YOUR BEST RETRIEVER MAY NOT BE BEST FOR YOUR RERANKER: ADAPTIVE COORDINATION FOR LLM RECOMMENDATION CASCADES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale recommender systems typically adopt a two-stage retrieve-then-rerank cascade, in which a retriever narrows a massive item pool to a small candidate set, and a reranker produces the final ranking. This creates an objective misalignment between recall-oriented retrieval and precision-oriented reranking: the retriever aims to ensure that relevant items are included, whereas the reranker must accurately distinguish them from competing candidates and place them at the top of the final ranking. Retrieval should therefore provide not only relevant items, but candidate sets that the downstream reranker can reliably resolve into high-quality final rankings. Prior work often addresses this misalignment through joint optimization, but discrete candidate selection limits end-to-end gradient flow, while the retrieval and reranking objectives may induce competing optimization directions. We propose R4Rec (Retrieve–Rerank–Reallocate–Refine), a cascaded framework that uses two trainable feedback mechanisms to coordinate pre-trained **retriever**s and an LLM-based **reranker**, aligning candidate generation with final reranking without updating the underlying retriever or reranker components. Across-Sample **Reallocation** learns context-dependent retriever quotas, assigning larger budgets to retrievers whose candidates better support the downstream reranker. It derives each retriever's contribution from closed-form, reranker-aligned Shapley values and uses this feedback to train a contextual bandit. Within-Sample **Refinement** instead operates at the request level: a separate contextual bandit decides whether to stop or refine the current candidate set and, when refinement is needed, generates a diagnostic query to retrieve additional candidates before reranking. Experiments on Amazon and MovieLens show that R4Rec improves accuracy at practical inference cost, with a statistically significant 6.43% relative CTR lift over the production baseline in an online A/B test[1].

1 INTRODUCTION

Modern recommender systems are commonly organized as two-stage cascades (Xu et al., 2023; Yuan et al., 2022). A lightweight retriever first narrows a large item pool, typically ranging from tens of thousands to millions of items, into a small candidate set, and a more precise reranker then selects the final items to expose to the user (Covington et al., 2016).

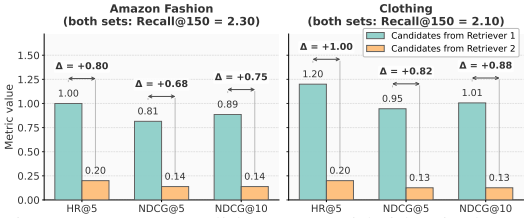

Figure 1: Two candidate sets with identical Recall@150 yield markedly different performance.

A central challenge in two-stage cascade recommendation is that retrieval and reranking optimize related but misaligned objectives. The retriever aims to maximize recall over a massive item pool, whereas the reranker must maximize precision within the candidate set returned by the retriever. Consequently, high recall does not necessarily imply high final top-K quality. As Fig. 1 illustrates, candidate sets with identical recall can lead to very different reranking outcomes. The candidate set from Retriever 1 may

[1]We released our code to support reproducibility (anonymous.4open.science/r/R4Rec-2EC5).

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.

contain the relevant item among alternatives that the reranker can clearly distinguish using the user's interaction history. In contrast, Retriever 2 can return off-intent or near-duplicate candidates that make the relevant item harder to identify, resulting in lower top-K quality. In other words, beyond achieving high recall, the retriever should construct candidate sets that the downstream reranker can reliably resolve into high-quality final rankings. Prior cascade studies connect the two stages through joint training (Wang et al., 2025), score fusion (Huang et al., 2025) or learning-to-rank objectives (Pei et al., 2019; Zheng et al., 2024). Yet direct alignment remains difficult: the top-K selection linking retrieval and reranking is discrete and non-differentiable, blocking end-to-end gradient flow. This motivates adaptive cascade coordination, which steers retrieval with downstream reranker feedback rather than coupling the two stages into one end-to-end trainable objective.

We therefore propose R4Rec (Retrieve, Rerank, Reallocate, Refine), a cascaded framework that coordinates multiple lightweight retrievers with a high-capacity LLM-based reranker without retraining either component end-to-end. **The goal is not merely to maximize candidate recall, but to construct candidate sets that the downstream reranker can effectively resolve.** R4Rec assumes a set of well-trained retrievers with distinct strengths across user contexts and treats them, together with the LLM-based reranker, as complementary frozen tools. It then uses two complementary feedback mechanisms to adapt retrieval based on downstream reranking utility. (1) Across-Sample Reallocation operates over a long horizon, casting retriever allocation as a cooperative game, in which each retriever receives downstream-aware credit based on its marginal contribution to the LLM reranker's final top-K success. By accumulating these credits across requests, R4Rec learns which retrievers are more reliable under similar user contexts and reallocates larger quotas to those that consistently produce reranker-resolvable candidate sets. (2) In contrast, Within-Sample Refinement acts on the current request after the initial retrieval and reranking steps. It uses a learned bandit gate whose arms correspond to stopping and local refinement, and updates the gate based on whether refinement improves the target item's rank in the reranked list. When refinement is predicted to be beneficial, R4Rec invokes a refiner to retrieve additional candidates and revise only the uncertain portion of the candidate set before reranking. By coupling reallocation across requests with refinement within each request, R4Rec extends the static retrieve-then-rerank pipeline into an adaptive retrieve–rerank–reallocate–refine cascade, guiding retrievers to produce candidate sets that the downstream reranker can reliably resolve and thereby improving final ranking quality.

Comprehensive experiments across five Amazon domains and MovieLens show that R4Rec improves candidate recall while constructing candidate sets that are more amenable to accurate LLM reranking, resulting in higher final recommendation performance than Collaborative Filtering (CF)-based and static cascade baselines. **Our contributions are threefold**:

- We propose R4Rec, which aligns retrievers with an LLM reranker through two coordination mechanisms; it consistently outperforms CF- and static-cascade baselines (§4.2) at practical inference cost (§4.5), with gains that stem primarily from more reranker-resolvable candidate sets and translate into a 6.43% relative CTR lift over LLM-Rec in the online A/B test. (§4.6).
- Across-Sample Reallocation casts retriever allocation as a cooperative game, learning context-dependent quotas from reranker-aligned credit to favor reranker-resolvable candidate sets (§4.3).
- Within-Sample Refinement learns a bandit gate that revises a candidate set only when doing so is predicted to improve the ranking, recovering initial-pass failures at marginal inference cost (§4.4).

2 Related Work and Motivation

Two-Stage Recommendation Cascade. Recommender systems typically use a two-stage cascade: retrieval selects a small candidate set from a large item pool, and reranking refines the final order based on user preferences (Huang et al., 2025; Pei et al., 2019). Let $\mathcal{I}$ be an item pool with $|\mathcal{I}| = N$, where each item $i \in \mathcal{I}$ is described by textual metadata, including attributes such as title, description, category, and brand. For a given user, we denote the interaction history by $x = (i_1, \ldots, i_{|x|})$. A two-stage recommendation cascade first applies a retriever f to reduce the large item pool $\mathcal{I}$ to a candidate list of size K_D: $D = f(x, \mathcal{I}) = (d_1, \ldots, d_{K_D})$, where each $d_j \in \mathcal{I}$ and $K_D \ll N$. The reranker g then takes the user history and the candidate item representations as input and produces an ordered ranking $R = g\big(x, D\big)$. The system finally exposes the top-K prefix $R_{1:K}$ to the user ($K \leq K_D$), with K typically around 10–30 and K_D around 100–200 (Wang et al., 2025).

The Retriever–Reranker Objective Gap. In this work, we assume that the retriever and reranker have been trained to serve their respective objectives. Let $y \in \mathcal{I}$ denote the **gold item**, i.e., the held-out next item for a request x. The retriever is usually evaluated by whether candidate D contains the

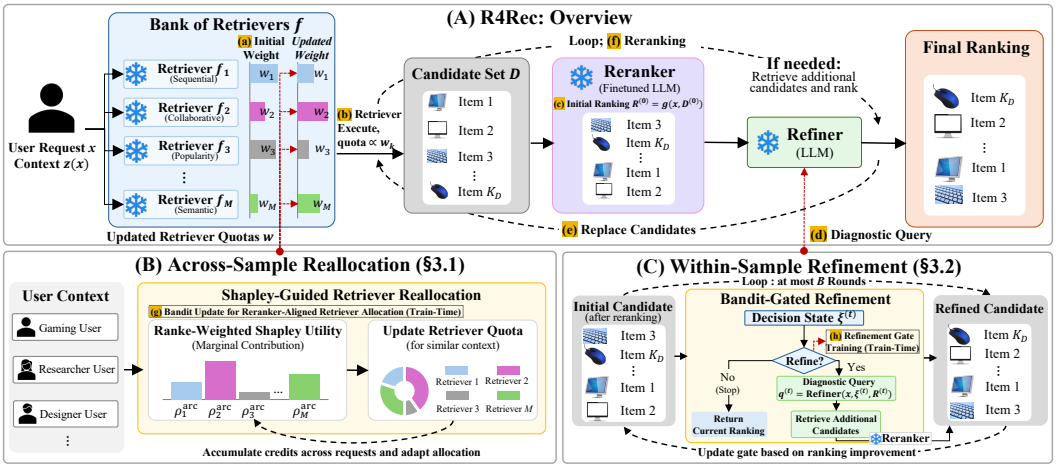

Figure 2: Overview of R4Rec.

gold item, e.g., $\text{Recall@}K_D(D) = \mathbf{1}\{y \in D\}$. However, the final recommendation quality depends on the ranked output of the reranker, $R = g(x, D)$. The final output is evaluated by ranking metrics such as $\text{HR@}K(R) = \mathbf{1}\{y \in R_{1:K}\}$. Since the reranker can only rank items that appear in D, the final hit probability decomposes as $\Pr(y \in R_{1:K}) = \Pr(y \in D) \cdot \Pr(y \in R_{1:K} \mid y \in D)$. The first factor $\Pr(y \in D)$ corresponds to the retriever's objective, i.e., retrieval recall: the gold item must first be included in the retrieved candidate list. The second factor $\Pr(y \in R_{1:K} \mid y \in D)$ corresponds to reranking success, the reranker's objective: even once the gold item is retrieved, the reranker must still rank it within the top-K. This decomposition shows that high retrieval recall is necessary but not sufficient for high final recommendation quality. A retriever optimized only for $\text{Recall@}K_D$ treats all candidate lists containing y as equivalent, despite substantial differences in their reranking difficulty. We call this mismatch between retrieval quality and final ranking quality the **retriever–reranker objective gap**.

Why This Gap Calls for Adaptive Cascade Coordination. Closing this objective gap through retriever–reranker joint training would require the retriever f to optimize not only target-item recall but also the construction of candidate sets that the reranker can reliably resolve. This is difficult for two reasons. First, discrete operations such as top-K_D retrieval and candidate construction break direct gradient flow across the retrieval–reranking boundary. This limitation is particularly pronounced for LLM rerankers, which consume candidates as textual prompts and generate rankings autoregressively. Although differentiable selection operators can approximate this boundary (Wang et al., 2025), they may introduce train–test mismatch and conflicting retrieval and reranking objectives. Second, the quality of a candidate set can be fully assessed only after reranking. Candidate sets with identical target-item recall may produce substantially different final rankings (Fig. 1), meaning that retrieval quality depends on how well the downstream reranker can resolve the retrieved candidates. A static retrieval policy therefore cannot always anticipate whether or how a candidate set should be revised. We thus formulate retriever–reranker alignment as an adaptive cascade coordination problem over fixed retrievers and reranker. See Appendix C for extended related work.

3 R4Rec: Retrieve, Rerank, Reallocate, Refine

As shown in Fig. 2, R4Rec is a cascaded framework in which multiple pre-trained retrievers first **Retrieve** candidates from a large item pool, and a reranker then **Reranks** them to produce the final recommendation. A single controller adaptively coordinates this cascade through two bandit-based feedback mechanisms: Across-Sample **Reallocation** shifts candidate quotas toward retrievers that better support downstream reranking across requests, while Within-Sample **Refinement** selectively revises uncertain candidate sets within the current request before reranking them. **The contextual-bandit formulation is motivated by continual deployment**: after serving a coordination decision, downstream item-level feedback such as clicks or purchases can be used to incrementally update the controller statistics, allowing the coordination policy to adapt as user and item distributions evolve.

3.1 Across-Sample Reallocation: Shapley-Guided Retriever Reallocation

Retriever selection is inherently request-dependent: different user histories, contexts, and intents may require different retrievers (Huang et al., 2025). For example, sequence-based retrievers may

be preferable for users with long interaction histories, whereas semantic retrievers may better capture sparse or context-dependent intents (§4.3). To make retrieval allocation context-aware, R4Rec represents each request x—the user's interaction history—by an d-dimensional context vector $z(x) \in \mathbb{R}^d$, summarizing retriever-relevant properties such as history length, rating statistics, category diversity, and textual features of recent interactions. We provide the full specification in Appendix D.

3.1.1 TRAINING-TIME RETRIEVER REALLOCATION

Initial Weight Allocation. Given a bank of retrievers $\mathcal{F} = \{f_1, \ldots, f_M\}$, we cast retriever selection as a contextual bandit, treating each retriever as an arm whose utility depends on the current user context. The bank comprises heterogeneous retrievers, which are detailed in Appendix E. For each request, the controller estimates a context-dependent utility for each retriever and converts these utilities into mixing weights $w = (w_1, \ldots, w_M)$ (Fig. 2a). The weight w_k sets the quota for retriever f_k: retrievers with larger weights contribute more candidates before deduplication and truncation to the candidate size K_D. Each retriever then runs in parallel and contributes candidates in proportion to its quota w_k (Fig. 2b). The resulting initial candidate set $D^{(0)}$ is passed to the reranker g, which produces an initial ranking $R^{(0)} = g\big(x, D^{(0)}\big)$ (Fig. 2c).

Retriever-Stage Cooperative Game. In the controller-training stage, we use training examples with held-out gold items to compute retriever-level utility via a cooperative game over the retriever bank. These utilities supervise the retriever controller but do not update the parameters of either the retrievers or the reranker. To quantify how much each retriever contributes to recovering the gold item, we compute its Shapley value (Lundberg & Lee, 2017; Shapley et al., 1953) in this game. Here, retrievers are treated as players, and a coalition $\mathcal{S}$ corresponds to a subset of retrievers whose retrieved items are pooled together. The characteristic function v maps each coalition to a binary value indicating whether the retrievers in that coalition recover the gold item in their pooled candidate list. Let $\tilde{D}_k^{(0)}$ denote the quota-truncated ranked list returned by retriever f_k after within-retriever deduplication, and let $D_k^{(0)} = \mathrm{set}(\tilde{D}_k^{(0)}) \subseteq D^{(0)}$ be the corresponding unordered item set contributed to the initial candidate after quota allocation and truncation. Let $\mathcal{T}(x, y) = \{k : y \in D_k^{(0)}\}$ be the set of retrievers that surface the gold item. A natural choice of characteristic function v is therefore the indicator that some retriever in the coalition surfaces y: $v_{\mathrm{recall}}(\mathcal{S} \mid x, y) = \mathbf{1}\left[y \in \bigcup_{k \in \mathcal{S}} D_k^{(0)}\right]$. This recall game admits a simple closed-form Shapley value: retrievers in $\mathcal{T}(x, y)$ share the utility uniformly as $1/|\mathcal{T}(x, y)|$, while all others receive zero utility; see Appendix H.2 for the proof. This avoids subset enumeration, but yields coarse attribution because a retriever that ranks the gold item first receives the same utility as one that ranks it near the tail (§4.3). We therefore introduce a **rank-weighted ordinal game** that incorporates each retriever's rank of the gold item.

