GENERATIVE CONTEXTUAL RANKING: REQUEST-LEVEL GLOBAL OPTIMIZATION OF RECOMMENDATION SLATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern recommender systems typically score user–item pairs independently, even though users consume recommendations as ordered slates. This mismatch prevents rankers from modeling how an item's utility depends on earlier recommendations and can place redundant content close together, reducing diversity and degrading user experience. To solve this problem, we propose Generative Contextual Ranking (GCR), which replaces independent scoring with autoregressive, request-level slate construction. At each step, GCR selects an item conditioned on the user's history and the items already placed, capturing inter-item effects such as topical redundancy, complementarity and position dependence. GCR is a lightweight architectural change that can be inserted at any ranking stage without a full-stack rewrite. Its principal cost is the latency of autoregressive decoding, which we offset with a set of serving optimizations that improve throughput by over 10×. Experiments on a large-scale short-form video platform and on a public benchmark demonstrate GCR can improve offline prediction quality and deliver statistically significant online gains.

1 INTRODUCTION

Modern recommendation systems are predominantly built on a pointwise user–item scoring paradigm (Covington et al., 2016; Zhai et al., 2024). In the ranking stage, each candidate is evaluated in isolation. The model sees a single user–item pair, predicts multiple engagement events, such as like, share, etc, and reduces them to one score by which candidates are then ranked. This independence is precisely what has made the paradigm so successful. Because no candidate's score depends on any other, scoring is embarrassingly parallel, can be sharded and cached freely, and maps cleanly onto the multi-stage retrieval-and-ranking funnel that powers large-scale platforms.

That assumption, however, does not match how recommendations are delivered. To balance serving cost, latency, throughput, and user experience, production systems rarely return one item at a time. They return multiple items in a single request, commonly referred to as a slate (Wang et al., 2019). Users consume the slate in order, so the value of an item depends on what preceded it. A user who enjoys cat videos, for instance, will engage far less with the fifth consecutive cat video than with the first, even though all five match that user's interests equally well. The items, however, are still selected independently. Each is scored as though it were the first item in the request, with no awareness of the others chosen alongside it.

This mismatch between pointwise scoring and slate-level consumption introduces two limitations. First, the model cannot exploit the full recommendation context, since the prediction for an item is made without knowledge of what has already been shown in the same request. Second, near-duplicate items receive near-identical scores and are therefore ranked adjacent to one another. Production systems patch this with a policy layer that runs downstream of ranking and applies diversity rules to push similar items apart. Such corrections are blunt. They reorder a list whose composition is already fixed, and they offer no guarantee that the resulting slate is a coherent sequence rather than a merely spread-out one.

Recent generative recommendation methods offer a promising alternative by casting recommendation as sequence generation. Systems such as OneRec generate an entire request end-to-end,

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.

conditioning each item on the full request context (Deng et al., 2025). These approaches, however, typically decode over semantic IDs (SIDs) (Rajput et al., 2023), intermediate tokens that must be carefully constructed and then mapped back to real items. Their effectiveness is therefore tightly coupled to the quality of the SID vocabulary, which adds substantial modeling complexity and leaves a gap between what the model emits and what the system can serve. Adopting such systems also requires redesigning the existing retrieval, ranking model and the serving stack, a cost few production platforms can absorb in a single step.

To address these challenges, we propose **Generative Contextual Ranking (GCR)**, which captures the slate-level, context-aware benefits of generative modeling while remaining compatible with existing recommendation infrastructure. Instead of scoring items independently, GCR constructs the output slate autoregressively. At each step, it selects the next item conditioned on both the user interaction history and the items already selected for the current request. GCR decodes directly over real candidate items rather than intermediate SIDs, so every step emits something the system can serve and no separate token vocabulary must be learned or maintained. GCR also executes the policy layer inside the decoding loop, so the slate satisfies diversity and product rules as it is built rather than through post-hoc correction alone.

Our approach offers four main advantages.

- **Request-level global optimization.** GCR conditions each item selection on the evolving slate context and therefore optimizes the request as a whole. This directly addresses the topic duplication problems that pointwise systems can only patch heuristically.
- **Stage-agnostic deployment.** The same formulation can serve as early-stage ranking (ESR), late-stage ranking (LSR), or post-LSR reranking. GCR can be inserted at any one of these stages, or subsume several of them, allowing teams to adopt it where it delivers the most value.
- **Decoding over real items.** GCR decodes directly over real candidate item IDs rather than intermediate SIDs. It therefore inherits neither the modeling complexity of SID construction nor its dependence on SID quality, and every decoding step emits an item the system can serve. This lets GCR attach to an existing recommendation stack and supports step-by-step, low-risk adoption rather than a full-stack rewrite.
- **Capacity-neutral serving.** We show that autoregressive slate construction with an in-model policy layer is feasible at industrial scale. A set of serving optimizations recovers over $10\times$ in throughput, allowing GCR to replace the production ranker on the same hardware.

2 Related Work

2.1 Sequential and Context-Aware Ranking

Sequential recommendation models use self-attention, target-aware attention, and memory mechanisms to encode users' interaction histories (Kang & McAuley, 2018; Sun et al., 2019; Zhou et al., 2018; Xia et al., 2023; Zhai et al., 2024; Pi et al., 2019). These methods produce rich representations of user preferences but typically retain a pointwise prediction objective: each candidate is scored against the user context independently of the other items selected for the same slate.

