 10]. \tag{16}$$

HR@10 measures successful retrieval but does not distinguish the ordering of successful items within the top-10.

Normalized Discounted Cumulative Gain (NDCG@10). NDCG@10 additionally rewards ranking the positive item closer to the top:

$$\mathrm{NDCG@10} = \frac{1}{N}\sum_{u=1}^{N} \frac{\mathbb{1}[r_u \le 10]}{\log_2(r_u + 1)}. \tag{17}$$

Because each ML-20M and Amazon Books evaluation instance contains one relevant target, the ideal DCG is one.

Group AUC (GAUC). GAUC evaluates candidate ordering separately within each user. This evaluates whether a model correctly orders positive and negative candidates for the same user rather than comparing scores across unrelated users.

For KuaiRand-1K, the final metric is the unweighted macro average across its eight behavioral tasks:

$$\mathrm{MacroGAUC} = \frac{1}{8}\sum_{t=1}^{8} \mathrm{GAUC}_t\,. \tag{18}$$

Thus, each behavioral task contributes equally to the reported metric.

D.1.3 More Details on Baselines

We compare with representative methods covering conventional attention, linear attention, and sparse history modeling.

- **HSTU** (Zhai et al., 2024) introduces specialized sequential transduction units for large-scale generative recommendation, providing a strong dense-attention backbone for long-sequence recommendation.
- **LinRec** (Liu et al., 2023) linearizes the attention computation to reduce its sequence-length complexity from quadratic to linear while retaining global history aggregation.
- **FuXi-Linear** (Ye et al., 2026) is a recent linear-attention model designed for efficient long-term sequential recommendation.
- **BlossomRec** (Ma et al., 2026) reduces long-history computation through sparse history processing rather than densely processing the complete historical sequence.
- **DeltaRec variants.** We apply DeltaRec to the corresponding recommendation backbones while retaining their embedding layers and feature processing. The original long-history attention computation is replaced by GDR, followed by CWI-based history selection and candidate state sharing.

D.1.4 More Details on Training and Implementation

Baselines use their released implementations and dataset-specific settings. Methods are compared using the same processed data, frozen candidate sets, and evaluation protocol. Main quality experiments use histories of at most 1,024 events; history-length scaling is evaluated separately. Model and optimizer configurations are provided in the released code.

For DeltaRec, training follows Appendix C.1: full-history GDR warm-up, trajectory-level CWI supervision, selector distillation, index-cache preparation, and sparse fine-tuning. The full-history warm-up model is denoted Full-GDR. The ranking model is fine-tuned on selected histories while the selector and its feature snapshot remain frozen. Comparisons with native backbones therefore evaluate the complete adapted systems rather than the selector alone.

The default grouping uses offline training-data CWI clusters and arithmetic-mean utility aggregation. Selection retains a mandatory recent suffix and fills the remaining budget by utility. The implementation also supports category-based grouping, primarily for KuaiRand-1K. Table 1 uses nominal retention ratio $\rho = 0.25$ with the 32-event minimum in Appendix C.2; the realized ratio is B/n for a nonempty history. This minimum is not applied to the retention-ratio sweep.

D.2 Fixed-Workload Timing and Cache Accounting

RQ2 uses 512 KuaiRand-1K validation requests arranged into 64 batches of eight. Each request contains exactly 2,048 real history events and 100 candidate exposures. All history events precede the earliest candidate timestamp. Eligibility filtering supplies the required history length without padding or duplicated events. The same requests and batches are used across execution variants.

Measurements use an NVIDIA L40S GPU and chunk size 64, with $\rho = 0.25$ for the sparse K-state and G-state configurations. Each configuration has 640 measured runs after three warm-up iterations. We report mean, P50, and P95 latency in milliseconds per batch of eight requests. Overall latency is mean prefill plus mean cached-serving latency, not the sum of their percentiles. Cache payload is measured in MiB/request using BF16 representations and is distinct from peak GPU memory.

Prefill starts from prepared selection metadata and includes gathering retained events, packing sparse histories, and constructing recurrent states. Cached serving reuses these states for candidate-specific scoring. Utility prediction, group assignment, and history selection are outside both timed stages. The benchmark measures cached-stage execution, not end-to-end latency including metadata preparation; cache-hit rates and preparation costs for unseen requests are not measured.

The 1-state variant shares one history state across all candidates, K-state constructs an independent state for each candidate, and G-state constructs group-shared states. With B retained events per sparse state, K-state requires KB recurrent writes and G-state requires at most GB. Reported grouping speedups use K-state execution as the reference.