Rank-Weighted Shapley Utility. Let $r_{k,y} \geq 1$ denote the rank at which retriever f_k places the gold item y in its own ranked list, with $r_{k,y} = \infty$ when $k \notin \mathcal{T}(x, y)$. We first assign a logarithmically discounted rank payoff, $J(r) = \frac{1}{\log_2(r+1)}$, which mirrors the logarithmic rank discount of normalized discounted cumulative gain (NDCG), giving full utility when the gold item is ranked first and gradually smaller utility as it appears lower in the ranked list. To preserve a unit amount of retriever-stage utility per labeled request, we normalize this payoff by the best gold-surfacing retriever in the current user context: $\bar{J}_{k,y} = \frac{J(r_{k,y})}{\max_{\ell \in \mathcal{T}(x,y)} J(r_{\ell,y})},\ k \in \mathcal{T}(x, y)$. Thus, the best-ranking retriever among those that surface y receives normalized payoff 1, while lower-ranking retrievers receive smaller positive payoffs. Define the rank-weighted recall game: $v_{\mathrm{rank}}(\mathcal{S} \mid x, y) = \max_{k \in \mathcal{S} \cap \mathcal{T}(x,y)} \bar{J}_{k,y}$, with $v_{\mathrm{rank}}(\mathcal{S}) = 0$ when $\mathcal{S} \cap \mathcal{T}(x, y) = \emptyset$. A coalition's value is the normalized rank-discounted payoff of its best-ranking gold-surfacing retriever. The Shapley value of v_{rank} admits a closed form; see Proposition 2 in Appendix H for the proof. This avoids the exponential cost of the direct Shapley value definition, which would require marginal-contribution evaluation over all retriever coalitions and cost $O(M2^M)$ for M retrievers.

Specifically, sort the elements of $\mathcal{T}(x, y)$ by ascending rank as $\tau_1, \tau_2, \ldots, \tau_{|\mathcal{T}(x,y)|}$ (best rank first) and adopt the convention $\bar{J}_{\tau_{|\mathcal{T}(x,y)|+1},y} = 0$. The closed-form rank-weighted Shapley value assigned to retriever τ_j is:

$$\phi_{\tau_j}(x, y) \;=\; \sum_{i=j}^{|\mathcal{T}(x,y)|} \frac{\bar{J}_{\tau_i,y} - \bar{J}_{\tau_{i+1},y}}{i}, \tag{1}$$

while retrievers outside $\mathcal{T}(x, y)$ receive zero utility. Unlike generic Shapley computation, which requires enumerating all retriever coalitions, this closed form only depends on the rank-ordered

retrievers in $\mathcal{T}(x, y)$. Thus, R4Rec obtains principled game-theoretic utility assignment with negligible overhead, keeping across-sample reallocation efficient even with many retrievers (§4.5).

Cascade-Gated Reward. Since a retriever's contribution is realized only if its candidates also survive the reranker, the controller's Shapley value combines this utility with the cascade-success indicator. We define the reranker-aligned Shapley value as

$$\rho_k^{\mathrm{acr}}(x, y, R^{(0)}) \;=\; \underbrace{\mathbf{1}[\, y \in R^{(0)}_{1:K}\,]}_{\text{cascade-gate}} \cdot \quad \underbrace{\phi_k(x, y)}_{\text{Shapley value (Eq. 1)}} \quad . \tag{2}$$

The first factor is the cascade gate, which enforces cascade success: the gold item must be both retrieved by the retriever bank and retained in the reranker's top-K output. The second factor is the rank-weighted Shapley value share of retriever k (Eq. 1). This reward is a cascade-gated retriever-stage attribution, rather than a counterfactual measure of each retriever's contribution to reranking.

Bandit Update for Reranker-Aligned Retriever Reallocation (Fig. 2g). The retriever mixing weights w, which determine per-retriever retrieval quotas, are learned through a LinUCB-style update (Li et al., 2010) driven by the reranker-aligned Shapley value in Eq. 2. LinUCB models each retriever's expected utility as a linear function of the request context and augments this estimate with an uncertainty bonus, so that allocation balances exploiting retrievers with high estimated utility against exploring those whose utility is still uncertain. Concretely, for each retriever k the controller maintains statistics $A_k \in \mathbb{R}^{d\times d}$ and $b_k \in \mathbb{R}^d$, initialized as $A_k = \lambda I$ and $b_k = 0$, which define the ridge-regression estimate $\hat{\theta}_k = A_k^{-1} b_k$. Given a request x, the controller computes $u_k(x) = z(x)^\top \hat{\theta}_k + \alpha_{\mathrm{acr}} \sqrt{z(x)^\top A_k^{-1} z(x)}$, where the first term estimates the expected rank-weighted Shapley value, and the second is the uncertainty bonus that encourages exploration, with α_{acr} controlling its strength. The mixing weights are then obtained by a temperature-T softmax over these scores, $w_k = \frac{\exp(u_k(x)/T)}{\sum_{j=1}^{M} \exp(u_j(x)/T)}$. After reranking and Shapley utility assignment, let $\mathcal{P}(x) = \{k : \mathrm{set}(\widetilde{D}_k^{(0)}) \cap D^{(0)} \neq \varnothing\}$ denote the retrievers with at least one returned item retained in the merged and truncated candidate set $D^{(0)}$. We update $A_k \leftarrow A_k + z(x)z(x)^\top$ and $b_k \leftarrow b_k + \rho_k^{\mathrm{acr}} z(x)$ only for $k \in \mathcal{P}(x)$, including zero-reward observations; all other retrievers retain their current statistics. The statistics thus accumulate over retriever-specific participation histories during training.

3.1.2 Test-Time Retriever Reallocation

For controlled offline evaluation, the learned allocation bandit is frozen for stable evaluation, whereas in deployment it is continuously updated from incoming feedback. For each request, the controller uses the learned bandit statistics to compute context-dependent retriever utilities and mixing weights, which determine the retrieval quotas. The resulting candidate is then passed to the fixed LLM reranker for final top-K prediction.

3.2 Within-Sample Refinement: A Bandit-Gated Revision Operator

Across-Sample Reallocation adapts the retriever composition for each user context, while Within-Sample Refinement uses the reranking outcome of each request to decide whether to STOP or further REFINE the candidate set through additional retrieval and reranking.

3.2.1 Training-Time Refinement Gate

Decision State. Suppose the refinement process is repeated over B rounds indexed by t, as shown in Fig. 2. At refinement round t, the reranker produces $R^{(t)} = g\big(x, D^{(t)}\big)$. The controller then summarizes the current reranking state into a decision state feature $\xi^{(t)} = [\, h^{(t)}_{\text{state}};\, z(x)\,]$, where the deterministic map Φ produces $h^{(t)}_{\text{state}} = \Phi\big(R^{(t)}, D^{(t)}, \eta^{(t)}\big)$ from the current ranking $R^{(t)}$, candidate pool $D^{(t)}$, and retriever diagnostics $\eta^{(t)}$, and $z(x)$ is the cached request context of Appendix D. $\eta^{(t)}$ is a quantitative diagnostic report of how the retrievers behaved—which retrievers proposed each candidate, the scores they assigned, and how strongly they agree with one another (see Appendix F).

REFINEMENT Operator. A refinement rewrites part of the candidate set and reranks it; the same operator is used at training and at inference, and only the decision to invoke it differs. Given the decision state $\xi^{(t)}$ and the user history x, a frozen general-purpose LLM—distinct from the reranker g and acting only as a query writer—generates

a short refinement query $q^{(t)} = \mathrm{Refiner}_{\mathrm{LLM}}\big(x, \xi^{(t)}, R^{(t)}_{1:K}\big)$ (Fig. 2d). This query names the missing or underrepresented aspect of the candidate, such as a category or intent (e.g., `Retrieve items that complement casual cotton-and-denim outfits rather than more of the same tops`). Let $\mathcal{Q} \subseteq [M]$ denote the retrievers capable of natural-language querying, such as Sentence-Transformer-based semantic retrievers (Reimers & Gurevych, 2019). Only these retrievers are re-invoked with $q^{(t)}$, while the remaining ones keep their original outputs: $\widetilde{D}^{(t+1)} = \big(\bigcup_{k\in\mathcal{Q}} f_k(x, q^{(t)})\big) \cup \big(\bigcup_{k\notin\mathcal{Q}} f_k(x)\big)$. The controller then refreshes only part of the candidate, $D^{(t+1)} = \mathrm{PartialReplace}_{K_D}\big(D^{(t)}, \widetilde{D}^{(t+1)}, \gamma\big)$, replacing at most a γ fraction of the current candidates, and passes the result back to the reranker (Fig. 2f).

Refinement Gate Training (Fig. 2h). As in across-sample reallocation, the refinement gate is trained on gold-labeled requests. At each refinement round, the gate treats STOP as a fixed zero-value action and estimates the value of REFINE using an optimistic LinUCB score. Specifically, for the state feature $\xi^{(t)}$, the gate computes $u^{(t)}_{\mathrm{wtn}} = \hat{\theta}^{\top}_{\mathrm{wtn}}\xi^{(t)} + \alpha_{\mathrm{wtn}}\sqrt{\xi^{(t)\top}A^{-1}_{\mathrm{wtn}}\xi^{(t)}}$. During training, the controller executes REFINE if $u^{(t)}_{\mathrm{wtn}} > 0$ and otherwise selects STOP. When REFINE is executed, the resulting ranking change is observed as $\rho^{(t)}_{\mathrm{wtn}} = G_y\big(R^{(t+1)}\big) - G_y\big(R^{(t)}\big)$, where $G_y(R) = \frac{1}{\log_2(\mathrm{rank}_R(y)+1)}$ if $y \in R$ and 0 otherwise, and $\mathrm{rank}_R(y)$ is the 1-indexed position of the gold item y. The gate then updates its LinUCB statistics as $A_{\mathrm{wtn}} \leftarrow A_{\mathrm{wtn}} + \xi^{(t)}\xi^{(t)\top}$, $b_{\mathrm{wtn}} \leftarrow b_{\mathrm{wtn}} + \rho^{(t)}_{\mathrm{wtn}}\xi^{(t)}$, $\hat{\theta}_{\mathrm{wtn}} = A^{-1}_{\mathrm{wtn}}b_{\mathrm{wtn}}$. If STOP is selected, the counterfactual refinement outcome remains unobserved, no bandit update is performed, and the refinement process terminates. The LinUCB bonus therefore drives exploration of uncertain refinement actions, while the gold item y is used only to construct rewards for executed refinements and is never available at inference.

3.2.2 TEST-TIME REFINEMENT DECISION

During controlled offline evaluation, the gate is frozen and no gold label is available, so exploration is disabled and the posterior-mean score $u^{(t)}_{\mathrm{wtn}} = \hat{\theta}^{\top}_{\mathrm{wtn}}\xi^{(t)}$ decides whether to act. The controller applies Refine if $u^{(t)}_{\mathrm{wtn}} > 0$, and otherwise stops and returns the top-K ranking $R^{(t)}1$. The loop runs for at most B rounds and terminates as soon as the gate declines to refine. R4Rec thus spends extra inference only where refinement is predicted to help (§4.4–§4.5).

4 EXPERIMENTS

We address the following research questions: **RQ1**. Does R4Rec outperform CF-based and static-alignment cascades across diverse domains, and do these gains stem from higher candidate recall, improved reranker resolvability, or both? **RQ2**. Does across-sample reallocation outperform single-retriever, uniform-allocation, and reward-ablated alternatives while adapting to user context? **RQ3**. Does within-sample refinement recover single-pass reranking failures while achieving a better accuracy–cost frontier than no-refinement and always-refinement baselines? **RQ4**. Does R4Rec deliver performance gains while maintaining practical latency overhead? **RQ5**. Do R4Rec's offline gains translate into significant CTR improvements in a real-world online recommendation service?

4.1 EXPERIMENTAL SETTINGS

We compare R4Rec against two families of baselines: (i) CF-based recommenders (CF-Rec), (ii) static-alignment cascades that align or fuse the two stages a priori. For fairness, all methods are evaluated under the same item pool, train/test split, metrics, and candidate size $K_D = 150$. For alignment-cascade baselines, we use the same retriever bank and LLM-based reranker where applicable to ensure a fair comparison. In the sequential recommendation setting, we adopt a leave-one-out evaluation protocol: the last interacted item of each user is held out as the ground-truth target, and the model receives the preceding interaction history as input (See Appendix A.2). We evaluate performance using HR@K and NDCG@K; implementation details are provided in Appendix B.

4.2 OVERALL PERFORMANCE AND THE SOURCE OF THE GAP (RQ1)

(1) R4Rec outperforms most baselines. Compared with CF-Rec, R4Rec's advantage indicates that lightweight CF-Rec have limited capacity for precise final ranking over a large item pool (Table 1). Although CF-Rec can efficiently score the entire item catalog, they often struggle to distinguish subtle preference differences within such a large item space. This limitation is most pronounced in large-catalog domains: on Clothing domain, whose item pool is the largest (i.e., 10^5), R4Rec improves HR@10 by roughly $6\times$ over the best CF-Rec baseline (i.e., SASRec), compared with

Table 1: Accuracy on **Amazon** and **MovieLens**. Best and second-best are **bold** and underlined. †: significant gain over the best baseline (paired t-test, $p < 0.01$).

Domain		Fashion			Grocery			Clothing			Health			CDs			MovieLens		
Types	**Models**	N@5	N@10	H@10	N@5	N@10	H@10	N@5	N@10	H@10	N@5	N@10	H@10	N@5	N@10	H@10	N@5	N@10	H@10
CF Rec	SASRec	0.22	0.22	0.32	0.95	1.32	2.59	0.10	0.10	0.16	0.15	0.15	0.24	1.08	1.23	1.94	6.03	7.93	15.40
	BERT4Rec	0.13	0.21	0.41	1.10	1.35	2.67	0.03	0.03	0.08	0.03	0.03	0.08	0.08	0.20	0.57	4.84	6.22	14.38
	GRU4Rec	0.16	0.16	0.24	0.60	0.97	2.03	0.08	0.08	0.08	0.12	0.17	0.32	0.72	0.85	1.38	6.30	8.25	16.78
	MF	0.24	0.30	0.50	0.89	1.62	3.80	0.04	0.04	0.10	0.11	0.11	0.20	0.50	0.63	0.90	0.65	0.96	2.00
	STAMP	0.03	0.06	0.16	1.11	1.72	3.72	0.03	0.06	0.16	0.12	0.17	0.32	0.39	0.48	0.81	0.70	1.21	2.67
	ItemCF	0.16	0.19	0.30	0.93	1.21	2.30	0.00	0.03	0.10	0.20	0.23	0.30	0.70	0.87	1.60	2.26	3.12	6.31
Cascade	LCRON	0.25	0.28	0.40	0.38	0.74	1.90	0.19	0.26	0.60	0.04	0.04	0.10	0.46	0.49	0.70	1.39	2.01	4.20
	FS-LTR	0.23	0.26	0.50	0.94	1.54	3.50	0.05	0.08	0.20	0.13	0.13	0.20	0.54	0.70	1.20	1.05	1.59	3.40
	RRCM	0.33	0.42	0.70	0.79	1.04	2.25	0.22	0.32	0.80	0.10	0.16	0.30	0.24	0.35	0.63	3.64	4.62	8.20
	PG	0.44	0.47	0.57	1.36	1.85	3.70	0.19	0.22	0.31	0.11	0.17	0.35	0.48	0.54	0.73	1.03	1.44	3.03
	CEM	0.47	0.49	0.62	1.56	1.86	3.30	0.19	0.26	0.44	0.12	0.15	0.28	0.39	0.43	0.58	1.14	1.57	3.30
	R4Rec	**0.81**†	**0.89**†	**1.20**†	**2.56**†	**3.19**†	**5.67**†	**0.98**†	**1.01**†	**1.27**†	**0.61**†	**0.65**†	**1.10**†	**1.09**	**1.36**†	**2.23**†	**9.03**†	**10.93**†	**18.90**†

1.1–4.5× on the other domains. Compared with static-alignment cascades, R4Rec preserves the modular separation between retrieval and reranking and coordinates the two stages directly, rather than forcing their integration through indirect surrogate signals. Its gains suggest a broader systems insight: for inherently decoupled cascades, effective coordination can be more robust than end-to-end joint optimization, which remains vulnerable to gradient conflict and train–test mismatch.