Context-aware reranking methods incorporate cross-item interactions using recurrent encoders, self-attention, or personalized set representations (Ai et al., 2018; Pei et al., 2019; Pang et al., 2020; Li et al., 2022; Xi et al., 2022). Permutation-based approaches instead evaluate or select among candidate orderings (Shi et al., 2023). Most of these methods either refine all candidate scores in a single pass or choose among slates constructed in advance. GCR differs by constructing the deployed slate sequentially: at every step, each remaining candidate is evaluated in the context of the items actually selected at preceding positions.

2.2 Generative Slate Construction

Prior work has approached slate optimization through reinforcement learning, autoregressive generation, and non-autoregressive generative modeling. SlateQ (Ie et al., 2019) makes reinforcement

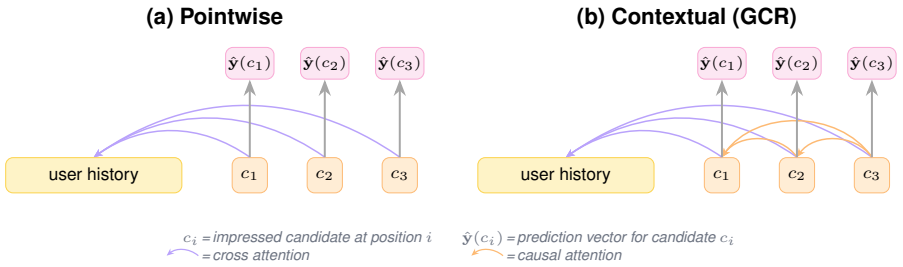


Figure 1: Attention-pattern comparison during teacher-forced training over the ordered impressed sequence $\mathcal{I}$. **(a) Pointwise:** each impressed candidate c_i cross-attends only to the user history, so $\hat{\mathbf{y}}(c_i)$ is independent of the other impressed candidates. **(b) Generative Contextual Ranking:** each impressed candidate c_i additionally causally attends to the preceding impressed candidates $c_1, \ldots, c_{i-1}$, so $\hat{\mathbf{y}}(c_i)$ is conditioned on the logged prefix $\mathcal{I}_{<i}$.

learning tractable by decomposing slate utility into per-item value estimates under a user-choice model. Autoregressive methods such as Seq2Slate and GFN4Rec construct ordered slates through sequential selection (Bello et al., 2019; Liu et al., 2023). More recent systems improve generation efficiency or preference alignment using multi-token prediction and hierarchical planning (Lin et al., 2025; Pang et al., 2025).

Alternative formulations include context-wise generative reranking, generator– evaluator frameworks, conditional variational models, and diffusion-based slate generation (Feng et al., 2021; Ren et al., 2024; Jiang et al., 2019; Tomasi et al., 2025). These approaches differ in whether they generate a single slate directly, produce several alternatives, or refine candidate scores jointly. GCR combines direct selection from an existing candidate set with causal attention over the complete selected prefix. Its additional focus is production integration: configurable, non-learned policies are evaluated inside every decoding step so that product constraints influence slate construction rather than being applied only before or after generation.

2.3 GPU-Native Recommendation Serving

GPU-native serving systems move traditionally external recommendation operations into the model graph to reduce communication and synchronization overhead. SilverTorch (Xue et al., 2026), for example, represents approximate nearest-neighbor retrieval and feature filtering as in-model tensor operations, replacing standalone CPU services at the retrieval stage.

GCR applies this principle to autoregressive ranking. Unlike retrieval components that execute once per request, GCR's policy layer must be evaluated after every decoding step. Keeping this logic outside the model would introduce repeated CPU–GPU synchronization or service calls. GCR therefore expresses value aggregation and policy constraints as GPU-resident tensor operations and executes them within the same compiled graph as the decoder.

3 Problem Formulation

Generative Contextual Ranking (GCR) is agnostic to its position in a multi-stage recommendation pipeline: it can serve as an Early-Stage Ranker (ESR), Late-Stage Ranker (LSR), or post-LSR reranker. Accordingly, the input may contain tens to thousands of candidates, while the output slate may contain tens to hundreds of items.

Formally, let $\mathcal{H} = (h_1, \ldots, h_n)$ denote a user's interaction history, ordered chronologically, and let $\mathcal{C} = \{c_1, \ldots, c_K\}$ denote the candidate set of K supplied by the upstream stage. GCR constructs an ordered slate $\mathcal{O} = (o_1, \ldots, o_S)$, where S is the slate length and $S \leq K$. We define $\mathcal{O}_{<s} = (o_1, \ldots, o_{s-1})$ as the slate prefix at step s, and $\mathcal{C}_s = \mathcal{C} \setminus \mathcal{O}_{<s}$ as the candidates still remaining at that step. The selection of o_s depends on both the UIH $\mathcal{H}$ and the prefix $\mathcal{O}_{<s}$.

3.1 Pointwise Ranking

A conventional ranker predicts user engagement with each candidate independently. Let $\mathcal{F}_u$ denote user-side features and $\mathcal{F}_{c_i}$ denote the features of candidate c_i. For J engagement tasks, the model produces

$$\begin{aligned}\hat{\mathbf{y}}(c_i) &= [\hat{y}_1(c_i), \ldots, \hat{y}_J(c_i)] \\ &= f_\theta(c_i \mid \mathcal{H}, \mathcal{F}_u, \mathcal{F}_{c_i}), \quad \forall\, c_i \in \mathcal{C}.\end{aligned} \tag{1}$$

where $\hat{\mathbf{y}}(c_i)$ contains predictions for outcomes such as likes, watch time, and shares.