D.3 Details about Model Analysis Settings

D.3.1 History-Selection Ablations

Table 3 compares DeltaRec with four alternatives. Full-length GDR processes all available history events. Candidate similarity replaces CWI utility with event–candidate similarity. Recency-only selects recent events without candidate conditioning, while Rank-only uses candidate-independent importance scores.

Variants use separately trained systems and their best recorded validation checkpoints under the same dataset-specific evaluation protocol. This is a system-level comparison of selection policies, not a selector-only intervention on a common frozen model. The table reports HR@10 for ML-20M and Amazon Books and Macro-GAUC for KuaiRand-1K.

Table 5: Fixed-checkpoint validation at nominal retention ratio $\rho = 0.10$. ML-20M and Amazon Books use HSTU-retrieved candidates. Parentheses report relative changes from the corresponding backbone in this table.

	ML-20M		Amazon Books		KuaiRand-1K
Method	HR@10	NDCG@10	HR@10	NDCG@10	Macro-GAUC
HSTU (Zhai et al., 2024)	0.1496	0.0753	0.0441	0.0238	0.5348
DELTAREC (HSTU)	**0.2308** (+54.28%)	**0.1258** (+67.07%)	**0.0445** (+18.04%)	**0.0240** (+20.00%)	**0.5414** (+1.23%)
FuXi-Linear (Ye et al., 2026)	0.1857	0.1008	0.0350	0.0185	0.5222
DELTAREC (FuXi-Linear)	**0.2121** (+14.22%)	**0.1128** (+11.90%)	**0.0409** (+16.86%)	**0.0226** (+22.16%)	**0.5322** (+1.91%)
BlossomRec (Ma et al., 2026)	0.2114	0.1135	0.0426	0.0218	**0.5387**
DELTAREC (BlossomRec)	**0.2196** (+3.88%)	**0.1202** (+5.90%)	**0.0458** (+7.51%)	**0.0277** (+27.06%)	0.5355 (-0.59%)

Table 6: Fixed-checkpoint validation at nominal retention ratio $\rho = 0.50$. ML-20M and Amazon Books use HSTU-retrieved candidates. Parentheses report relative changes from the corresponding backbone in this table.

	ML-20M		Amazon Books		KuaiRand-1K
Method	HR@10	NDCG@10	HR@10	NDCG@10	Macro-GAUC
HSTU (Zhai et al., 2024)	0.1496	0.0753	0.0441	0.0238	0.5348
DELTAREC (HSTU)	**0.2441** (+63.17%)	**0.1335** (+77.29%)	**0.0445** (+18.04%)	**0.0241** (+20.50%)	**0.5366** (+0.34%)
FuXi-Linear (Ye et al., 2026)	0.1857	0.1008	0.0350	0.0185	0.5222
DELTAREC (FuXi-Linear)	**0.2319** (+24.88%)	**0.1243** (+23.31%)	**0.0449** (+28.29%)	**0.0246** (+32.97%)	**0.5403** (+3.47%)
BlossomRec (Ma et al., 2026)	0.2114	0.1135	0.0426	0.0218	**0.5387**
DELTAREC (BlossomRec)	**0.2272** (+7.47%)	**0.1228** (+8.19%)	**0.0522** (+22.54%)	**0.0312** (+43.12%)	0.5209 (-3.30%)

D.3.2 PARAMETER SENSITIVITY

Figure 4(a,b) varies the group count G at fixed $\rho = 0.25$, while Figure 4(c,d) varies ρ with a fixed grouping configuration. The ranking checkpoint and selector are not retrained for individual settings. We report Macro-GAUC and prefill latency to examine the state and history budgets separately.

The KuaiRand-1K retention sweep uses the category-based grouping option with a fixed mapping and varies ρ from $1/512$ to 1. It does not use the 32-event minimum from Table 1. The mandatory recent suffix counts toward the retained budget, and predicted utility fills the remaining positions. When the suffix exhausts the budget, selection is determined by recency. This sweep tests budget sensitivity under a fixed grouping option rather than comparing grouping signals.