Table 2: Performance of R4Rec with various LLM-Rec rerankers as the backbone. Each entry reports LLM-Rec with Static-Global / **R4Rec**, and $\Delta\%$ denotes the relative gain of R4Rec.

Reranker Backbone	Fashion			Grocery			Clothing			Health			CDs			MovieLens		
	HR@10	NDCG@10	Δ%	HR@10	NDCG@10	Δ%	HR@10	NDCG@10	Δ%	HR@10	NDCG@10	Δ%	HR@10	NDCG@10	Δ%	HR@10	NDCG@10	Δ%
Rec-SAVER	0.48/**1.10**	0.23/**0.89**	208.06	2.20/**3.00**	0.95/**1.44**	43.97	0.16/**0.60**	0.05/**0.29**	377.50	0.20/**0.90**	0.14/**0.39**	264.29	0.57/**2.00**	0.22/**1.03**	309.53	7.72/**18.8**	3.78/**9.13**	142.53
Rec-R1	0.40/**0.80**	0.29/**0.49**	84.48	2.70/**4.90**	1.25/**2.14**	76.34	0.20/**1.40**	0.15/**0.78**	510.00	0.20/**1.00**	0.08/**0.43**	418.75	1.50/**2.00**	0.90/**0.93**	18.33	8.61/**18.2**	4.32/**9.00**	109.86
TWiCERec	0.80/**1.20**	0.57/**0.89**	53.07	2.40/**5.67**	1.27/**3.19**	143.72	0.20/**1.27**	0.13/**1.01**	605.96	0.30/**1.10**	0.16/**0.65**	286.46	1.30/**2.23**	1.11/**1.36**	47.03	6.98/**18.9**	4.03/**10.9**	170.62
Rank-R1	0.97/**1.05**	0.68/**0.74**	8.54	2.43/**5.18**	1.27/**2.54**	106.58	0.32/**1.13**	0.14/**0.69**	322.99	0.32/**1.05**	0.14/**0.50**	242.63	1.50/**1.78**	0.91/**1.02**	15.38	12.3/**14.3**	6.75/**7.84**	16.20
ReRe	0.90/**0.65**	0.62/**0.38**	−33.24	2.50/**4.70**	1.09/**2.05**	88.04	0.30/**0.97**	0.20/**0.55**	199.17	0.20/**0.97**	0.08/**0.43**	411.25	1.20/**2.11**	0.72/**1.07**	62.22	12.6/**13.3**	6.15/**6.16**	2.86
f-GRPO	0.55/**0.81**	0.37/**0.55**	47.96	2.19/**4.94**	1.03/**2.17**	118.13	0.65/**1.05**	0.30/**0.67**	92.44	0.16/**0.97**	0.09/**0.47**	464.24	1.40/**1.94**	0.86/**1.19**	38.47	12.5/**12.6**	5.89/**5.72**	−1.04

*$\Delta\%$ is the average of the relative improvements in HR@10 and NDCG@10.

(2) The gain holds across LLM-Rec backbones, and comes from conditional reranking. Because processing a large number of candidates in an LLM prompt is inefficient, existing LLM-Rec methods typically assume that a strong candidate set has already been retrieved and focus only on reranking it (Lin et al., 2025). R4Rec is complementary to these methods: it can use any LLM-Rec model as its reranker while coordinating how its candidate set is generated and refined. We therefore pair each of six LLM-Rec backbones with either STATIC-GLOBAL—the strongest fixed retriever allocation, selected per dataset on validation and frozen at test—or full R4Rec. The two conditions use the same retriever bank, retrieval budget, reranker, and prompt, differing only in whether retriever allocation and candidate refinement adapt to each request. As shown in Table 2, R4Rec improves HR@10 across nearly all six backbones. This indicates that the gain stems not from a particular LLM-Rec model but from coordinating retrieval and reranking so that each reranker receives a candidate set it can rank more accurately.

We further examine whether R4Rec improves HR@10 by increasing retrieval recall or by constructing candidate sets that the reranker can resolve more effectively. For each method, HR@10 can be decomposed as

$$\Pr(y \in R_{1:10}) \;=\; \underbrace{\Pr(y \in D)}_{\text{recall } a} \cdot \underbrace{\Pr(y \in R_{1:10} \mid y \in D)}_{\text{conditional rerank } b}, \tag{3}$$

where $a = \frac{\#\{y \in D\}}{\#\text{requests}}$, $b = \frac{\#\{y \in R_{1:10}\}}{\#\{y \in D\}}$, i.e., a is the retrieval recall and b is the conditional reranking success rate given retrieval. Since the reranker only ranks items in D, $a \cdot b$ exactly equals HR@10. Table 3 reports (a_0, b_0) for the LLM-Rec with STATIC-GLOBAL and (a_1, b_1) for R4Rec. Note that we use TWiCERec (Park et al., 2026) as the fixed LLM-Rec reranker. The resulting HR@10 gain is decomposed as

$$a_1 b_1 - a_0 b_0 \;=\; \underbrace{(a_1 - a_0)\,\frac{b_0 + b_1}{2}}_{\text{from recall (A)}} \;+\; \underbrace{(b_1 - b_0)\,\frac{(a_0 + a_1)}{2}}_{\text{from conditional rerank (B)}}. \tag{4}$$

As shown in Table 3, retrieval recall improves across most domains ($a_1 > a_0$), with its contribution (A) remaining modest in several datasets. Conditional reranking success also increases substantially ($b_1 > b_0$), and contribution (B) is larger in Fashion, Grocery, Clothing, Health, while the two contributions are comparable in CDs and MovieLens. Thus, under the same retrieval budget, R4Rec improves HR@K through both improved retrieval and more reranker-resolvable candidate sets, with

Table 3: Decomposition of HR@10 gains into retrieval recall and conditional reranking success.

Dataset	a_0	b_0	$HR_0@10$	a_1	b_1	$HR_1@10$	(A)Recall Gain	(B)Rerank Gain
Fashion	3.30	24.24	0.80	3.10	38.71	1.20	-0.06	0.46
Grocery	15.70	15.29	2.40	20.40	27.79	5.67	1.01	2.26
Clothing	2.10	9.52	0.20	2.90	43.79	1.27	0.21	0.86
Health	2.25	13.33	0.30	3.20	34.38	1.10	0.23	0.57
CDs	4.30	30.23	1.30	5.61	39.75	2.23	0.46	0.47
MovieLens	25.19	27.71	6.98	41.10	45.99	18.90	5.86	6.06

the latter providing a substantial source of gain across domains and thereby reducing the retriever–reranker objective gap. Appendix F.4 further verifies the reranking effect at the request level: with retrieval success held fixed, R4Rec's allocation yields significantly better and more stable gold ranks than Static-Global's, and refinement does the same for the candidate sets it revises.

4.3 Effect of Across-Sample Reallocation (RQ2)

We evaluate across-sample reallocation along two axes: (1) the retriever allocation policy and (2) its adaptation to user context. All experiments disable within-sample refinement so that it does not mask the allocation effect, and share the same retriever bank, reranker, and candidate size K_D.

Table 4: Ablation of across-sample reallocation.

Domain	Metric	Single	Uniform	Recall Rank	Rank+Gate
Fashion	NDCG@10	0.226	0.558	0.570 0.540	**0.650**
	NDCG@20	0.265	0.653	0.717 0.661	**0.800**
Clothing	NDCG@10	0.051	0.189	0.157 0.154	**0.240**
	NDCG@20	0.073	0.308	0.236 0.280	**0.270**
Grocery	NDCG@10	0.317	0.691	0.693 1.057	**1.280**
	NDCG@20	0.534	1.013	1.168 1.424	**1.870**
Health	NDCG@10	0.074	0.142	0.222 0.316	**0.315**
	NDCG@20	0.123	0.142	0.326 0.415	**0.440**
CDs	NDCG@10	0.225	0.321	0.345 0.931	**1.194**
	NDCG@20	0.419	0.625	0.528 1.363	**1.589**
MovieLens	NDCG@10	3.730	3.179	2.929 6.162	**7.860**
	NDCG@20	4.906	4.116	4.022 7.753	**9.470**

(1) Allocation and Reward Ablation. R4Rec learns a reranker-aligned retriever mixture through the across-sample allocation bandit. We decouple two design axes of across-sample reallocation: (i) how retriever contributions are assigned and (ii) whether the cascade-success gate is applied. Table 4 compares four allocation/credit variants: **single**, which uses the highest-recall single retriever, typically SASRec; **uniform**, which assigns equal quotas to all retrievers; **shapley-recall**, which assigns Shapley contributions based only on whether each retriever retrieves the gold item (§3.1); and **shapley-rank**, which further weights the Shapley contribution by the target item's rank in each retriever output. We additionally evaluate the Shapley-based variants with the cascade-success gate $\mathbb{1}[y \in R^{(0)}_{1:K}]$ turned on or off (+Gate). Three consistent trends emerge. First, **uniform tends to beat single**, indicating an ensemble effect from complementary candidates across retrievers. Second, **shapley-recall tends to beat uniform**, showing that learning context-dependent retriever weights helps beyond uniform mixing. Third, **shapley-rank tends to perform better than shapley-recall**, suggesting that recall-only attribution may be too coarse, as discussed in §3.1.1. Lastly, **enabling the gate is generally beneficial across credit schemes**: by rewarding a retriever only when the reranker actually keeps the gold in its top-K, the gate aligns retriever selection with downstream reranking success rather than mere retrieval recall.

(2) Context Adaptation. Bucketing requests by interaction-history length and sparsity, we find that sequence-based retrievers tend to receive higher mean weight w_k for long-history users, while semantic retrievers are favored for sparse or short-history ones (Fig. 3). The across-sample gain thus stems from context-dependent selection, not a better fixed reweighting.

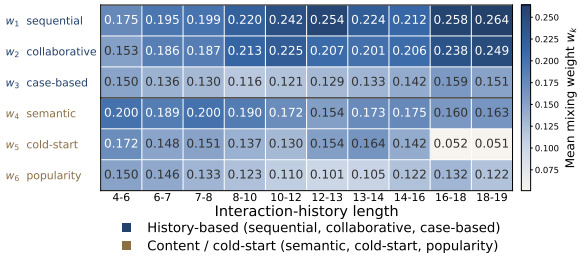

Figure 3: Mean mixing weights w_k by history length.

4.4 Effect of Within-Sample Refinement (RQ3)

Within-sample refinement is applied only to low-confidence requests. We evaluate whether it improves the accuracy–cost trade-off by outperforming no-refinement when additional inference is allowed, and by matching or exceeding always-refinement while issuing fewer refinement calls. To isolate this effect, all experiments keep the trained allocation ban-

Table 5: Trigger ablation of within-sample refinement.

Domain	Metric	No-Refine	Random	Always	Ours	Calls†
Fashion	NDCG@10	0.65	0.55	0.45	**0.89**	0.21
	NDCG@20	0.80	0.64	0.61	**0.94**	
Grocery	NDCG@10	1.28	2.29	2.19	**3.19**	0.50
	NDCG@20	1.87	2.70	2.88	**3.76**	
Clothing	NDCG@10	0.24	0.64	0.78	**1.01**	0.24
	NDCG@20	0.27	0.64	0.85	**1.08**	
Health	NDCG@10	0.32	0.39	0.43	**0.65**	0.51
	NDCG@20	0.42	0.39	0.55	**0.77**	
CDs	NDCG@10	1.19	1.32	0.82	**1.36**	0.25
	NDCG@20	1.59	1.73	1.28	**1.50**	
MovieLens	NDCG@10	7.86	7.10	2.92	**10.93**	0.25
	NDCG@20	9.47	9.00	4.20	**12.64**	

† *Calls* counts the refinement rounds of **Ours**; **Random** uses the same average refinement budget as **Ours**; **No-Refine** and **Always** are fixed at 0 and 4.

dit fixed. We compare four variants: (a) **No-Refine**, which returns the initial ranking without additional inference; (b) **Random**, which uses the same average refinement budget as Ours but randomly selects requests to refine; (c) **Always-Refine**, which applies the full B refinement rounds to every request; and (d) **Ours**, which refines only when the learned gate predicts it to be beneficial.

Table 5 shows two trends. First, **Ours outperforms No-Refine**, indicating that within-sample refinement recovers reranking failures missed by the initial pass. Second, **Ours generally outperforms Always-Refine while issuing fewer refinement calls**, showing that refinement is not uniformly beneficial: applied to already reliable rankings, it wastes computation and can even perturb well-resolved rankings. Third, **Ours outperforms Random under the same average refinement budget**, demonstrating that the gain comes from identifying the requests that benefit from refinement rather than from the number of refinement calls itself. Thus, the gain comes not from refining more, but from refining the right requests. Appendix F.3 provides qualitative examples of this behavior.

4.5 Efficiency (RQ4)

We assess whether R4Rec's performance gains come at a practical cost along three axes: per-component latency, the refinement-call distribution, and the scalability of the Shapley value. **(1) Per-Component Latency.** Table 6 breaks down per-request latency (s) across nodes; end-to-end latency assumes refinement is always invoked. Most latency comes from the LLM reranker call and the multi-retriever call, while the controller overhead (weight allocation, decision state) is negligible in the total. The refinement stage is conditional: they are entered only when the gate score satisfies $u^{(t)}_{\text{wtn}} > 0$, and are skipped entirely otherwise. The additional cost is incurred only on the subset of requests selected by the gate for refinement and is small relative to the accuracy gain.

Table 6: Per-Request Latency Breakdown (s).

Stage	Component	p50	p95	% Total
Controller	Initial Weight Allocation (Across-Sample)	0.03	0.07	0.70
	Decision State (Within-Sample)	0.01	0.04	0.13
Retrieval	Multi-Retriever Execute	1.76	2.44	37.04
	Candidate Merge	0.05	0.08	1.04
Reranking	LLM reranker	0.37	0.50	7.84
Refinement (Conditional)	Generating Diagnostic Refinement Query	0.49	0.67	10.29
	Additional Rerank (re-retrieval+reranking)	2.04	2.56	42.95
Total	**End-to-end**	-	-	**100.0**

(2) Refinement-Call Distribution. Fig. 4 reports the distribution of the per-request refinement count t. Most requests complete without refinement: only 13.9% of requests trigger any refinement, amounting to 37.1 refinement rounds per 100 requests. Refinement usage is bimodal: a single round (5.7%) or the full four-round budget (7.5%), rarely in between.

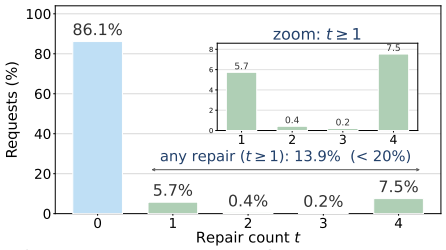

Figure 4: Per-request refinement counts.

(3) Shapley Scalability. The rank-weighted Shapley credit admits a closed form, avoiding the $O(M2^M)$ subset enumeration of the direct definition and costing $O\big(\sum_k |\widetilde{D}^{(0)}_k| + |\mathcal{T}(x,y)| \log |\mathcal{T}(x,y)|\big)$ per request. Sweeping the retriever count M, we find that naive Shapley scales exponentially while our closed form scales near-linearly, keeping the across-sample cost small.