Because the prediction for c_i does not depend on the other selected items, its estimated utility is unchanged by the composition or ordering of the slate. Consequently, inter-item effects must be handled by downstream reranking or rule-based adjustments.

3.2 Contextual Prediction

GCR instead updates each candidate's predictions as the slate is constructed. At step s, it computes

$$\hat{\mathbf{y}}_s(c_i) = f_\theta\left(c_i \mid \mathcal{H}, \mathcal{F}_u, \mathcal{F}_{c_i}, \mathcal{O}_{<s}\right), \qquad c_i \in \mathcal{C}_s. \tag{2}$$

Thus, the predicted utility of a candidate may change as the slate prefix evolves. This conditioning enables GCR to model inter-item effects that are invisible to pointwise rankers. For example, the model can reduce the predicted engagement of a candidate when topically similar content already appears in $\mathcal{O}_{<s}$, or increase it when the candidate complements the preceding items. Figure 1 illustrates this distinction.

3.3 Policy Layer

At each decoding step the model produces an engagement-prediction vector $\hat{\mathbf{y}}_s(c)$ for every remaining candidate, while selection requires a single comparable score. The component that performs this reduction exists in most industrial ranking systems, though its scope and name vary across platforms. We refer to it as the *policy layer*. In our design it has two stages. First, a value model (VM) maps the J predictions to a scalar reflecting the relative importance of each engagement task. Second, configured product and diversity constraints adjust that scalar using the candidate and the slate prefix $\mathcal{O}_{<s}$. We denote the resulting score as $\mathrm{PL}_s(c)$. Greedy decoding then selects

$$o_s = \operatorname*{argmax}_{c \in \mathcal{C}_s} \mathrm{PL}_s(c). \tag{3}$$

Infeasible candidates receive a score of $-\infty$ and can never be selected.

4 System Design

Generative Contextual Ranking (GCR) uses an encoder–decoder architecture. The encoder summarizes the user interaction history and user-side features into a contextual user representation. The decoder combines that representation with candidate features and the slate prefix $\mathcal{O}_{<s}$ to produce contextual engagement predictions. Training and inference exercise this architecture differently. During training, teacher forcing over logged slates allows all positions to be predicted in a single parallel forward pass. At inference the decoder constructs the slate autoregressively, with the in-model policy layer reducing the predictions to a single score and selecting one candidate at each step. Because every step is a full forward pass, serving cost rather than model quality becomes the binding constraint, and the last part of this section describes the optimizations that make the design deployable.

4.1 Model Architecture

4.1.1 Input Representation

The model consumes three inputs, namely the user interaction history (UIH), user-side features, and candidate-side features. The UIH is a chronological sequence of previously engaged items together with their associated engagement signals, and each item ID is mapped to an embedding through

a lookup table. User-side features remain constant across candidates within a request, whereas candidate-side features vary by candidate. Sparse and dense features on each side are processed by their respective modules and fused into a user representation and a per-candidate representation.

4.1.2 Encoder and Decoder Architecture

The encoder applies N_e Hierarchical Sequential Transduction Unit (HSTU) (Zhai et al., 2024) self-attention layers to the concatenation of the n UIH tokens $h_1, \ldots, h_n$ and the m fused user-side tokens $u_1, \ldots, u_m$. Its output is the contextual user representation that conditions the decoder.

The decoder applies N_d HSTU-based layers to the fused candidate representations. It operates directly over the candidates in $\mathcal{C}$ rather than generating intermediate identifiers. Each candidate attends in two ways. It cross-attends to the encoder output, which captures its affinity with the user context, and it causally attends to the slate prefix, which captures its relationship to the items already selected. At inference this applies to the remaining candidates $\mathcal{C}_s$ at step s, and at training it applies to all impressed positions at once under a causal mask. The training-time attention pattern is shown in Figure 2(c).

4.2 Training

Let $\tilde{\mathcal{O}} = (\tilde{o}_1, \ldots, \tilde{o}_M)$ denote the ordered sequence of M impressed candidates in a logged request, that is, the items actually shown to the user, where each $\tilde{o}_s \in \mathcal{C}$. A training example contains only $\tilde{\mathcal{O}}$, with $M < K$, rather than the full candidate pool the model decodes over at inference. Because the logged sequence is known, teacher forcing conditions the prediction at position s on the logged prefix $\tilde{\mathcal{O}}_{<s} = (\tilde{o}_1, \ldots, \tilde{o}_{s-1})$. A causal attention mask allows the model to compute predictions at all M positions in a single parallel forward pass, as shown in Figures 2(a) and 2(c).

4.2.1 Prediction Head

The HSTU decoder output and the dense-feature projection are combined and passed through a shared expert layer followed by a Multi-gate Mixture-of-Experts (MMoE) (Ma et al., 2018; Shazeer et al., 2017). Learned gates over the shared experts produce one representation per task, each of which is processed by a dedicated prediction head to yield $\hat{\mathbf{y}}$.

4.2.2 Training Objective

The objective averages the loss within each task and then sums across tasks,

$$\mathcal{L} = \sum_{j=1}^{J} \frac{\lambda_j}{B_j} \sum_{b=1}^{B_j} \ell_j\big(\hat{y}_{b,j}, y_{b,j}\big). \tag{4}$$

In Equation 4, B_j is the number of batch elements carrying a valid label for task j, λ_j is the per-task weight, and ℓ_j is binary cross-entropy for classification tasks and squared error for regression tasks. Averaging within a task before summing keeps the contribution of each task independent of how many elements happen to be labelled for it.