E MORE EXPERIMENTAL RESULTS

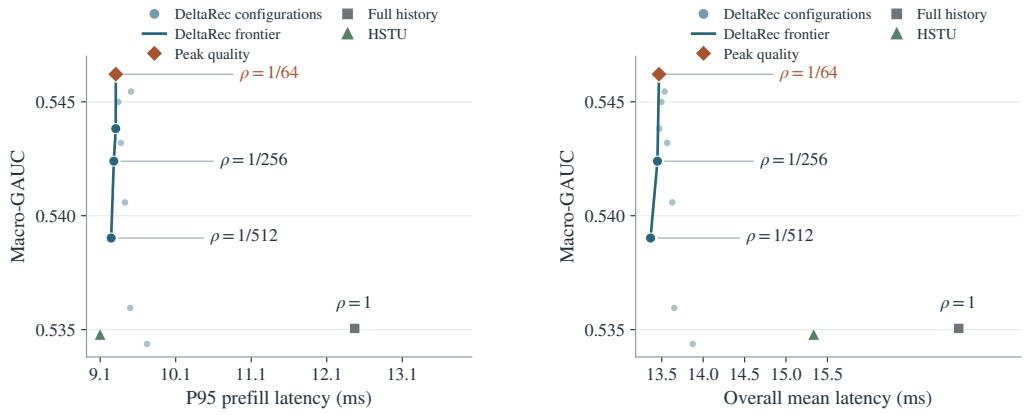
E.1 CROSS BUDGET GENERALIZATION

We evaluate sparse models fine-tuned at nominal retention ratio $\rho = 0.25$ using $\rho = 0.10$ and $\rho = 0.50$, without retraining the ranking model or selector. Retained histories are selected for each evaluation budget. This tests cross-budget reuse of fixed checkpoints; the selector itself is trained on CWI targets rather than a retention-specific objective.

Relative to the baselines reported in Tables 5 and 6, DELTAREC improves 14 of 15 backbone–dataset–metric comparisons at each ratio. HSTU and FuXi-Linear improve on all three datasets at both ratios. BlossomRec improves on ML-20M and Amazon Books but falls below its KuaiRand-1K baseline by 0.59% at $\rho = 0.10$ and 3.30% at $\rho = 0.50$. Thus, cross-budget reuse is effective in most reported comparisons, but retaining more history does not consistently improve quality.

E.2 MORE RESULTS ON QUALITY–EFFICIENCY TRADE-OFFS (RQ4)

Figure 5 pairs Macro-GAUC with P50 prefill latency across the retention configurations above. The trained checkpoint, grouping configuration, and validation protocol remain fixed while ρ changes. A



(a) P95 prefill latency.

(b) Overall mean latency.

Figure 6: Quality–efficiency trade-offs across retention ratios of the same sparse-trained model. Overall mean latency is mean prefill plus mean cached-serving latency.

configuration is Pareto-dominated if another tested configuration has no lower quality and no higher latency, with a strict improvement in at least one dimension.

The frontier summarizes the evaluated operating points. The $\rho = 1$ reference is the full-retention evaluation of the same sparse-trained checkpoint. The comparison therefore concerns retention configurations of that checkpoint rather than all full-history alternatives.

Figure 6 evaluates the same retention configurations using P95 prefill latency and overall mean latency. Several compressed configurations achieve higher Macro-GAUC and lower measured latency than the $\rho = 1$ evaluation of the same sparse-trained model. The peak-quality configuration at $\rho = 1/64$ remains on the observed frontier under both metrics. Overall mean latency includes prefill and cached serving, but not metadata preparation.

E.3 Scaling Analysis with History Length

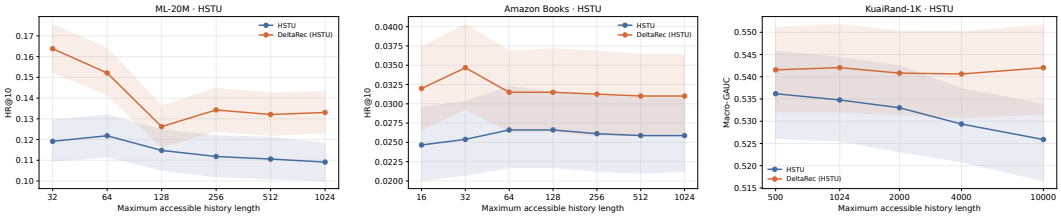
We vary the maximum available history length at evaluation while holding the trained checkpoints, evaluation users, and candidate sets fixed. Figure 7 compares each native backbone with its DeltaRec variant, with HSTU in the top row and FuXi-Linear in the bottom row. We report HR@10 on ML-20M and Amazon Books and Macro-GAUC on KuaiRand-1K.

With HSTU, both methods decline after an early peak on ML-20M, although DeltaRec partially recovers and remains above the baseline. On Amazon Books, early gains give way to a slight decline and nearly flat tails. On KuaiRand-1K, native HSTU deteriorates as history grows, whereas DeltaRec remains relatively stable, widening the quality gap.