4.6 Online A/B Test (RQ5)

To verify the offline gains in a live service, we conduct an approximately 12-week online A/B test from July to late September 2026 in an e-commerce domain with four mutually exclusive user-level arms: Random, CF-Rec, Static-Global, and Ours. All arms share the same item pool and exposure format under a pre-specified traffic allocation. We evaluate CTR and compare Ours with each baseline using logistic regression with user-clustered robust standard errors and Holm correction. As shown in Table 7, Ours achieves the highest CTR, with all improvements remaining statistically significant after correction. See Appendix G for detailed results.

Table 7: Online A/B Test Results

Model	# Users	# Impression	# Clicks	CTR
Random	35.36K	214.81K	2.06K	0.96%
CF-Rec	54.08K	330.92K	4.31K	1.30%
Static-Global	16.54K	92.31K	1.65K	1.79%
Ours	44.80K	267.93K	5.09K	**1.90%**

5 Conclusion

We present R4Rec, a retrieve–rerank–reallocate–refine framework that aligns retrieval with LLM reranking without retraining the underlying components. By combining across-sample reallocation and within-sample refinement, R4Rec adaptively constructs reranker-friendly candidate sets, improving recommendation performance across datasets and backbones with minimal overhead.

AI USE STATEMENT

We used generative AI tools solely for grammar correction and language polishing. All AI-assisted text was reviewed by the authors, who take full responsibility for the final content of this work.

ETHICS STATEMENT

Our online experiments were conducted in a production recommender system under applicable internal policies and privacy requirements. The experiments used standard recommendation interactions without exposing users to interventions beyond the normal service experience. All reported results are aggregated, and no personally identifiable information or user-level data is disclosed.

REPRODUCIBILITY STATEMENT

We provide detailed information necessary for reproducing our experiments in the Appendix, including model configurations, training and evaluation settings, implementation details, and additional experimental specifications. Furthermore, we have publicly released all of our code used in this work.

REFERENCES

Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pp. 758–759, 2009.

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.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, and Julian McAuley. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*, 2024.

Junjie Huang, Jiarui Qin, Jianghao Lin, Ziming Feng, Weinan Zhang, and Yong Yu. Unleashing the potential of multi-channel fusion in retrieval for personalized recommendations. In *Proceedings of the ACM on Web Conference 2025*, pp. 483–494, 2025.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 2333–2338, 2013.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pp. 197–206. IEEE, 2018.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pp. 1419–1428, 2017.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 661–670, 2010.

Shijun Li, Wooseong Yang, Yu Wang, Tianxin Wei, and Joydeep Ghosh. Rrcm: Ranking-driven retrieval over collaborative and meta memories for llm recommendation. *arXiv preprint arXiv:2605.07129*, 2026.

Jiacheng Lin, Tian Wang, and Kun Qian. Rec-r1: Bridging generative large language models and user-centric recommendation systems via reinforcement learning. *arXiv preprint arXiv:2503.24289*, 2025.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024.

Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. Stamp: short-term attention/memory priority model for session-based recommendation. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1831–1839, 2018.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Chung Park, Taesan Kim, Hyeongjun Yun, Dongjoon Hong, Junui Hong, Kijung Park, MinCheol Cho, Min Sung Choi, Jihwan Seok, and Jaegul Choo. Think wise, collaborate effectively: A rationale-aware llm-based recommender with reinforcement learning from collaborative signals. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 15609–15616, 2026.

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

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

Lloyd S Shapley et al. A value for n-person games. 1953.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1441–1450, 2019.

Rohan Surana, Gagan Mundada, Junda Wu, Xintong Li, Yizhu Jiao, Bowen Jin, Sizhe Zhou, Tong Yu, Ritwik Sinha, Jiawei Han, et al. F-grpo: Factorized group-relative policy optimization for unified candidate generation and ranking. *arXiv preprint arXiv:2605.12995*, 2026.

Junfei Tan, Yuxin Chen, An Zhang, Junguang Jiang, Bin Liu, Ziru Xu, Han Zhu, Jian Xu, Bo Zheng, and Xiang Wang. Reinforced preference optimization for recommendation. *arXiv preprint arXiv:2510.12211*, 2025.

Raphael Tang, Crystina Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Türe. Found in the middle: Permutation self-consistency improves listwise ranking in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2327–2340, 2024.

Alicia Y Tsai, Adam Kraft, Long Jin, Chenwei Cai, Anahita Hosseini, Taibai Xu, Zemin Zhang, Lichan Hong, Ed H Chi, and Xinyang Yi. Leveraging llm reasoning enhances personalized recommender systems. *arXiv preprint arXiv:2408.00802*, 2024.

Yunli Wang, Zhen Zhang, Zhiqiang Wang, Zixuan Yang, Yu Li, Jian Yang, Shiyang Wen, Peng Jiang, and Kun Gai. Learning cascade ranking as one network. *arXiv preprint arXiv:2503.09492*, 2025.

Lanling Xu, Zhen Tian, Gaowei Zhang, Junjie Zhang, Lei Wang, Bowen Zheng, Yifan Li, Jiakai Tang, Zeyu Zhang, Yupeng Hou, Xingyu Pan, Wayne Xin Zhao, Xu Chen, and Ji-Rong Wen. Towards a more user-friendly and easy-to-use benchmark library for recommender systems. In *SIGIR*, pp. 2837–2847. ACM, 2023.

Enming Yuan, Wei Guo, Zhicheng He, Huifeng Guo, Chengkai Liu, and Ruiming Tang. Multi-behavior sequential transformer recommender. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pp. 1642–1652, 2022.

Kai Zheng, Haijun Zhao, Rui Huang, Beichuan Zhang, Na Mou, Yanan Niu, Yang Song, Hongning Wang, and Kun Gai. Full stage learning to rank: A unified framework for multi-stage systems. In *Proceedings of the ACM Web Conference 2024*, pp. 3621–3631, 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 & data mining*, pp. 1059–1068, 2018.

Shengyao Zhuang, Xueguang Ma, Bevan Koopman, Jimmy Lin, and Guido Zuccon. Rank-r1: Enhancing reasoning in llm-based document rerankers via reinforcement learning. *arXiv preprint arXiv:2503.06034*, 2025.

A Dataset

A.1 Dataset Details

To evaluate our approach across diverse recommendation settings, we adopt five domains from **Amazon Review 2023**[1] (Hou et al., 2024) together with the **MovieLens** dataset[2] (Harper & Konstan, 2015). These benchmarks span markedly different consumption behaviors, from online retail purchasing to movie viewing, thereby exposing our method to heterogeneous interaction patterns; the corresponding statistics are reported in Table 8. From the Amazon Review collection, we select five domains of practical interest: (1) *Amazon* ***Fashion***, (2) ***Clothing*** *Shoes and Jewelry*, (3) ***Grocery*** *and Gourmet Food*, (4) ***Health*** *and Household*, and (5) ***CDs*** *and Vinyl*. We further include MovieLens 32M (***ml-32m***), a widely used large-scale benchmark for movie recommendation. These datasets also differ substantially in their interaction density: *Amazon Fashion* is characterized by short user sequences, whereas *ml-32m* is considerably denser, with an average sequence length of roughly 50 interactions per user.

Table 8: Dataset Statistics

Dataset	Domain	#Users	#Items	Avg Length	Avg Item Purchase	Sparsity
Amazon Review	Fashion	11,028	59,004	6.54	1.23	0.9994
	Clothing	13,766	107,353	11.83	1.65	0.9994
	Grocery	11,334	6,149	10.76	21.02	0.9926
	Health	21,152	74,784	8.49	2.50	0.9994
	CDs	17,035	83,774	13.71	1.97	0.9993
MovieLens	ml-32m	10,623	12,259	48.96	42.43	0.9960

A.2 Leave-One-Out Evaluation Protocol

We enforce a strict temporal separation between model/controller training and final evaluation. For each user u, the final interaction i_T is reserved exclusively as the final test target and is never used for retriever or reranker training, controller reward construction, feature computation, or hyperparameter selection. The retrievers and reranker are trained on the same historical training interactions. Both the across-sample controller and the within-sample refinement gate are supervised only using pre-test historical interactions, and therefore never observe i_T or any future interaction when constructing their rewards. We apply the same temporal restriction to all interaction-derived statistics. In particular, popularity statistics, collaborative-filtering matrices, case-based memories, retriever training data, and controller rewards are constructed solely from interactions available prior to the corresponding evaluation target. Fig.5 illustrates the temporal separation, and Table 9 summarizes the information available at each stage.

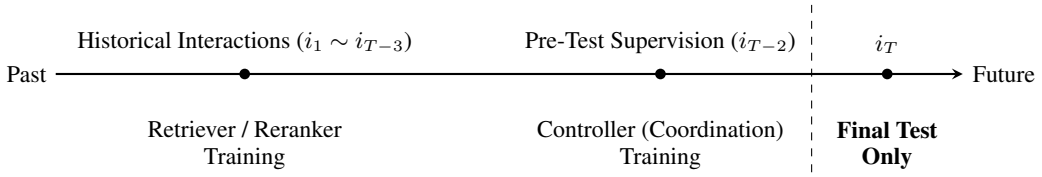

Figure 5: Temporal separation of training and evaluation. The final interaction i_T is reserved exclusively for evaluation and is never used for model training, controller reward construction, feature computation, or hyperparameter selection.

Table 9: Data access at each stage of the pipeline. The final test target i_T is inaccessible to all training and controller-learning stages.

Stage	Supervision / Input	Updated Component	Access to i_T?
Retriever training	Historical Interactions ($i_1 \sim i_{T-3}$)	Retrievers	No
Reranker training	Historical interactions ($i_1 \sim i_{T-3}$)	Reranker	No
Across-sample training	Pre-Test Historical Targets (i_{T-2})	Allocation controller	No
Refinement-gate training	Pre-test refinement gains (i_{T-2})	Refinement controller	No
Validation	Pre-test held-out interactions (i_{T-1})	Hyperparameter selection only	No
Final evaluation	Final interaction i_T	None	**Metric only**

[1] https://amazon-reviews-2023.github.io
[2] https://grouplens.org/datasets/movielens

A.3 Continual Adaptation after Deployment

The contextual-bandit formulation is motivated by continual deployment. In our controlled offline evaluation, both controllers are frozen at test time so that R4Rec has access to the same test-time information as all baselines. In deployment, however, the controllers are no longer frozen and their sufficient statistics are continuously updated from delayed item-level feedback, such as attributed clicks or purchases. Thus, the learning rule itself does not fundamentally change: historical next-item labels used for offline training are replaced by observed positive user feedback.

Across-Sample Reallocation. Suppose a served request x receives positive feedback on an exposed item y^{fb}, e.g., a click or purchase. After reranking and Shapley utility assignment, let $\mathcal{P}(x) = \{k : \mathrm{set}(\widetilde{D}_k^{(0)}) \cap D^{(0)} \neq \varnothing\}$ denote the retrievers with at least one returned item retained in the merged and truncated candidate set $D^{(0)}$. Using the logged initial retriever outputs and ranking $R^{(0)}$, we compute

$$\rho_k^{\mathrm{acr}} = \mathbf{1}\left[y^{\mathrm{fb}} \in R_{1:K}^{(0)}\right] \phi_k(x, y^{\mathrm{fb}}).$$

The gate is evaluated on the initial ranking, consistent with offline training. Using the participation set $\mathcal{P}(x)$ recorded from $D^{(0)}$, the controller applies

$$A_k \leftarrow A_k + z(x)z(x)^\top, \qquad b_k \leftarrow b_k + \rho_k^{\mathrm{acr}} z(x), \qquad k \in \mathcal{P}(x).$$

Participating retrievers are updated even when their credit is zero; all other retrievers retain their current statistics. Consequently, future retrieval quotas can adapt as the usefulness of different retrievers changes across users and over time.

Within-Sample Refinement. For a request that executes refinement, the system logs the intermediate rankings $R^{(t)}$ and $R^{(t+1)}$. Once positive feedback (e.g., click, purchase) on an item y^{fb} is observed, the same rank-gain reward used during offline training can be computed retrospectively:

$$\rho_{\mathrm{wtn}}^{(t)} = G_{y^{\mathrm{fb}}}\left(R^{(t+1)}\right) - G_{y^{\mathrm{fb}}}\left(R^{(t)}\right).$$

The refinement controller then applies the same LinUCB update using the logged state $\xi^{(t)}$. Only the outcome of an executed refinement is observed; the outcome of an unexecuted refinement remains unknown, which naturally gives rise to the exploration–exploitation setting of a contextual bandit.

Offline Evaluation versus Deployment. Freezing the controllers is therefore an evaluation choice rather than a requirement of R4Rec. Offline testing disables feedback-based updates to prevent test-label leakage and ensure a controlled comparison with baselines, whereas deployment can retain the same lightweight incremental updates as feedback arrives. A full empirical study of continual adaptation under delayed, noisy, and exposure-biased production feedback is left for future work.

B Reproducibility Details

Table 10 summarizes the notation used throughout the paper, grouped into cascade basics, the across-sample channel (Shapley-guided reallocation), and the within-sample channel (bandit-gated refinement), following the order in which the components are introduced. Table 11 reports the corresponding hyperparameter values, which are held fixed across all datasets unless noted otherwise.

All rerankers, including R4Rec, are instantiated as TWiCERec (fine-tuned GEMMA-4-4B-IT), while R4Rec's Refiner uses GEMMA-4-26B-A4B. Example prompts for reranking and refinement-query generation are provided in Fig. 6 and Fig. 7, respectively.

We conducted all experiments using four NVIDIA H200 GPUs. For reproducibility, the implementation is publicly accessible via anonymous links.

■ **Code Link**: https://anonymous.4open.science/r/R4Rec-2EC5

Table 10: Notation for R4Rec, covering the cascade setup, Shapley-Guided Retriever Allocation (Across-Sample Reallocation), and Bandit-Gated Local Refinement (Within-Sample Refinement).

Notation	Definition
Cascade Basics	
$\mathcal{I}$	Item pool
N	Number of items in the pool, $N = \lvert\mathcal{I}\rvert$
M	Number of retrievers in the retriever bank
x	User interaction history $x = (i_1, ..., i_{\lvert x\rvert})$
i	An item in the user interaction history x
$\mathcal{F}$	Bank of retrievers, $\mathcal{F} = \{f_1, \ldots, f_M\}$
f_k	A k-th retriever
D	Candidate list from retriever bank
K_D	Candidate list size $K_D = \lvert D\rvert$
g	Reranker
R	Ranking list over D produced by the reranker g
K	Size of the final top-K exposed to the user ($K < K_D < N$)
Across-Sample Reallocation (Shapley-Guided)	
$\mathcal{S}$	Coalition, a subset of retrievers whose retrieved items are pooled
v	Characteristic function of the coalition game
$D_k^{(0)}$	Deduplicated items contributed by f_k to the initial candidate after quota allocation and truncation
$\mathcal{T}(x, y)$	Set of retrievers that surface the gold item y
$r_{k,y}$	Rank at which f_k places the gold item y in its own list
$J(r)$	Logarithmically discounted rank payoff, $J(r) = \frac{1}{\log_2(r+1)}$
$\phi_k(x, y)$	Rank-weighted Shapley value of retriever k
ρ_k^{acr}	Reranker-aligned (cascade-gated) Reward
$u_k(x)$	Across-sample utility (LinUCB score) of retriever k
w_k	Mixing weight (retrieval quota) of retriever k
Within-Sample Refinement (Bandit-Gated)	
B	Maximum number of refinement rounds (refinement budget)
t	Refinement round index
$\xi^{(t)}$	State feature at round t
$\eta^{(t)}$	Retriever-side diagnostic report at round t
$\rho_{\text{wtn}}^{(t)}$	Reward of a refinement at round t
$G_y(R)$	DCG-style gain of the gold item y, $G_y(R) = \frac{1}{\log_2(\text{rank}_R(y)+1)}$
$u_{\text{wtn}}^{(t)}$	Within-sample gate score at round t
γ	Maximum fraction of candidates replaced during refinement

Table 11: Hyperparameter settings used in all experiments unless otherwise noted.