4.3 Inference

At inference, GCR applies Equation 3 sequentially until the slate is complete. Each selected item is removed from the candidate pool and appended to the slate prefix, as shown in Figure 2(b). We use greedy decoding for its latency efficiency.

4.3.1 Training–Inference Mismatch

Although teacher forcing allows predictions at all training positions to be computed in parallel, it creates a mismatch with autoregressive inference. Training predictions condition on the logged prefix $\tilde{\mathcal{O}}_{<s}$, whereas inference predictions condition on the prefix $\mathcal{O}_{<s}$ produced by the model's own selections. This distribution shift is known as exposure bias (Ranzato et al., 2016). Three properties of our setting limit its practical impact.

- **Closed candidate set.** The model selects from a fixed pool of K candidates already filtered by the upstream ranker. Any item the model places at a given position during inference was a plausible occupant of that position during training, so the shift is limited to permutations of a small known set rather than an open vocabulary.
- **Conditional prediction, not imitation.** The training objective predicts engagement *given* a prefix rather than reproducing the logged ordering. Learned conditionals, such as engagement decreasing after topically redundant content, hold regardless of which system assembled the prefix.
- **Short decoding horizon.** A shorter slate incurs less drift, whereas a longer slate amortizes computation and saves resources. We tune the slate length to balance these two effects, bounding the horizon over which errors can compound. The appropriate slate length depends on the stage at which GCR is deployed, and needs to be tuned.

4.4 In-Model Policy Execution

The policy layer defined in Section 3.3 is evaluated at every decoding step. In conventional serving architectures, this logic runs outside the neural model graph, often on the CPU or in a separate service. Retaining this boundary would introduce one synchronization or RPC boundary per decoding step—up to S round trips for a slate of length S—making autoregressive serving impractical under production latency constraints.

We instead implement both policy stages as vectorized, GPU-resident tensor operations in the same compiled serving graph as the decoder. This eliminates calls to external policy services while preserving the scoring and constraint behavior.

4.5 Inference Optimization

Autoregressive decoding multiplies the cost of a single forward pass by the slate length S. A direct implementation of GCR with an in-model policy layer runs roughly an order of magnitude below the throughput of the production system it replaces, so closing that gap is a deployment requirement rather than an optimization. We do so through four techniques, namely KV caching, batched inference, multi-forward dynamic pagination, and ahead-of-time graph compilation.

4.5.1 KV Cache

The encoder output and each candidate's cross-attention representation over the UIH do not change across decoding steps and can be reused. We also cache the key and value projections for the existing slate prefix. At each step, the selected item's key–value state is appended to the cache, and the remaining candidates are rescored against the updated prefix. The only work that is genuinely new at each step is the attention of the remaining candidates to the item just added.

4.5.2 Batched Inference

Batched inference processes multiple requests together to increase throughput. A batch is dispatched when it reaches the maximum batch size or when the maximum waiting time elapses, whichever occurs first. The waiting time bounds the queueing delay that batching would otherwise add to latency.

4.5.3 Multi-Forward Dynamic Pagination

The number of decoding steps a request should use is not fixed. Fewer steps produce shorter slates, which lowers per-request latency but raises request frequency, whereas more steps amortize the encoder computation and the KV cache across a longer slate and lower total capacity cost. Rather than execute the maximum number of steps for every request, multi-forward dynamic pagination provides length-specific execution paths over the same model weights, so no separate models are required. Each request takes the path matching its requested slate length, allowing the trade-off above to be set per surface or per user cohort. Section 4.5.4 describes how each supported path is compiled for serving.

4.5.4 AHEAD-OF-TIME GRAPH COMPILATION

We compile the full serving graph into optimized native code before deployment (Ansel et al., 2024), removing Python dispatch overhead and enabling operator fusion and shape-specialized kernels. Ahead-of-time compilers assume a static dataflow graph over fixed-shape tensors, and GCR violates that assumption in three ways. The decoding loop is a loop, the configured policy operations carry data-dependent branches, and candidate counts and configuration sizes are known only at run time. We address these in turn.

Static unrolling. Each supported slate length fixes the number of decoding steps, so the corresponding execution path from Section 4.5.3 is unrolled at compile time into a single fixed-shape artifact.

Branch-free rule evaluation. Conditional policy logic is expressed as data-parallel tensor algebra. Conditions become boolean masks, alternatives are combined with elementwise selection, and constraints apply additive or multiplicative score adjustments, including the $-\infty$ masking of hard constraints. This removes data-dependent branching from the compiled graph.

Bounded dynamic shapes. Candidate counts and policy-configuration sizes vary at run time, which would otherwise force the compiler onto a slow dynamic-shape path. We annotate both with compile-time upper bounds, which lets the compiler emit specialized kernels while still accepting variable-sized inputs at run time.

Together, these techniques place the per-request-configurable policy layer and the neural decoder inside one compiled serving graph. Section 5.2.3 reports their cumulative throughput impact.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate Generative Contextual Ranking on two datasets, an in-house dataset from a large-scale short-form video platform, and KuaiRand (Gao et al., 2022), a public recommendation benchmark. We run three evaluations on the in-house dataset. The first measures offline prediction quality, comparing context-aware GCR against the context-free pointwise LSR baseline, with both models trained on the same data. The second measures online engagement, deploying the full system as a post-LSR reranking stage and A/B testing it against the existing late-stage reranker over the same candidate set. The third measures serving throughput under controlled load testing on the production traffic mix, quantifying the contribution of each inference optimization from Section 4.5 and establishing that GCR can be deployed within the same GPU capacity as the system it replaces. On KuaiRand we compare GCR against depth-matched HSTU baselines trained under identical settings, which isolates the contribution of slate-aware conditioning in a setting others can reproduce. All metrics reported from the in-house dataset are directionally accurate but lightly perturbed for confidentiality, whereas KuaiRand results are reported unmodified. All offline and online results are statistically significant unless otherwise noted.