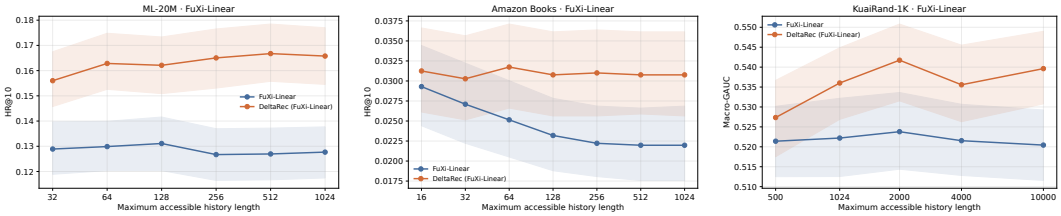
With FuXi-Linear, DeltaRec improves before plateauing on ML-20M, while the native baseline gains little and eventually declines slightly. On Amazon Books, DeltaRec remains stable as the baseline deteriorates. On KuaiRand-1K, both methods reach their highest observed quality at an intermediate length; further expansion produces fluctuations for DeltaRec and a decline for the baseline. History selection mitigates degradation in several settings, but longer history does not consistently improve quality.

E.4 Cross-Backbone Generalizability of Trained Selector

We evaluate all nine source–target combinations among HSTU, FuXi-Linear, and BlossomRec on the same KuaiRand-1K requests at $L = 1024$, using fixed category-based groups and common retention and recent-suffix settings. Each source selector uses its own frozen feature space to select history positions, which are then encoded and scored by the frozen target GDR encoder and ranking head. Transfer is compared with the target's own selector under the same evaluation configuration.

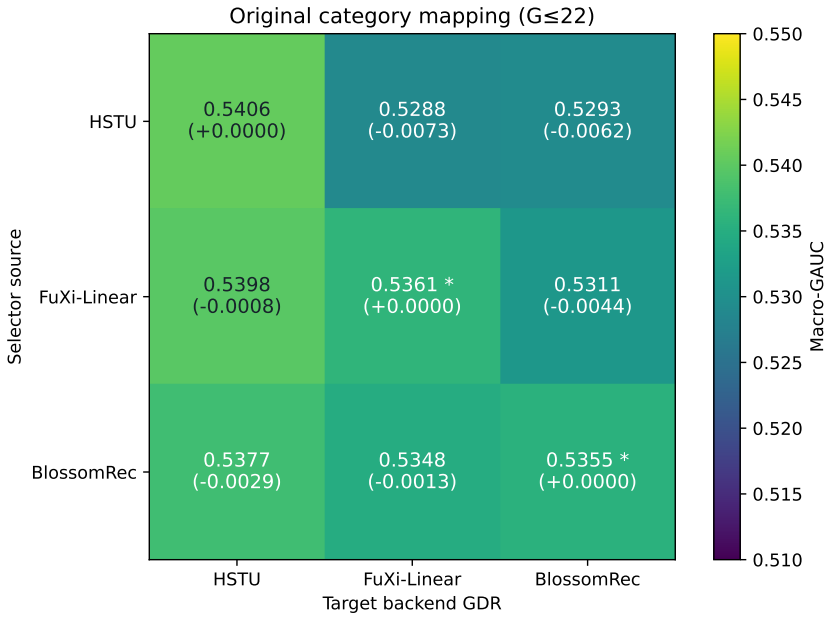


(a) HSTU on ML-20M (b) HSTU on Amazon Books (c) HSTU on KuaiRand-1K



(d) FuXi-Linear on ML-20M (e) FuXi-Linear on Amazon Books (f) FuXi-Linear on KuaiRand-1K

Figure 7: History-length scaling with fixed checkpoints. The top and bottom rows show HSTU and FuXi-Linear, respectively. Columns show ML-20M, Amazon Books, and KuaiRand-1K. Metrics are HR@10 for the first two datasets and Macro-GAUC for KuaiRand-1K.



Parentheses: measured change from same-target diagonal. * Matched Table 1 value.

Figure 8: Cross-backbone selector transfer on KuaiRand-1K with category-based grouping. Rows indicate selector sources and columns indicate target GDR backends. Parentheses report changes from the same-column diagonal, which uses the target backbone's own selector.

Figure 8 shows that the HSTU selector achieves 0.5293 Macro-GAUC on the BlossomRec target, compared with 0.5355 using the target's own selector, a difference of −0.0062. Selector reuse is possible without retraining, but quality preservation depends on the source–target pair.