Symbol	Meaning	Value	Selection
Cascade			
M	Retrievers in the bank	12	Fixed
K_D	Candidate list size	150	Fixed
K	Exposed top-K	{5, 10, 20}	Protocol
Across-Sample Reallocation			
α	LinUCB exploration coefficient	1.0	$\{0.1, 0.5, 1.0\}$
λ	Ridge regularization	1.0	Fixed
T	Temperature	0.5	Fixed
Within-Sample Refinement			
B	Refinement budget (max rounds)	4	$\{1, 2, 4\}$
γ	Max fraction replaced	0.3	$\{0.1, 0.3, 0.5\}$
τ	Gate threshold on $u_{\text{wtn}}^{(t)}$	0.0	Fixed

Example Prompt for the LLM Reranker

You are a fashion recommendation expert.
(Purchase History)
title: Red Cotton Polo Shirt
- item id: B01LWBTBKQ
- description: Short-sleeve pique polo, 10% cotton.
- review: Great fit, soft fabric. Runs slightly large.
- rating: Excellent title: Blue Denim Jeans
- item id: B073Q8TLZN
- description: Straight-leg mid-rise denim.
- review: Classic cut, held up after many washes.
- rating: Good
(Candidate List)
title: Navy Pique Polo Shirt
- item id: B07X3Y8M2K
- description: Classic-fit cotton polo with ribbed collar.
title: Nylon Windbreaker
- item id: B07J2NKQPL
- description: Lightweight packable shell. (truncate)

Based on the (Purchase History), rank the candidate items from most suitable to least suitable for this customer.
Carefully examine:
- item titles
- item descriptions
- item details
- customer reviews
- customer ratings
Ratings indicate preference strength:
- Excellent and Good are positive signals.
- Okay is a mixed or weak signal.
- Bad and Terrible are negative signals.
Return only the reordered candidate item IDs inside `<ranking>`tags, comma-separated.

Output format: `<ranking>`item id 1, item id 2, item id 3, ... `</ranking>`

Figure 6: Reranking Prompt

Example Prompt for the LLM Refiner

You are a retrieval-refinement agent for a recommendation system. The initial ranking was flagged as uncertain. Write ONE short retrieval query that will be used to fetch NEW candidate items the first pass may have missed. Make the query a concrete, imperative description of the items to find — name specific product types, categories, materials, styles, use-cases, and brands (e.g. 'leather belts, stainless-steel watches, and minimalist wallets for smart-casual outfits'). Target the intent gap the current top-K does not cover, given the user's history. Avoid vague phrasing and meta-commentary. Do NOT include item IDs. Do NOT refer to retrievers or scores. Reply ONLY with a JSON object: {"query": "...", "rationale": "..."} — no other text.

(Purchase History)
- title: Red Cotton Polo Shirt; rating: Excellent; review: "Great fit, soft fabric. Runs slightly large."; item id: B01LWBTBKQ
- title: Blue Denim Jeans; rating: Good; review: "Classic cut, held up after many washes."; item id: B073Q8TLZN
- title: Nylon Windbreaker; rating: Bad; review: "Material felt cheap and the zipper stuck."; item id: B07J2NKQPL
(Current Top-5)
1. B08J5R4PQ1 — Black Cotton Polo
2. B07X3Y8M2K — Navy Pique Polo
Shirt 3. B09W1K3SX2 — Black Cotton Tee
4. B05XYZ1234 — Grey Cotton Hoodie
5. B0919BLJMD — Full-Grain Leather Belt

(Confidence Breakdown) overall=0.60, score gap=0.03, agreement=0.38, variance=0.52, source coverage=0.62

Figure 7: Refinement-Query Generation Prompt

C Extended Related Work

C.1 LLM-Based Reranker

In a typical retrieve–rerank recommendation pipeline, the reranker is expected to make finer-grained preference judgments than the retriever by reordering a small set of relevant candidates according to the user's interests. Accordingly, prior work has often employed increasingly sophisticated and computationally heavy reranking models. More recently, LLMs have emerged as a natural alternative for this role. An LLM-based reranker is tuned to take the user history and retrieved candidates as input and generate a reordered candidate list that better reflects the preferences inferred from the user's past interactions. In this work, we consider the following LLM-based rerankers.

Rec-SAVER (Tsai et al., 2024) studies LLM-based recommendation with supervised rationale learning, training the model on target items together with explanatory rationales. The method shows that rationale-guided instruction tuning can strengthen recommendation quality under both zero-shot transfer and task-specific fine-tuning, including non-overlapping cross-domain settings. We use our own in-house implementation for this baseline.

TWiCERec (Park et al., 2026) augments LLM recommendation with collaborative filtering signals, combining rationale-oriented instruction tuning with CF-based guidance. Its training further incorporates confidence-aware reinforcement learning, where the reward reflects both recommendation correctness and the confidence of an external CF model. This baseline is particularly relevant to our LLM-based setting and showed competitive results in the zero-shot non-overlapping cross-domain scenario. Our implementation is based on the authors' official repository[3].

Rec-R1 (Lin et al., 2025) introduces GRPO-style reinforcement learning into recommendation, using next-item correctness as the optimization signal. In this setup, the model is rewarded when the generated item matches the ground-truth next interaction. We implemented this baseline on top of a public GRPO codebase[4].

Rank-R1 (Zhuang et al., 2025) enhances LLM-based setwise reranking with explicit reasoning trained via GRPO. It uses a simple verifiable reward based on whether the model selects the ground-truth relevant document, requiring no reasoning annotations. This baseline is particularly relevant as an RL-trained reasoning reranker over a fixed candidate set. We use our own in-house implementation for this baseline.

ReRe (Tan et al., 2025) adapts reinforcement learning with verifiable rewards (RLVR) to LLM-based recommendation. It uses constrained beam search to generate diverse on-policy hard negatives and augments binary correctness rewards with ranking rewards for finer-grained supervision. This baseline is particularly relevant as a recommendation-specific RL method that jointly improves negative sampling and ranking supervision. We refer to the authors' official repository for implementation details[5].

f-GRPO (Surana et al., 2026) jointly generates and ranks candidates within a single autoregressive LLM rollout. It factorizes candidate generation and ranking with separate group-relative advantages based on coverage and position-aware ranking rewards. This baseline is particularly relevant as an end-to-end approach that directly optimizes both candidate generation and downstream ranking.

C.2 Static-Alignment Cascaded Recommender

In cascaded recommendation, another line of work focuses on narrowing the retriever–reranker objective gap by improving the coordination between retrieval and downstream ranking. These methods align the cascade through different mechanisms, including joint or full-stage supervision across retrieval and ranking, optimized fusion of multiple retrieval channels, and learned retrieval policies guided by downstream recommendation quality. Representative approaches include LCRON and FS-LTR for cross-stage alignment, CEM and PG for global or personalized multi-retriever fusion, and RRCM for ranking-driven retrieval. While these approaches improve the candidates delivered to the downstream ranker, they do not explicitly revise the current candidate set based on the realized reranking outcome of each request. In this work, we use them as cascade-alignment baselines and employ the same LLM-based reranker whenever their formulation permits, isolating the effect of how retrieval and reranking are coordinated.

LCRON (Wang et al., 2025) jointly trains retrieval and ranking stages as a unified cascade ranking network. It introduces a surrogate loss based on the ground-truth survival probability across stages, together with auxiliary stage-wise losses for additional supervision. Unlike our LLM-based setting, LCRON uses DSSM (Huang et al., 2013) as the retriever and DIN (Zhou et al., 2018) as the ranker, and jointly optimizes these conventional models to study end-to-end cascade alignment. We refer to the authors' official repository for implementation details[6].

[3] https://github.com/cpark88/TWiCE-Rec
[4] https://github.com/huggingface/open-r1
[5] https://github.com/sober-clever/ReRe
[6] https://github.com/Kwai/LCRON

FS-LTR (Zheng et al., 2024) uses full-stage supervision to align each stage of a multi-stage recommendation pipeline with downstream selection behavior and final user feedback. It relabels items according to how far they progress through the cascade and trains each stage with the resulting learning-to-rank supervision. For our implementation, we retain the original FS-LTR supervision but replace its conventional ranking model with the same LLM-based reranker used in our framework, enabling a controlled comparison under an identical reranker.

RRCM (Li et al., 2026) formulates LLM recommendation as adaptive evidence acquisition over collaborative and item-metadata memories. It learns when and what to retrieve through GRPO, using final top-k recommendation quality as the reward for retrieval and reasoning decisions. For our implementation, we retain RRCM's adaptive dual-memory retrieval and ranking-driven supervision, while adapting its final recommendation stage to use the same LLM-based reranker as our framework for a controlled comparison.

CEM and PG (Huang et al., 2025) optimize multi-channel retrieval by learning how much candidate quota to allocate to each retrieval channel. CEM learns a globally shared allocation through black-box optimization, whereas PG produces user-specific channel weights through a policy-gradient objective. Since these methods operate only on candidate fusion before the ranking stage, we retain their original retrieval-allocation strategies and use the same LLM-based reranker as our framework for downstream ranking.

D CONTEXT FEATURE SPECIFICATION $z(x)$

For each request x the controller computes a deterministic, model-free context vector $z(x) \in \mathbb{R}^{29}$ from the user's purchase history only (titles, descriptions, reviews, ratings, and item metadata). Every entry is bounded to roughly $[-1, 1]$ or $[0, 1]$ and is computed in $O(|x|)$ over the history $x = (i_1, ..., i_{|x|})$. The context feature $z(x)$ is intentionally lightweight, incurring only a small per-request overhead for the contextual bandit (§4.5). It serves as the shared backbone of both mechanisms' representations: the across-sample allocation policy uses $z(x)$ directly, while the within-sample refinement gate augments it with round-dependent features to form $\xi^{(t)}$, giving the two policies a common, reproducible request representation. This restraint is deliberate: the retriever is by design a lightweight first-stage filter whose role is to cheaply narrow the catalog to items relevant to the user context, so enriching its coordination layer with heavy features would run counter to this role and, in production serving, introduce latency that such cascades cannot afford. Table 12 gives the full schema.

Table 12: The 29-dimensional context vector $z(x)$. $L = |x|$ is the history length; ratings map to $\{\text{Excellent,Good,Neutral,Bad,Terrible}\} \to [-1, 1]$; sentiment uses a small fixed polarity lexicon.

Idx	Group	Name	Definition (Range)
0	History Size	`log_hist_len`	$\log(1+L)/10$ ($[0, \sim 0.7]$)
1–4		`hist_bucket`	one-hot of $L \in \{0\}, [1, 3], [4, 10], [11, \infty)$
5	Review Text	`avg_review_len`	$\log(1+\overline{\text{len}})/10$, mean review length
6		`sentiment_mean`	lexicon polarity $\frac{\#\text{pos}-\#\text{neg}}{\#\text{pos}+\#\text{neg}} \in [-1, 1]$
7	Ratings	`rating_mean`	mean mapped rating $\in [-1, 1]$
8		`rating_var`	$\min(1,\ 2\,\text{Var}[\text{rating}]) \in [0, 1]$
12		`has_neg_review`	$\mathbf{1}[\exists$ rating $\in \{\text{Bad,Terrible}\}]$
9	Diversity	`category_entropy`	normalized Shannon entropy of history categories $H/\log N_{\text{cat}} \in [0, 1]$
10	Cold / Recency	`days_since_last`	recency of last interaction (log days); reserved, 0 when no timestamps
11		`is_cold_user`	$\mathbf{1}[L < 3]$
13–28	Text Embedding	`hist_embed`	16-dim hashed bag-of-words average over history title/description/review, L_2-normalized

(1) History Size (idx 0–4, 5 dims). A log-compressed length plus a one-hot bucketization, so the policy can react sharply to cold ($L<3$), short, medium, and long histories.

(2) Review Text (idx 5–6). The mean review length (log-normalized) proxies engagement depth; `sentiment_mean` is the average polarity from a fixed positive/negative word lexicon over review tokens, in $[-1, 1]$ (0 when no polarity words occur).

(3) Ratings (idx 7, 8, 12). The five ordinal ratings map to $[-1, 1]$; we encode their mean, a capped variance (taste consistency), and a flag for any strongly negative review.

(4) Category Diversity (idx 9). Shannon entropy over the categorical labels of the history items, normalized by $\log N_{\text{cat}}(x)$, where a label is the item's `category` field and $N_{\text{cat}}(x)$ is the number of distinct labels observed in this user's history—a per-sample quantity, not a global catalog constant. The normalizer vanishes when $N_{\text{cat}}(x) \leq 1$, in which case we set the feature to 0 by convention.

(5) Cold / Recency (idx 10–11). A cold-user flag and a reserved recency slot (set to 0 when the input carries no timestamps), so the schema extends without changing dimensionality.

(6) Textual Embedding (idx 13–28, 16 dims). A model-free *feature-hashing* embedding: each token of the history's title, description, and review fields is hashed (fixed seed) into one of 16 buckets; the per-bucket counts are averaged over all tokens and L_2-normalized. This captures coarse textual identity of the user's recent interactions without any learned encoder or external dependency, keeping $z(x)$ fully deterministic and instant to compute. While a learned text embedding model could yield richer features, it would substantially inflate the feature dimensionality and burden the retriever, undermining the low-latency, lightweight profile that makes the retriever deployable in production; we therefore deliberately avoid it, keeping the retriever lightweight by design.

E Retriever Bank

The bank $\mathcal{F} = \{f_1, \ldots, f_M\}$ is a set of frozen, black-box retrievers exposed through one uniform interface: given a request x (the user history), each f_k scores the global item catalog and returns its top candidates with internal scores. The contextual bandit treats each f_k as an arm and, from the context $z(x)$ (Appendix D), produces mixing weights $w \in \Delta^M$; weight w_k sets retriever k's candidate quota $\propto w_k$. We group the 12 implemented retrievers into four families.

Every f_k implements $\textsc{retrieve}(x) \to \{(\text{item}, \text{score}, \text{rank})\}$ and is queried over the same global catalog; all retrievers remain frozen throughout training and inference. This frozen, tool-style design allows the contextual bandit to treat $\mathcal{F}$ as a fixed action space and learn the allocation w while keeping the retrievers fixed. Adding or removing a retriever only changes M and the bandit's arm set; the rest of the system is unchanged.

(1) Sequential Model. These consume the time-ordered history $x = (i_1, \ldots, i_{|x|})$ and predict the next item. SASRec (Kang & McAuley, 2018) is a causal self-attention encoder trained with next-item cross-entropy; GRU4Rec (Hidasi et al., 2015) is a GRU session encoder; BERT4Rec (Sun et al., 2019) is a bidirectional transformer trained with a cloze (masked-item) objective; NARM (Li et al., 2017) combines a GRU-based global encoder with an attentive local encoder to capture sequential behavior and session intent; STAMP (Liu et al., 2018) explicitly prioritizes short-term intent from the last click while modeling general session interest. These models are most effective when sequential order or recency carries the predictive signal.

(2) Collaborative Filtering. Matrix Factorization (MF) (Koren et al., 2009) factorizes the user–item interaction matrix with a regularized squared-error objective optimized by stochastic gradient descent (SGD); ItemCF is a co-occurrence cosine baseline. These capture global co-purchase structure and are strong on dense catalogs.

(3) Popularity Baseline. This baseline ranks items by their global interaction frequency in the training data, excluding items already observed in the user's history. It serves as a robust fallback for cold and sparse users.