5.2 IN-HOUSE DATASET

5.2.1 OFFLINE RESULTS

Prediction quality is measured using Normalized Entropy (NE) (He et al., 2014), defined as the ratio of the model's cross-entropy loss to the background entropy of the label y,

$$\mathrm{NE}(p, y) = \frac{\mathrm{CrossEntropy}(p, y)}{\mathrm{Entropy}(y)} \tag{5}$$

where p is the model's predicted probability. Lower NE indicates better performance, and $\Delta\mathrm{NE} < 0$ means the model outperforms the baseline. We use NE rather than AUC because NE directly reflects calibration quality, and because it is the standard offline metric for engagement prediction in industrial systems owing to its consistency with online outcomes. At our scale, a relative NE reduction of 0.05% is considered statistically and practically significant.

We compare GCR against the pointwise LSR baseline on a held-out evaluation set spanning billions of samples across three primary engagement prediction tasks, denoted Tasks A, B, and C. Both models are trained on the same data, which makes per-task NE a well-defined comparison. A direct offline comparison against the listwise evaluator is not feasible, because it is trained on list-level aggregate labels such as total watch time over a slate, drawn from a separate data pipeline, which renders per-task NE ill-defined between the two systems.

Table 1 reports NE gains broken down by slate position, which isolates the contribution of contextual conditioning. Positions are numbered from one, so position 1 carries no preceding context. At position 1, GCR nonetheless achieves modest NE improvements over the pointwise baseline, attributable to its additional decoder layers and richer feature interactions rather than to context. The gains grow substantially at later positions, and for all three tasks the NE improvement beyond position 5 is roughly 7–10$\times$ larger than at the first position. This pattern is consistent with contextual conditioning supplying additional predictive signal as the slate grows, since at position s the model attends to all $s-1$ preceding items and can account for inter-item effects such as topical redundancy and complementarity that are invisible to a pointwise scorer.

5.2.2 Online Results

We evaluate the complete GCR system through an online A/B test on a large-scale short-form video platform. GCR operates as a post-LSR reranker: it receives the top-K candidates produced by LSR and constructs the final slate autoregressively, applying the in-model policy layer at every decoding step.

The production baseline follows the generator–evaluator framework of Wang et al. (2019). Its generators construct multiple candidate slates from the same top-K input by applying score perturbations and business constraints. A learned listwise evaluator then scores these slates and selects the one with the highest predicted utility. Using the same upstream candidate set ensures that the experiment measures the effect of slate construction and reranking rather than differences in retrieval. Table 2 summarizes the results.

GCR improves watch time and view count, indicating greater content consumption, while gains in reshares and follows show that the improvement extends to downstream engagement actions. Collectively, these results demonstrate that contextual slate construction improves multiple dimensions of online engagement rather than only the primary optimization metric.

Table 1: NE gain (%) of GCR over pointwise LSR by slate position. Lower is better.

Task	**Pos 1**	**Pos 2–5**	**Pos >5**	**Overall**
Task A	-0.13	-0.47	-0.92	-0.58
Task B	-0.09	-0.41	-0.73	-0.48
Task C	-0.19	-1.08	-1.97	-1.31

Table 2: Online A/B: GCR vs generator-evaluator.

Metric	**Δ (%)**
Watch Time	$+0.60$
View Count	$+0.23$
Reshare	$+2.84$
Follow	$+4.85$

5.2.3 Inference Throughput Optimization

Autoregressive slate construction introduces substantial serving overhead because a naive implementation performs a decoder evaluation at every slate position. We measure the cumulative effect of the optimizations introduced in Section 4.5 under controlled load.

Throughput is measured in completed requests per second (QPS). All configurations use the same hardware, query set, batching policy, and distribution of requested slate lengths. Table 3 reports QPS relative to the naive GCR implementation, whose throughput is normalized to 1.

The complete optimization stack improves throughput by approximately $10\times$. This gain closes the serving-efficiency gap between autoregressive GCR and the production reranker, enabling deployment within the existing GPU capacity.

Table 3: Cumulative QPS improvement over naïve GCR, normalized to 1.

Configuration	Relative QPS
GCR: naïve implementation	1
+ KV cache (Section 4.5.1)	$\sim 2\times$
+ Batched inference (Section 4.5.2)	$\sim 3.4\times$
+ Multi-forward pagination (Section 4.5.3)	$\sim 5.4\times$
+ Ahead-of-time compilation (Section 4.5.4)	$\sim 10\times$

5.3 Open-Source Dataset

We additionally evaluate GCR on KuaiRand (Gao et al., 2022), a publicly available short-video recommendation dataset collected from Kuaishou. KuaiRand provides timestamped impression logs and per-item engagement labels, allowing us to reconstruct ordered slates and compare pointwise and slate-aware prediction under a per-user chronological split. We reserve each user's final eight slates for evaluation. Full preprocessing and splitting procedures are provided in the Supplemental Materials.

To isolate the effect of contextual conditioning, we compare two models with matched architectures, parameter counts, and FLOPs. Both models use the same training examples, features, optimization settings, and random seed; they differ only in the candidate attention mask. In the pointwise HSTU baseline, each candidate attends to the user history and itself but not to other candidates. In GCR, each candidate additionally attends causally to the preceding items in the slate. This controlled comparison minimizes confounding from model capacity and attributes performance differences primarily to slate-aware conditioning.