(4) Content / Semantic. The Semantic Retriever ranks catalog items by TF–IDF similarity between item text and the interaction history, whereas the Dense Semantic Retriever replaces TF–IDF with a pretrained sentence-Transformer (`bge-small-en-v1.5`) and cosine similarity over cached catalog embeddings. The Cold-Start baseline combines semantic similarity with item recency to promote

fresh, long-tail items. Because they score items directly from text, these three retrievers are the only ones that can recall items absent from the interaction logs, making them essential when the gold item is unseen (i.e., under large cold-item fractions). The Case-based baseline is a non-parametric memory populated from interaction logs that predate the training window: each record pairs a past user context (the interaction history) with the next item that user subsequently purchased, both stored in textual form (history $\rightarrow$ next-item). At inference, given the current request, it retrieves the stored histories most similar to the query under Jaccard token similarity and recommends their associated next items, yielding an instance-level signal without any learned parameters.

Crucially, these text-based retrievers expose a common text-query interface, which within-sample refinement exploits: when refinement is invoked, it re-queries the retrievers that consume the user context as text (Semantic, Dense Semantic, Cold-Start) with the injected query, while ID-only retrievers fall back to their standard output.

F Decision State Feature $\xi^{(t)}$ and Refinement Process

At refinement round t (0-indexed), $D^{(t)}$ is the set of candidate items and $R^{(t)}$ is the frozen reranker's ordering of them. Each candidate $d \in D^{(t)}$ is described by three retriever-side quantities. First, $\mathrm{src}(d) = \{f_k \in \mathcal{F} : d \in D_k^{(t)}\}$ denotes the retrievers that returned item d, and $r_k(d)$ its 1-indexed rank in retriever k's returned list, where

$$n_k = \mathrm{clip}(\mathrm{round}(K_D \cdot w_k \cdot \mu),\, n_{\min},\, n_{\max}) \tag{5}$$

is the per-retriever pool size induced by the selector weight w_k, K_D is the target size of the fused pool, and $\mu > 1$ oversamples each list so that the union still reaches K_D once cross-retriever duplicates are collapsed (we use $K_D = 150$, $\mu = 2$, $n_{\min} = 10$, $n_{\max} = 100$). Second, let $s_k(d)$ denote the raw relevance score assigned to item d by retriever f_k. Since different retrievers may use different scoring functions and scales, we normalize these scores within each retriever's returned list:

$$\hat{s}_k(d) = \frac{s_k(d) - \min_{d' \in D_k^{(t)}} s_k(d')}{\max_{d' \in D_k^{(t)}} s_k(d') - \min_{d' \in D_k^{(t)}} s_k(d')}, \tag{6}$$

computed within a single retriever's list so that scores from differently-scaled retrievers become comparable; when the list is constant the denominator vanishes and we set $\hat{s}_k \equiv 1$. Third, the fused cross-retriever score $s(d)$, for which we adopt a weighted variant of Reciprocal Rank Fusion (RRF) (Cormack et al., 2009),

$$s(d) \;=\; \sum_{k \in \mathrm{src}(d)} \frac{w_k}{c + r_k(d)}, \tag{7}$$

where we set $c = 60$. This consumes only ranks and selector weights and never raw score magnitudes: it is invariant to each retriever's score scale, rewards multi-retriever consensus, and lets the selector modulate how much each retriever's ordering counts toward that consensus. Since $\sum_k w_k = 1$, Eq. 7 is a convex combination of per-retriever reciprocal-rank contributions, and $w_k \equiv 1$ recovers standard RRF; the same normalization bounds it analytically, $s(d) \in \big(0,\, (c+1)^{-1}\big]$, with the upper end attained only by an item ranked first by every retriever.

Because the clips in Eq. 5 can make $\sum_k n_k$ over- or undershoot the target, we form the pool as the deduplicated union $\bigcup_k D_k^{(t)}$ and, when it exceeds K_D, truncate it to the K_D highest-scoring items under $s(\cdot)$; a union smaller than K_D is used as is. Each $r_k(d)$ is the position in retriever k's *pre-truncation* list, so $s(\cdot)$ is unaffected by the truncation it induces. We collectively denote these retriever-side quantities

$$\eta^{(t)} = \Big\{ \big(d,\ \mathrm{src}(d),\ \{r_k(d)\}_{k \in \mathrm{src}(d)},\ \{s_k(d)\}_{k \in \mathrm{src}(d)},\ \{\hat{s}_k(d)\}_{k \in \mathrm{src}(d)},\ s(d)\big) \ :\ d \in D^{(t)} \Big\}. \tag{8}$$

We write $\mathrm{clip}(x, a, b) = \min(\max(x, a), b)$ and apply it only where a raw expression can leave its intended range; unit-interval features use $\mathrm{clip}(x, 0, 1)$, while ratios of the form $|\cdot|/|\mathcal{F}|$ are inherently bounded and left unclipped. Every dimension of $\xi^{(t)}$ is placed on a common scale by a *fixed* rescaling rather than by running feature statistics. This is possible because each score-based dimension is a function of quantities that are already bounded by construction—$\hat{s}_k \in [0, 1]$ by Eq. equation 6, and

Table 13: State dimensions of $h^{(t)}_{\text{state}} = \Phi(\cdot)$ (idx 0–11) and the appended request context (idx 12–40), giving $\xi^{(t)} \in \mathbb{R}^{41}$. We write a_i (resp. d_i) for the i-th item in reranker order $\mathcal{R}_{K_{\text{top}}}$ (resp. fused-score order $\mathcal{D}_{K_{\text{top}}}$); indices 0–3 use $\mathcal{R}_{K_{\text{top}}}$, indices 6–9 use $\mathcal{D}_{K_{\text{top}}}$. Ranges are $[0, 1]$ unless noted (see §F for conventions).

idx	name	definition (range $[0, 1]$ unless noted)
0	`score_gap`	$\text{clip}\big((s(a_1){-}s(a_2))/\max(\lvert s(a_1)\rvert, \lvert s(a_2)\rvert, 10^{-6})\big)$; 0.5 if $\lvert R^{(t)}\rvert{<}2$ or $s(a_1), s(a_2)$ unavailable. *Normalized* reranker top-1/2 margin (can be negative before clipping, since reranker order need not match fused-score order).
1	`agreement`	$\lvert\text{src}(a_1)\rvert/\max(1, \lvert\mathcal{F}\rvert)$: fraction of active retrievers that surfaced the reranker's top-1.
2	`consistency`	$1 - \tanh(\text{Var}\, V)$ with $V = \{\, s_k(d) \;:\; c \in \mathcal{R}_{K_{\text{top}}},\, k \in \text{src}(d) \,\}$; 0.5 if $\lvert V\rvert{<}2$. Cross-source score agreement on the head items.
3	`src_coverage`	$\lvert\bigcup_{c\in\mathcal{R}_{K_{\text{top}}}} \text{src}(d)\rvert/\max(1, \lvert\mathcal{F}\rvert)$: distinct sources in the reranker top-K_{top}.
4	`round_used`	$\min\big(1,\ t/\max(1, B)\big)$.
5	`round_left`	$\max(0,\ B - t)/\max(1, B)$.
6	`topk_gap`	$s(d_1) - s(d_2)$ over $\mathcal{D}_{K_{\text{top}}}$; $s(d_1)$ if a single item, 0 if empty. *Raw* fused-score gap (unbounded; standardized before the bandit).
7	`topk_mean`	$\text{mean}\{s(d) : d \in \mathcal{D}_{K_{\text{top}}}\}$; 0 if empty (unbounded; standardized before the bandit).
8	`topk_std`	$\text{std}\{s(d) : d \in \mathcal{D}_{K_{\text{top}}}\}$ (population); 0 if $\lvert\mathcal{D}_{K_{\text{top}}}\rvert{<}2$ (unbounded; standardized before the bandit).
9	`candidate_coverage`	$\lvert\bigcup_{d\in\mathcal{D}_{K_{\text{top}}}} \text{src}(d)\rvert/\max(1, \lvert\mathcal{F}\rvert)$: distinct sources in the candidate top-K_{top}.
10	`cand_size`	$\min(1,\ \lvert D^{(t)}\rvert/D_{\max})$ with $D_{\max}{=}100$: softly normalized docket size.
11	`bias`	1.0 (constant).
12–40	$z(x)$	the 29-dimension request context (Appendix D).

$s(\cdot) \in \big(0, (c+1)^{-1}\big]$ by Eq. equation 7—so an analytic upper bound, and hence a constant divisor, is available for each of them without estimating anything from data. Keeping the feature map fixed means a given state always maps to the same ξ, so the Gram matrix $A^{(t)}$ accumulates outer products in a single coordinate system and the confidence widths $\|\xi\|_{(A^{(t)})^{-1}}$ remain comparable across rounds; it also bounds $\|\xi^{(t)}\|$, so no single dimension dominates $\hat{\theta}^{\top}_{\text{wtn}}\xi^{(t)}$. All variances and standard deviations are *population* statistics, so degenerate (single-element or constant) inputs evaluate to 0 rather than being undefined; any residual NaN is mapped to 0 before the vector enters the bandit—a numerical guard, since one NaN would irrecoverably corrupt $A^{(t)}$—except where an explicit convention is stated above (notably $\hat{s}_k \equiv 1$ on a constant list). The budget B is the maximum number of refinement rounds the system may perform; only the two round-progress dimensions (idx $4-5$) depend on B, so the gate policy is budget-aware, and since it is trained under a single budget we hold B fixed between training and evaluation. Finally, two length-K_{top} windows are used ($K_{\text{top}} = 5$): $\mathcal{R}_{K_{\text{top}}}$, the first K_{top} items of $R^{(t)}$ (*reranker* order), and $\mathcal{D}_{K_{\text{top}}}$, the top-$K_{\text{top}}$ of $D^{(t)}$ ranked by $s(\cdot)$ (*fused-score* order); features computed on these two windows capture complementary views of the same pool.

F.1 Decision State Feature $\xi^{(t)}$

The controller summarizes the reranking state into $\xi^{(t)} = [\, h^{(t)}_{\text{state}};\, z(x)\,] \in \mathbb{R}^{41}$, where the deterministic map Φ compresses the current ranking $R^{(t)}$, candidate pool $D^{(t)}$, and retriever diagnostics $\eta^{(t)}$ into a bounded state summary $h^{(t)}_{\text{state}} = \Phi(R^{(t)}, D^{(t)}, \eta^{(t)}) \in \mathbb{R}^{12}$, and $z(x) \in \mathbb{R}^{29}$ is the request context of Appendix D, computed once per request and reused across rounds and by both mechanisms. Table 13 specifies the full feature schema.

F.2 Refinement Gate: Decision and Learning

The gate is a contextual bandit (LinUCB) with sufficient statistics $A_{\text{wtn}} \in \mathbb{R}^{41\times 41}$ (initialized $\lambda_{\text{ridge}} I$) and $b_{\text{wtn}} \in \mathbb{R}^{41}$; let $\hat{\theta}_{\text{wtn}} = A_{\text{wtn}}^{-1} b_{\text{wtn}}$. It estimates the net value of refining directly from $\xi^{(t)}$, with the value of stopping fixed at 0.

Decision. The within-sample gate scores the current round t by

$$u_{\text{wtn}}^{(t)} = \begin{cases} \hat{\theta}_{\text{wtn}}^{\top}\xi^{(t)} + \alpha_{\text{wtn}}\sqrt{\xi^{(t)\top} A_{\text{wtn}}^{-1}\xi^{(t)}}, & \text{training (UCB),} \\ \hat{\theta}_{\text{wtn}}^{\top}\xi^{(t)}, & \text{evaluation (posterior mean),} \end{cases}$$

using the exploration bonus during training and the greedy mean at test time. It performs another refinement round if $u_{\text{wtn}}^{(t)} > 0$ and stops otherwise.

Reward (offline, gold-labeled). For each round t that executed a refinement, let $\text{rank}_R(y)$ be the 1-indexed position of the gold item y in ranking R (absent $\Rightarrow$ rank $= \infty$, hence $G_y = 0$). Define

$$\rho_{\text{wtn}}^{(t)} = G_y(R^{(t+1)}) - G_y(R^{(t)}), \qquad G_y(R) = \frac{1}{\log_2(\text{rank}_R(y)+1)}.$$

This is the DCG-style marginal rank gain of round t's refinement; a request with no executed refinement yields no signal. In training mode only, the gate applies the standard LinUCB rank-one update $A_{\text{wtn}} \leftarrow A_{\text{wtn}} + \xi^{(t)}\xi^{(t)\top}$, $b_{\text{wtn}} \leftarrow b_{\text{wtn}} + \rho_{\text{wtn}}^{(t)}\xi^{(t)}$; evaluation freezes $(A_{\text{wtn}}, b_{\text{wtn}})$.

F.3 Refinement Process

Each round proceeds as in Algorithm 1. Given the round's query $q^{(t)}$, RetrieveWithQuery re-scores the query-injectable retrievers (Semantic, Dense Semantic, Cold-Start) by injecting the query tokens into the request text; all other retrievers—Case-based and the ID-only retrievers (Sequential, Collaborative, Popularity, etc.)—ignore $q^{(t)}$ and fall back to their standard output, so the call is always well defined. Let $\mathcal{C}_{\text{new}}$ denote the resulting pool of freshly retrieved candidates. ReplaceLowScore retains the top $\lfloor(1-\gamma)|D^{(t)}|\rfloor$ candidates of the current candidates by fused score $s(\cdot)$ and refills the remaining slots with the highest-scoring members of $\mathcal{C}_{\text{new}}$ not already retained, keeping $|D^{(t+1)}|=|D^{(t)}|$ fixed. The loop terminates at the first of a gate stop or budget exhaustion ($t \geq B$).

A qualitative example follows. Fig. 8 illustrates how a single refinement round transforms the exposed list $R^{(0)}$ into $R^{(1)}$. Given a user whose history centers on physical pain management and mobility (knee supports, plantar fasciitis insoles), the initial list is dominated by household items—gift wrap, a toilet step, a contact lens case—that share no semantic axis with this intent; only the ginger extract is tangentially related through its musculoskeletal claim. The Refiner LLM diagnoses this mismatch and emits a query q that names both the recovered intent and the observed drift, grounding the reformulation in concrete history items rather than a generic category label. Re-retrieval under q replaces all five candidates: the two history anchors reappear directly as a plantar fasciitis compression sleeve and corrective orthotic insoles, while the remaining slots cover complementary modalities of pain relief (thermal, topical, and oral), so that the refined list improves alignment without collapsing into near-duplicates of a single product type.

F.4 Candidate-Set Rerankability Analysis

We further verify, at the level of individual requests, whether R4Rec's gains are explained by increased target recall alone, or by the construction of candidate sets that the downstream reranker can resolve more reliably. To remove the effect of retrieval recall, the across-sample analysis uses a common matched-gold subset consisting only of requests for which the candidate sets of all three compared allocation policies contain the gold item; every row of Panel A is therefore computed on the identical set of requests and rows are directly comparable. For each candidate set, we independently permute the input order $S=10$ times and rerank each permutation with the same frozen LLM reranker. All reranking calls share an identical prompt template and greedy decoding (temperature 0), so identical input orders always produce identical outputs; consequently, any variation across

An example of query q for Refinement Process

#Candidate List (Before Refinement)
<title>core products jeanie rub fleece pad cover only
<title>step and go 2 pack 7 toilet step (pack of 2)
<title>adecco llc portable contact lens case travel kit mirror + bottle + tweezers container holder (black)
<title>hallmark foil christmas paper with cut lines on reverse (3 rolls: 60 sq ft ttl) gift wrap, tri-pack, red, green, gold solid
<title>pure encapsulations ginger extract | supplement to support the digestive, musculoskeletal, and cardiovascular systems | 120 capsules

#Query q
Since the history centers on physical pain management and mobility (knee support, plantar fasciitis insoles) while the current Top-5 drifts into unrelated household items (gift wrap, toilet steps), retrieve orthopedic foot care, pain relief, knee compression, and physical therapy recovery supplies instead.

#Candidate List (After Refinement)
<title>plantar fasciitis compression sleeve
<title>powerstep pinnacle maxx insoles - over-pronation corrective orthotic inserts for maximum stability - plantar fasciitis relief, heel & foot pain relief & arch support insoles (m 6-6.5, f 8-8.5)
<title>sunbeam xl heating pad for back, neck, and shoulder pain relief with sponge for moist heating option, extra large 12 x 24", blue
<title>penetrex joint & muscle therapy - 2oz cream - intensive concentrate for joint and muscle recovery, premium formula with arnica, vitamin b6 and msm provides relief for back, neck, hands, feet
<title>sports research collagen peptides powder - hydrolyzed type 1 & 3 collagen protein supplement for skin, nails, bones & joints - easy mixing vital nutrients & amino acids, collagen for women & men

Figure 8: A refinement round on a user whose history centers on pain management and mobility. The Refiner LLM diagnoses the drift in $R^{(0)}$, emits the query q, and re-retrieval yields the aligned list $R^{(1)}$.

Table 14: Candidate-set rerankability under random candidate-order permutations on MovieLens. Panel A uses the common matched-gold subset in which every compared allocation's candidate set contains the gold item; Panel B uses gate-triggered refinements whose pre- and post-refinement sets both contain the gold item (first refinement round). Each set is independently shuffled S=10 times and reranked by the same frozen LLM reranker with an identical prompt and greedy decoding (temperature 0), so all cross-permutation variance is attributable to candidate-order sensitivity. Requests for which the reranker omitted the gold item from its output in any permutation are censored (no rank is imputed) and reported separately. Higher HC@10 is better; lower Rank Std. and Mean Rank are better. Best per panel in **bold**.

Method	HC@10 ↑	Rank Std. ↓	Mean Rank ↓
(A) Across-Sample Allocation (refinement disabled)			
Static-Global (ref.)	0.550	28.5	33.4
Uniform	0.562	**21.2**	24.3
Shapley + Gate (R4Rec)	**0.594**	26.0	**21.0**
(B) Within-sample refinement (gate-triggered, first round)			
Before Refinement $D^{(t)}$	0.383	28.0	**30.1**
After Refinement $D^{(t+1)}$	**0.430**	**25.0**	30.6

permutations is attributable solely to candidate-order sensitivity and is not confounded with decoding stochasticity. This design is consistent with prior observations that LLM-based listwise rankers are sensitive to input order (Liu et al., 2024; Tang et al., 2024).

We report three metrics: (i) **Hit Consistency@10**, $\mathrm{HC@10}(x) = \frac{1}{S}\sum_{s=1}^{S} \mathbf{1}\left[y \in R_{1:10}^{(s)}\right]$, computed per request and averaged over the subset (higher is better); (ii) **Gold Rank Std.**, the per-request standard deviation of the gold item's rank across the S permutations, averaged over the subset (lower is better); and (iii) **Mean Gold Rank** (lower is better). Ranks are taken over the reranker's own returned ordering of the K_D=150 candidates; requests for which the reranker omitted the gold item in any permutation are censored rather than imputed, and the censoring rate is reported separately. We report MovieLens, whose matched subset is largest (n_{acr}=823). All values are subset-conditional and hence not directly comparable to the population-level conditional success rate b of Table 3: requests on which several policies simultaneously retrieve the gold item tend to be ones where the gold item is surfaced at high ranks by multiple retrievers, which inflates absolute values relative to b_0.

Panel A of Table 14 isolates across-sample reallocation: refinement is disabled, and only the allocation policy varies under the same retriever bank, reranker, and retrieval budget. Even conditioned on every policy having already retrieved the gold item, R4Rec places the gold item into the top-10 more consistently than Static-Global (0.594 vs. 0.550 HC@10) and at a substantially higher average position (21.0 vs. 33.4). Uniform mixing occupies a distinct regime: it attains the lowest cross-permutation rank variance (21.2), but does so by producing homogeneous candidate sets in which the gold item sits at a stable yet mediocre position (24.3), and it converts this stability into top-10 hits less often than R4Rec. R4Rec instead moves the gold item upward in the list, which raises both HC@10 and mean rank at the cost of somewhat larger rank dispersion; since Rank Std. is measured over the full $K_D{=}150$ ordering, part of this dispersion reflects movement within the region of the list that the reranker actually acts upon rather than instability of the top-10 decision itself. **These request-level observations indicate that the across-sample gain is not explained by target recall alone: reranker-aligned credit acts on conditional reranking quality through the composition of the candidate set, consistent with the population-level decomposition of Eq. 3, in which the rerank term (B) dominates.**

Panel B isolates within-sample refinement. Among requests for which the learned gate actually triggered refinement at test time, we select those whose pre- and post-refinement candidate sets, $D^{(t)}$ and $D^{(t+1)}$ at the first refinement round, both contain the gold item ($n_{\text{wtn}}{=}168$); the analysis therefore measures the effect on gate-selected requests (an effect-on-treated estimate). The Before-Refinement values are markedly lower than those of Panel A, confirming that the gate indeed selects candidate sets that are fragile under reranking (consistent with §4.4). After refinement, Hit Consistency rises from 0.383 to 0.430 and cross-permutation rank variance falls from 28.0 to 25.0, while the mean gold rank is essentially unchanged (30.1 vs. 30.6). Refinement on these requests therefore does not push the gold item higher in the list on average; rather, it stabilizes where the item lands, so that a larger fraction of permutations place it inside the top-10. Local refinement thus contributes not only by recalling new targets, but also by making candidate sets in which the target is already present less order-sensitive under reranking.

Algorithm 1 One refinement round at state $(R^{(t)}, D^{(t)})$

Compute $\xi^{(t)} = [\, h^{(t)}_{\text{state}};\, z(x) \,]$ (Table 13)
if $t \geq B$ **then**
    return STOP (budget)
end if
REFINEMENT $\leftarrow [\, u^{(t)}_{\text{wtn}}(\xi^{(t)}) > 0 \,]$ ▷ § F.2
if $\neg$ REFINEMENT **then**
    return STOP (gate)
end if
$q^{(t)} \leftarrow \textsc{Refiner}_{\text{LLM}}\big(x, \xi^{(t)}, R^{(t)}_{1:K}\big)$ ▷ one natural-language query
$\mathcal{C}_{\text{new}} \leftarrow \bigcup_{k \in \mathcal{F}} \textsc{RetrieveWithQuery}(k, q^{(t)})$ ▷ query-injectable retrievers use $q^{(t)}$; Others ignore it (fallback)
$D^{(t+1)} \leftarrow \textsc{ReplaceLowScore}(D^{(t)}, \mathcal{C}_{\text{new}};\, \eta)$
$R^{(t+1)} \leftarrow \textsc{FrozenReranker}\big(D^{(t+1)}\big)$
return $(R^{(t+1)}, D^{(t+1)})$; loop to round $t{+}1$

G Details of the Online A/B Test

G.1 Experimental Setup

We evaluate the proposed method in a live e-commerce recommendation service through an approximately 12-week online A/B test conducted from July to late September 2026. Users are assigned at the user level to one of four mutually exclusive arms: **Random**, **CF-Rec**, **Static-Global**, and **Ours**. Each user remains in the same arm throughout the experiment, preventing cross-arm contamination. The four arms follow a pre-specified traffic allocation and share the same recommendation item pool and exposure format. For each recommendation opportunity, the corresponding method produces a Top-10 recommendation list. The resulting experiment contains 150,776 unique users and 905,969

Table 15: Comparison of R4Rec's two bandit-based feedback mechanisms.

	Across-Sample Reallocation (§ 3.1)	**Within-Sample Refinement (§ 3.2)**
Role	Long-horizon retriever reallocation across requests	Request-level decision of whether to refine the current candidate set
Arms	M retrievers $\{f_1, \ldots, f_M\}$; UCB scores converted to soft quotas via a temperature-T softmax	Two actions, REFINEMENT vs. STOP, with the value of STOP fixed at 0
Context	Request context $z(x) \in \mathbb{R}^{29}$ (Appendix D)	Round state $\xi^{(t)} = [\, h^{(t)}_{\text{state}};\, z(x)\,] \in \mathbb{R}^{41}$ (Appendix F)
Reward	Cascade-gated rank-weighted Shapley credit $\rho^{\text{acr}}_k(x, y, R^{(0)}) = \mathbb{1}\big[\, y \in R^{(0)}_{1:K}\big] \cdot \phi_k(x, y)$ (Eq. 2)	Marginal rank gain of an executed refinement, $\rho^{(t)}_{\text{wtn}} = G_y\big(R^{(t+1)}\big) - G_y\big(R^{(t)}\big)$
Decision rule	Mixing weights $w_k \propto \exp\big(u_k(x)/T\big)$ set per-retriever retrieval quotas	Refine iff $u^{(t)}_{\text{wtn}} > 0$, for at most B rounds
Statistics	Per-arm (A_k, b_k), $k = \{1, \ldots, M\}$, with exploration strength α_{acr}	A single shared $(A_{\text{wtn}}, b_{\text{wtn}})$ with exploration strength α_{wtn}
Training $\rightarrow$ Test	Statistics accumulate across labeled requests (Fig. 2B); frozen at test time	Trained on gold-labeled requests (UCB); greedy posterior mean at test, no gold labels (Fig. 2C)

recommendation impressions. The numbers of users assigned to Random, CF-Rec, Static-Global, and Ours are 35,355, 54,076, 16,541, and 44,804, respectively. The corresponding numbers of impressions are 214,807, 330,921, 92,312, and 267,929. Because the traffic allocation is specified at the user level, the different sample sizes across arms reflect the experimental allocation rather than post-hoc filtering.

G.2 Evaluation Metric

We use click-through rate (CTR) as the primary online metric. For an arm m, CTR is computed as

$$\text{CTR}_m = \frac{\sum_i \text{Click}_{i,m}}{\sum_i \text{Impression}_{i,m}}, \tag{9}$$

where $\text{Click}_{i,m}$ and $\text{Impression}_{i,m}$ denote the numbers of clicks and impressions generated under arm m, respectively. The observed CTRs are 0.9571% for Random, 1.3015% for CF-Rec, 1.7853% for Static-Global, and 1.9001% for Ours. We report both the absolute CTR difference and relative CTR lift of Ours over each baseline. For a baseline b, the relative lift is defined as

$$\text{RelLift}(b) = \frac{\text{CTR}_{\text{Ours}} - \text{CTR}_b}{\text{CTR}_b} \times 100. \tag{10}$$

G.3 Statistical Significance Test

A user can generate multiple recommendation impressions during the experiment. Consequently, treating all impressions as independent observations may underestimate the uncertainty because observations from the same user can be correlated. We therefore assess statistical significance using logistic regression with user-level cluster-robust standard errors.

For each baseline $b \in \{\text{Random}, \text{CF-Rec}, \text{Static-Global}\}$, we fit a separate binary logistic regression using impressions from Ours and b:

$$\text{logit}\,(\Pr(y_{u,j} = 1)) = \alpha + \beta \mathbb{I}[m_{u,j} = \text{Ours}], \tag{11}$$

where $y_{u,j}$ indicates whether user u clicked on the j-th recommendation impression, and $\mathbb{I}[m_{u,j} = \text{Ours}]$ indicates whether the impression was generated by Ours. Standard errors are clustered by

Table 16: Detailed results of the online A/B test. Absolute lift is measured in percentage points (%p), and relative lift is computed with respect to each baseline. Adjusted p-values use Holm correction over the three comparisons with Ours.

Comparison	Baseline CTR	Ours CTR	Abs. Lift	Rel. Lift	Odds Ratio	95% CI	z	Adj. p
Ours vs. Random	0.9571%	1.9001%	+0.9430%p	+98.52%	2.0043	[1.8936, 2.1215]	23.987	1.151×10^{-126}
Ours vs. CF-Rec	1.3015%	1.9001%	+0.5986%p	+45.99%	1.4688	[1.4017, 1.5392]	16.112	4.187×10^{-58}
Ours vs. Static-Global	1.7853%	1.9001%	+0.1149%p	+6.43%	1.0656	[1.0002, 1.1353]	1.966	0.0493

hashed user ID so that arbitrary dependence among repeated impressions from the same user is allowed. The null hypothesis for each comparison is

$$H_0 : \beta = 0, \tag{12}$$

corresponding to no difference in click probability between Ours and the respective baseline. We use two-sided Wald tests for inference. Since Ours is compared against three baselines, we additionally apply the Holm correction to the three resulting p-values to control the family-wise error rate under multiple comparisons. We use a significance level of $\alpha = 0.05$ after correction.

G.4 Online Results

Table 16 summarizes the full online results. Ours obtains the highest CTR of 1.90%, compared with 0.96%, 1.30%, and 1.79% for Random, CF-Rec, and Static-Global, respectively. Relative to Random, Ours increases CTR by 0.9430 percentage points, corresponding to a 98.52% relative improvement. The corresponding odds ratio is 2.0043 with a 95% confidence interval of $[1.8936, 2.1215]$. The difference is statistically significant ($z = 23.987$, Holm-adjusted $p = 1.151 \times 10^{-126}$). Compared with CF-Rec, Ours improves CTR by 0.5986 percentage points, or 45.99% relatively. The estimated odds ratio is 1.4688 with a 95% confidence interval of $[1.4017, 1.5392]$. This difference is also statistically significant ($z = 16.112$, Holm-adjusted $p = 4.187 \times 10^{-58}$). Compared with the strongest baseline, Static-Global, Ours improves CTR from 1.7853% to 1.9001%. This corresponds to an absolute improvement of 0.1149 percentage points and a relative CTR lift of 6.43%. The odds ratio is 1.0656 with a 95% confidence interval of $[1.0002, 1.1353]$. The improvement remains statistically significant after correcting for the three pairwise comparisons ($z = 1.966$, Holm-adjusted $p = 0.0493$).

Overall, the online experiment shows that Ours achieves higher CTR than all three comparison methods. Importantly, the improvement over Static-Global, which provides the strongest baseline CTR, remains significant after accounting for repeated observations within users and multiple hypothesis testing.

H Closed-Form Shapley Value of the Retriever-Stage Coalition Game

This section provides the formal basis for the Shapley-guided retriever reallocation introduced in §3.1.1. We first define the retriever-stage recall game and its rank-weighted refinement, then derive both closed-form Shapley values, and finally explain how the cascade-success gate connects this retriever-stage reallocation to the LLM reranker's top-K decision.

H.1 Game-Theoretic Setup

Fix a single labeled request (x, y), where x is the user context and $y \in \mathcal{I}$ is the held-out gold item. Let $\mathcal{F} = \{f_1, \ldots, f_M\}$ be the retriever bank. For each retriever f_k, let $\widetilde{D}_k^{(0)} = (d_{k,1}, d_{k,2}, \ldots)$ denote the quota-truncated ranked list returned by f_k before cross-retriever deduplication, after removing duplicate items within the same retriever output, and let $D_k^{(0)} = \texttt{set}(\widetilde{D}_k^{(0)})$ denote the corresponding item set. The initial candidate $D^{(0)}$ is obtained by merging these retriever outputs, removing duplicates across retrievers, and truncating to the candidate size.