Table 4 reports AUC for five engagement tasks on the held-out slates. The KuaiRand-1K evaluation contains 36,881 candidate impressions, whereas KuaiRand-27K contains 1,017,496. On KuaiRand-1K, GCR outperforms the pointwise baseline on four of five tasks: click, like, long-view, and profile-enter. The only exception is comment, the sparsest outcome. On KuaiRand-27K, GCR improves click, like, and long-view AUC, while results on profile-enter and comment are weaker. The consistent gains on the three common engagement tasks across both dataset scales provide additional evidence that causal slate context improves engagement prediction beyond the in-house setting.

Table 4: AUC on KuaiRand over held-out slates (higher is better; bold is best per task within each dataset).

Dataset	**Model**	**Click**	**Like**	**Long-view**	**Comment**	**Profile-enter**
KuaiRand-1K	HSTU	0.7411	0.7780	0.7451	**0.8133**	0.7027
	GCR	**0.7424**	**0.7814**	**0.7461**	0.8063	**0.7041**
KuaiRand-27K	HSTU	0.7517	0.9206	0.7603	**0.8543**	**0.8345**
	GCR	**0.7525**	**0.9213**	**0.7614**	0.8528	0.8335

6 Conclusion

We presented Generative Contextual Ranking, which replaces independent user–item scoring with autoregressive, request-level slate construction. By conditioning each selection on the user history and the items already placed, and by executing the policy layer inside the decoding loop, GCR optimizes the request as a whole rather than repairing a list after it has been ordered. Because it decodes over real candidate items rather than intermediate identifiers, GCR can be inserted at any ranking stage without a full-stack rewrite. A set of serving optimizations recovers over $10\times$ in throughput, allowing GCR to replace the production ranker on the same hardware. Experiments on a large-scale short-form video platform and on a public benchmark show improved offline prediction quality and statistically significant online gains.

AI USAGE STATEMENT

We wrote the original manuscript and used generative AI tools to revise selected passages for clarity, concision, and organization. We also used generative AI tools to assist with portions of the code implementation and to create diagrams illustrating the model and system architecture. We developed the research methodology, conducted the experiments, and determined the claims and interpretation of the results. We reviewed and tested all AI-assisted code and reviewed, revised, and verified all AI-assisted writing and diagrams. We take full responsibility for the final content of this work.

REFERENCES

Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. Learning a deep listwise context model for ranking refinement. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 135–144, 2018.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, Geeta Chauhan, Anjali Chourdia, Will Constable, Alban Desmaison, Zachary DeVito, Elber Fontenla, et al. PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. Seq2slate: Re-ranking and slate optimization with RNNs. *arXiv preprint arXiv:1810.02019*, 2019.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198, 2016.

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, 2025.

Yufei Feng, Binbin Hu, Yu Gong, Fei Sun, Qingwen Liu, and Wenwu Ou. GRN: Generative rerank network for context-wise recommendation. *arXiv preprint arXiv:2104.00860*, 2021.

Chongming Gao, Shijun Li, Yuan Zhang, Jiawei Chen, Biao Li, Wenqiang Lei, Peng Jiang, and Xiangnan He. Kuairand: An unbiased sequential recommendation dataset with randomly exposed videos. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management*, CIKM '22, pp. 3953–3957, 2022. doi: 10.1145/3511808.3557624. URL `https://doi.org/10.1145/3511808.3557624`.

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quinonero Candela. Practical lessons from predicting clicks on ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pp. 1–9, 2014.

Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. SLATEQ: A tractable decomposition for reinforcement learning with recommendation sets. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pp. 2592–2598, 2019.

Ray Jiang, Sven Gowal, Yuqiu Qian, Timothy Mann, and Danilo J. Rezende. Beyond greedy ranking: Slate optimization via List-CVAE. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *Proceedings of the IEEE International Conference on Data Mining*, pp. 197–206, 2018.

Yi Li, Jieming Zhu, Weiwen Liu, Liangcai Su, Guohao Cai, Qi Zhang, Ruiming Tang, Xi Xiao, and Xiuqiang He. PEAR: Personalized re-ranking with contextualized transformer for recommendation. In *Companion Proceedings of the ACM Web Conference 2022*, pp. 62–66, 2022.

Zhijie Lin, Zhuofeng Li, Chenglei Dai, Wentian Bao, Shuai Lin, Enyun Yu, Haoxiang Zhang, and Liang Zhao. GReF: A unified generative framework for efficient reranking via ordered multi-token prediction. *arXiv preprint arXiv:2510.25220*, 2025.

Shuchang Liu, Qingpeng Cai, Zhankui He, Bowen Sun, Julian McAuley, Dong Zheng, Peng Jiang, and Kun Gai. Generative flow network for listwise recommendation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1524–1534, 2023.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1930–1939, 2018.

Liang Pang, Jun Xu, Qingyao Ai, Yanyan Lan, Xueqi Cheng, and Jirong Wen. SetRank: Learning a permutation-invariant ranking model for information retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 499–508, 2020.

Yunsheng Pang, Zijian Liu, Yudong Li, Shaojie Zhu, Zijian Luo, Chenyun Yu, Sikai Wu, Shichen Shen, Congying Xia, Yanchi Liu, Haifeng Chen, and Liang Wang. HiGR: Efficient generative slate recommendation via hierarchical planning and multi-objective preference alignment. *arXiv preprint arXiv:2512.24787*, 2025.

Changhua Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, Junfeng Ge, Wenwu Ou, and Dan Pei. Personalized re-ranking for recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 3–11, 2019.

Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. Practice on long sequential user behavior modeling for click-through rate prediction. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2671–2679, 2019.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan H. Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems 36*, 2023.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *Proceedings of the 4th International Conference on Learning Representations*, 2016.

Yuxin Ren, Qiya Yang, Yichun Wu, Wei Xu, Yalong Wang, and Zhiqiang Zhang. Non-autoregressive generative models for reranking recommendation. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, Barcelona, Spain, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

Xiaowen Shi, Fan Yang, Ze Wang, Xiaoxu Wu, Muzhi Guan, Guogang Liao, Yongkang Wang, Xingxing Wang, and Dong Wang. PIER: Permutation-level interest-based end-to-end re-ranking framework in e-commerce. *arXiv preprint arXiv:2302.03487*, 2023.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformers. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1441–1450, 2019.

Federico Tomasi, Francesco Fabbri, Justin Carter, Elias Kalomiris, Mounia Lalmas, and Zhenwen Dai. Prompt-to-slate: Diffusion models for prompt-conditioned slate generation. In *Proceedings of the 19th ACM Conference on Recommender Systems*, 2025. doi: 10.1145/3705328.3748072.

Fan Wang, Xiaomin Fang, Lihang Liu, Yaxue Chen, Jiucheng Tao, Zhiming Peng, Cihang Jin, and Hao Tian. Sequential evaluation and generation framework for combinatorial recommender system. *arXiv preprint arXiv:1902.00245*, 2019.

Yunjia Xi, Weiwen Liu, Jieming Zhu, Xilong Zhao, Xinyi Dai, Ruiming Tang, Weinan Zhang, Rui Zhang, and Yong Yu. Multi-level interaction reranking with user behavior history. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1336–1346, 2022.

Xue Xia, Pong Eksombatchai, Nikil Pancha, Dhruvil Deven Badani, Po-Wei Wang, Neng Gu, Saurabh Vishwas Joshi, Nazanin Farahpour, Zhiyuan Zhang, and Andrew Zhai. TransAct: Transformer-based realtime user action model for recommendation at Pinterest. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5249–5259, 2023.

Bi Xue, Hong Wu, Lei Chen, Chao Yang, Yiming Ma, Fei Ding, Zhen Wang, Liang Wang, Xiaoheng Mao, Ke Huang, Xialu Li, Peng Xia, Rui Jian, Yanli Zhao, Yanzun Huang, Yijie Deng, Harry Tran, Ryan Chang, Min Yu, Eric Dong, Jiazhou Wang, Qianqian Zhang, Keke Zhai, Hongzhang Yin, Pawel Garbacki, Zheng Fang, Yiyi Pan, Min Ni, Kevin Greer, Rui Zhang, and Yang Liu. Silvertorch: A unified model-based system to democratize large-scale recommendation on gpus. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 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 International Conference on Machine Learning*, volume 235 of *PMLR*, pp. 58484–58509, 2024.

Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1059–1068, 2018.

A Supplemental Materials: KuaiRand Reproducibility Details

A.1 Dataset and slate reconstruction

KuaiRand-1K contains 1,000 users, approximately 4.37 million videos, and 11.7 million standard-policy interactions across the two periods in Table 5; KuaiRand-27K scales this to 27,285 users and 215.9 million interactions over the same two periods. KuaiRand exposes no explicit request ID or displayed position, but each logged event carries a `tab` feature indicating the serving surface. Slates are reconstructed as maximal same-timestamp runs in which every candidate has tab 1 (the single-column short-form video interface), with raw row order as the position proxy. Only tab 1 events are used, for both candidate slates and UIH timelines. Same-timestamp tab-1 runs are chained into non-overlapping slate examples per user in raw chronological order, with no truncation and no shuffling. Each example contains a user token, the adjacent preceding tab-1 UIH capped at 200 events, one ordered candidate slate, and candidate-level engagement outcomes. Each user's last 8 slate chains are held out for evaluation (∼80/20 split), with earlier chains supplying training examples whose history-plus-target ranges do not overlap. Random-exposure logs are not used.

Table 5: KuaiRand raw periods and processed slate counts. Periods are joined per user into one chronological event store; evaluation holds out each user's last 8 slates. Log names prepend `log_standard_` to the period and append `_1k.csv` (one file) or `_27k_part1/2.csv` (two parts).

Artifact	**1K**	**27K**
Events `4_08_to_4_21` (earlier)	5,055,984	90,261,063
Events `4_22_to_5_08` (later)	6,657,061	125,663,960
Training slates	30,354	851,219
Evaluation slates	7,230 (19.2%)	199,655 (19.0%)
Evaluation candidate labels	36,881	1,017,496

The resulting slate counts are summarized in Table 5. HSTU and GCR consume identical processed examples; the treatment changes only the attention behavior of the final four layers.

A.2 Features and objective

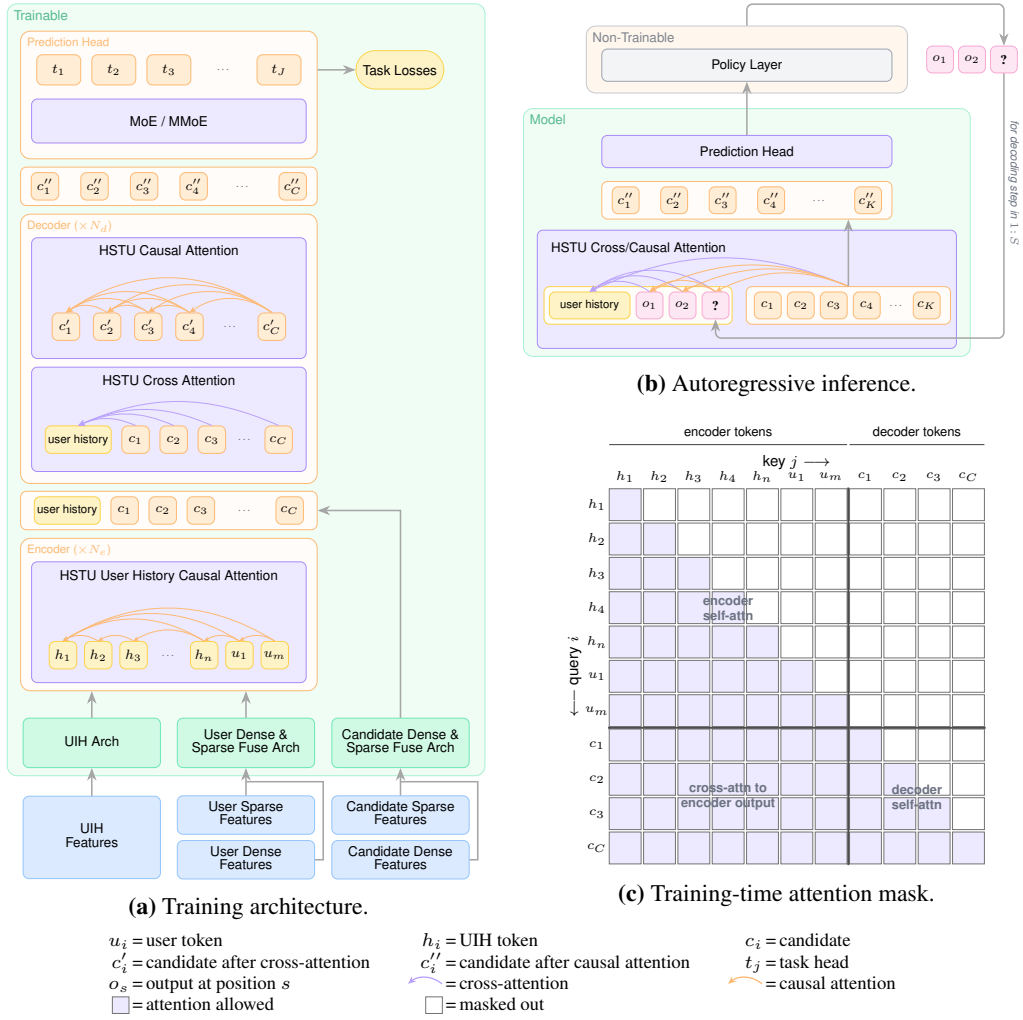
Each request starts with a learned user token combining user ID, user type, log1p-transformed numeric profile features, and embeddings for 26 categorical profile fields. UIH and candidate videos receive video-ID embeddings plus content metadata (hashed author and music IDs, video and upload type, visibility, music type, up to five mean-pooled tags, duration, dimensions, aspect ratio, presence indicators) and event context (relative time, minute of day, day of month, random-exposure indicator, tab); monthly aggregate video statistics are not used. UIH events include observed historical engagement; candidate outcomes (click, like, follow, long-view, comment, forward, hate, profile-enter, play-time, stay-time) are never model inputs. Both models produce eight logits per candidate through the same head (LayerNorm, 64-to-256 linear, SiLU, dropout 0.1, 256-to-8 output) and train with masked binary cross entropy; we report the five tasks with enough positives for comparison (follow, forward, and hate are omitted as too sparse on 1K). Positive rates on the evaluation candidates are 51.36%, 2.48%, 37.80%, 0.40%, 2.64% for click, like, long-view, comment, profile-enter on 1K (36,881 candidates) and 50.77%, 2.58%, 37.18%, 0.35%, 2.60% on 27K (1,017,496 candidates).

A.3 Depth-matched models

Both models share the same depth configuration, denoted (4+4): four shared encoder blocks plus four upper blocks. UIH states update through all eight blocks in both. In HSTU (4+4), each candidate attends to the full valid UIH and itself, never another candidate. In GCR (4+4), the first four blocks are the same candidate-independent encoder; the final four are full-joint decoder layers in which candidate j additionally attends to earlier candidate representations, never future ones. No prior candidate outcome is exposed.

A.4 Training

One epoch, batch size 32, AdamW (lr 0.001, weight decay 0.0001), embedding width 64 with one attention head, attention dropout 0, linear and head dropout 0.1, seed 42 with deterministic algorithms and TF32 disabled.



(a) Training architecture.

(b) Autoregressive inference.

(c) Training-time attention mask.

Figure 2: GCR training and inference. **(a)** Teacher-forced training computes all impressed-item predictions in one parallel forward pass. **(b)** Inference-time decoding loop (simplified). The encoder and the two HSTU attention blocks of Figure 2(a) are abstracted into a single HSTU Cross/Causal Attention layer. Candidates cross-attend to user history (purple) and causally attend to the prior-output prefix $\mathcal{O}_{<s}$ (orange); the policy layer selects the next o_s and feeds it back into the query slot. **(c)** The training-time attention mask separates encoder and decoder tokens and enforces causal attention among impressed candidates.