Define the set of retrievers that surface the gold item as

$$\mathcal{T}(x, y) = \{k : y \in D_k^{(0)}\}, \qquad N = |\mathcal{T}(x, y)|. \tag{13}$$

For $k \in \mathcal{T}(x,y)$, let $r_{k,y}$ denote the position of y in the ranked list $\widetilde{D}_k^{(0)}$; for $k \notin \mathcal{T}(x,y)$, set $r_{k,y} = \infty$. We first assign a logarithmically discounted rank payoff $J(r) = \frac{1}{\log_2(r+1)}$, which mirrors the logarithmic rank discount used in discounted cumulative gain and normalized discounted cumulative gain (NDCG): a retriever receives full credit when it ranks the gold item first, and the credit is gradually discounted as the gold item appears lower in its ranked list. To preserve a unit amount of retriever-stage credit per labeled request, we normalize this payoff by the best gold-surfacing retriever in the current request. For $k \in \mathcal{T}(x,y)$, define

$$\bar{J}_{k,y} = \frac{J(r_{k,y})}{\max_{\ell \in \mathcal{T}(x,y)} J(r_{\ell,y})}, \tag{14}$$

and set $\bar{J}_{k,y} = 0$ for $k \notin \mathcal{T}(x,y)$. When $\mathcal{T}(x,y) \neq \emptyset$, the best-ranking gold-surfacing retriever therefore has normalized payoff 1, while lower-ranking gold-surfacing retrievers receive smaller positive payoffs. We consider two cooperative games over the retriever bank.

Recall game. A coalition $\mathcal{S}$ succeeds if some retriever in it surfaces the gold item:

$$v_{\text{recall}}(\mathcal{S} \mid x,y) \;=\; \mathbf{1}\left[\, y \in \bigcup_{k \in \mathcal{S}} D_k^{(0)} \,\right]. \tag{15}$$

Rank-Weighted Recall Game. A coalition's value is the normalized rank-discounted payoff of its best-ranking gold-surfacing retriever:

$$v_{\text{rank}}(\mathcal{S} \mid x,y) \;=\; \max_{k \in \mathcal{S} \cap \mathcal{T}(x,y)} \bar{J}_{k,y}, \qquad v_{\text{rank}}(\mathcal{S}) = 0 \text{ if } \mathcal{S} \cap \mathcal{T}(x,y) = \emptyset. \tag{16}$$

We suppress the arguments (x,y) when clear from context. Both games are monotone. The rank-weighted game is also submodular: adding more retrievers cannot decrease a coalition's value, but the marginal gain of adding a retriever diminishes once the coalition already contains a higher-ranking gold-surfacing retriever. Moreover, if every retriever in $\mathcal{T}(x,y)$ places y at the same rank, then all normalized payoffs equal 1, and v_{rank} reduces exactly to the zero-one recall game v_{recall}.

By the classical definition of the Shapley value (Lundberg & Lee, 2017), the Shapley value of retriever k in a coalition game v is

$$\phi_k[v] \;=\; \frac{1}{M!} \sum_{\pi \in \mathfrak{S}_M} \left[v(S_k^\pi \cup \{k\}) - v(S_k^\pi) \right], \tag{17}$$

where $\mathfrak{S}_M$ is the set of all permutations of $\{1,\dots,M\}$ and S_k^π is the set of retrievers that appear before k in permutation π. Intuitively, $\phi_k[v]$ measures each retriever's contribution as its marginal effect on the coalition value—the difference between including and excluding it—averaged over all orders in which the retrievers may be added.

H.2 Closed-Form Shapley Values

We first derive the basic-recall closed form (Proposition 1), which then serves as the building block for the rank-weighted closed form (Proposition 2).

Proposition 1 (Closed-Form Shapley Value for the Recall Game)**.** *For the retriever-stage recall game in equation 15,*

$$\phi_k[v_{\text{recall}}](x,y) = \begin{cases} \dfrac{1}{N}, & k \in \mathcal{T}(x,y), \\ 0, & k \notin \mathcal{T}(x,y). \end{cases} \tag{18}$$

Proof. Write $N := N(x,y) = |\mathcal{T}(x,y)|$. For a permutation π, define the marginal contribution $\Delta_k^\pi = v_{\text{recall}}(S_k^\pi \cup \{k\}) - v_{\text{recall}}(S_k^\pi)$. The Shapley value equation 17 is the expectation $\phi_k = \mathbb{E}_\pi[\Delta_k^\pi]$ under a uniformly random permutation π over $\mathfrak{S}_M$. If $k \notin \mathcal{T}(x,y)$, adding k to any coalition cannot place y in the union, so $\Delta_k^\pi = 0$ for every π and hence $\phi_k = 0$. If $k \in \mathcal{T}(x,y)$, consider the position of k relative to the other $\mathcal{T}(x,y)$-members in π. If some retriever in $\mathcal{T}(x,y)$

precedes k, the union already contains y and $\Delta_k^\pi = 0$; otherwise k is the first $\mathcal{T}(x, y)$-member in π and the value jumps from 0 to 1, so $\Delta_k^\pi = 1$. Since $\Delta_k^\pi \in \{0, 1\}$,

$$\phi_k = \mathbb{E}_\pi[\Delta_k^\pi] = \Pr_\pi[k \text{ is the first } \mathcal{T}(x, y)\text{-member in } \pi] .$$

By symmetry, each of the N retrievers in $\mathcal{T}(x, y)$ is equally likely to appear first among them under a uniform π, so this probability is $1/N$, giving $\phi_k = 1/N$. □

Proposition 2 (Closed-Form Rank-Weighted Shapley Value)**.** *Assume $\mathcal{T}(x, y) \neq \emptyset$, and let $r_{k,y}$ denote the rank of the gold item y in retriever k's candidate list. Order the retrievers in $\mathcal{T}(x, y)$ by ascending rank,*

$$\tau_1, \tau_2, \ldots, \tau_N, \qquad r_{\tau_1,y} \leq r_{\tau_2,y} \leq \cdots \leq r_{\tau_N,y},$$

and adopt the convention $\bar{J}_{\tau_{N+1},y} = 0$. For the rank-weighted game in equation 16, the Shapley value of retriever τ_j is

$$\phi_{\pi_j}[v_{\text{rank}}](x, y) \;=\; \sum_{i=j}^{N} \frac{\bar{J}_{\tau_i,y} - \bar{J}_{\tau_{i+1},y}}{i}, \tag{19}$$

and $\phi_k[v_{\text{rank}}] = 0$ for every $k \notin \mathcal{T}(x, y)$. If $\mathcal{T}(x, y) = \emptyset$, all Shapley values are zero.

Proof. Write $\bar{J}_i := \bar{J}_{\tau_i,y}$ and $T_i := \{\tau_1, \ldots, \tau_i\}$ for the prefix of the best i retrievers in $\mathcal{T}(x, y)$; recall $N := N(x, y) = |\mathcal{T}(x, y)|$. Since J is decreasing in rank and the τ_i are sorted by ascending rank, $\bar{J}_1 \geq \bar{J}_2 \geq \cdots \geq \bar{J}_N \geq \bar{J}_{N+1} = 0$. For each $i \in \{1, \ldots, N\}$, define the indicator game $u_i(\mathcal{S}) = \mathbf{1}[\mathcal{S} \cap T_i \neq \emptyset]$.

We claim

$$v_{\text{rank}}(\mathcal{S}) = \sum_{i=1}^{N} (\bar{J}_i - \bar{J}_{i+1})\, u_i(\mathcal{S}), \qquad \forall\, \mathcal{S} \subseteq \{1, \ldots, M\}. \tag{20}$$

To verify this, fix $\mathcal{S}$ and let ℓ be the smallest index with $\tau_\ell \in \mathcal{S}$, i.e., τ_ℓ is the best-ranked gold-surfacing retriever in $\mathcal{S}$. If no such index exists, then $\mathcal{S} \cap \mathcal{T}(x, y) = \emptyset$ and both sides of equation 20 are zero. Otherwise, since $T_i = \{\tau_1, \ldots, \tau_i\}$, we have $\mathcal{S} \cap T_i \neq \emptyset$ iff $\tau_\ell \in T_i$ iff $i \geq \ell$; hence $u_i(\mathcal{S}) = 1$ precisely when $i \geq \ell$, and the right-hand side telescopes to

$$\sum_{i=\ell}^{N} (\bar{J}_i - \bar{J}_{i+1}) = \bar{J}_\ell - \bar{J}_{N+1} = \bar{J}_\ell,$$

which equals $v_{\text{rank}}(\mathcal{S})$ by definition.

The coefficients $\bar{J}_i - \bar{J}_{i+1} \geq 0$ make equation 20 a non-negative linear combination of indicator games. Each u_i is a zero-one recall game with successful player set T_i, $|T_i| = i$, so by Proposition 1, $\phi_k[u_i] = 1/i$ if $k \in T_i$ and 0 otherwise. By the linearity of the Shapley value applied to equation 20,

$$\phi_k[v_{\text{rank}}] = \sum_{i=1}^{N} (\bar{J}_i - \bar{J}_{i+1})\, \phi_k[u_i].$$

For $k \notin \mathcal{T}(x, y)$, no prefix T_i contains k, so every summand vanishes and $\phi_k = 0$. For $k = \tau_j \in \mathcal{T}(x, y)$, $k \in T_i$ iff $i \geq j$, so

$$\phi_{\tau_j}[v_{\text{rank}}] = \sum_{i=j}^{N} \frac{\bar{J}_i - \bar{J}_{i+1}}{i},$$

which is the claimed identity. □

Three consequences are immediate and useful in practice.

Corollary 1 (Efficiency over the normalized ordinal game)**.** *The rank-weighted Shapley values satisfy*

$$\sum_{k=1}^{M} \phi_k[v_{\text{rank}}](x, y) \;=\; v_{\text{rank}}(1, \ldots, M \mid x, y) \;=\; \mathbf{1}[\mathcal{T}(x, y) \neq \emptyset].$$

Thus, whenever at least one retriever surfaces the gold item, the total retriever-stage credit is exactly one; otherwise, it is zero.

Corollary 2 (Monotonicity in Rank). *For $1 \le j < j' \le N$ with $r_{\tau_j,y} < r_{\tau_{j'},y}$, we have*

$$\phi_{\tau_j}[v_{\mathrm{rank}}] > \phi_{\tau_{j'}}[v_{\mathrm{rank}}] > 0.$$

In particular, the retriever that places the gold at the best rank receives the largest credit whenever the ranks in $\mathcal{T}(x,y)$ are not all equal.

Corollary 3 (Reduction to the Recall Closed Form). *If $r_{k,y} = r^*$ for every $k \in \mathcal{T}(x,y)$, then $\bar{J}_{k,y} = 1$ for every $k \in \mathcal{T}(x,y)$, and*

$$\phi_k[v_{\mathrm{rank}}] = \begin{cases} \dfrac{1}{N}, & k \in \mathcal{T}(x,y), \\ 0, & k \notin \mathcal{T}(x,y). \end{cases}$$

Thus, the normalized rank-weighted game recovers the uniform Shapley allocation of the zero-one recall game as a special case.

Corollary 4 (Computability). *Computing $\phi_k[v_{\mathrm{rank}}]$ for all retrievers avoids the exponential subset enumeration in the direct Shapley definition. Given the per-retriever ranked outputs, we first scan the outputs to identify $\mathcal{T}(x,y)$ and the ranks $r_{k,y} : k \in \mathcal{T}(x,y)$, and then sort those retrievers by rank. This gives a per-request cost of*

$$O\left(\sum_{k=1}^{M} |\widetilde{D}_k^{(0)}| + |\mathcal{T}(x,y)| \log |\mathcal{T}(x,y)|\right),$$

instead of the $O(M2^M)$ cost of evaluating marginal contributions over all retriever coalitions. Once the retrievers in $\mathcal{T}(x,y)$ are sorted, the closed-form Shapley values in equation 19 can be computed by a single backward sweep in $O(|\mathcal{T}(x,y)|)$ time.

H.3 Axiomatic Properties

Because $\phi_k[v_{\mathrm{rank}}]$ is the Shapley value of the explicit game v_{rank}, it inherits the four classical Shapley axioms.

Efficiency. By Corollary 1, the total credit assigned to all retrievers equals the grand coalition value. Under the normalized rank-weighted game, this value is exactly one whenever the gold item is surfaced by at least one retriever and zero otherwise. Thus, retriever-stage credit is conserved rather than duplicated across multiple retrievers, while the distribution of that unit credit reflects the retrievers' ordinal ranks of the gold item.

Symmetry. If two retrievers contribute identically to every coalition, they receive the same Shapley value. In the rank-weighted game, two gold-surfacing retrievers with the same rank have the same normalized ordinal payoff and are symmetric with respect to v_{rank}; two retrievers that both fail to surface the gold item are null and also receive identical zero credit. When all gold-surfacing retrievers are tied at the same rank, the attribution reduces to the uniform $1/N$ allocation of Proposition 1.

Null player. A retriever that does not surface the gold item is a null player in v_{rank}, because adding it to any coalition never changes the coalition value. Such a retriever receives zero credit.

Additivity. For any two games v and v' over the same retriever set, $\phi[v + v'] = \phi[v] + \phi[v']$. The proof above made explicit use of this property: v_{rank} was decomposed into a non-negative linear combination of indicator games, and the additivity axiom transferred the known recall-game Shapley values to the rank-weighted game term by term.

H.4 Cascade-Success Gate

The main text uses a cascade-gated version of the normalized rank-weighted Shapley value:

$$\rho_k^{\mathrm{acr}}(x, y, R^{(0)}) \;=\; \mathbf{1}[y \in R_{1:K}^{(0)}] \cdot \phi_k[v_{\mathrm{rank}}](x,y), \tag{21}$$

where $R_{1:K}^{(0)}$ is the set of items in the top-K prefix of the initial LLM reranking output $R^{(0)}$.

This gate connects retriever-stage rank-weighted credit to cascade success. A retriever receives credit only when two conditions hold simultaneously: the retriever surfaces the gold item into its

ranked output, so that $k \in \mathcal{T}(x, y)$ and $r_{k,y} < \infty$, and the LLM reranker retains the gold item in its top-K output. Since the gate

$$\beta(x, y) := \mathbf{1}[y \in R^{(0)}_{1:K}]$$

is a global multiplicative factor shared by all retrievers for the current request, the gated vector $\beta \cdot \phi[v_{\text{rank}}]$ is itself the Shapley value of the scaled game $\beta \cdot v_{\text{rank}}$:

$$\phi_k[\beta \cdot v_{\text{rank}}] = \beta\, \phi_k[v_{\text{rank}}].$$

The cascade-gated credit therefore preserves the Shapley axioms for the scaled game. On cascade-successful requests, the total credit assigned across retrievers is one; on requests where the reranker fails to retain the gold item, all retrievers receive zero credit. The relative attribution among retrievers within a successful cascade still reflects the normalized ordinal payoffs $\bar{J}_{k,y}$: retrievers that place the gold at higher ranks receive larger Shapley shares.

This construction should be interpreted as a cascade-gated retriever-stage Shapley credit, not as the Shapley value of a full reranker-counterfactual game. The Shapley game v_{rank} attributes the retriever-stage event of surfacing the gold item under a normalized ordinal payoff; the cascade gate β then filters this credit by the observed success of the LLM reranker on the initial candidate.

H.5 Connection to LinUCB Updates

Combining Eq. 17 with the LinUCB-style update in §3.1.1, we update only retrievers in $\mathcal{P}(x)$, whose returned items are retained in the initial candidate set $D^{(0)}$:

$$A_k \leftarrow A_k + z(x)z(x)^\top, \qquad b_k \leftarrow b_k + \rho^{\text{acr}}_k(x, y, R^{(0)})z(x), \qquad k \in \mathcal{P}(x).$$

These updates include zero-reward observations; the statistics of all other retrievers remain unchanged. The predictor $z(x)^\top \hat{\theta}_k$, where $\hat{\theta}_k = A_k^{-1} b_k$, models context-dependent cascade-gated Shapley credit over retriever k's participation history. The symmetry property ensures that two retrievers with identical ordinal contributions on a request receive identical per-request credit; under identical update histories, this leads to identical posterior contributions. A retriever that never surfaces the gold item receives zero Shapley credit under this reward and therefore does not accumulate positive reward mass in b_k. Finally, for any single request, the total scalar reward assigned across all retrievers is exactly

$$\beta(x, y) \cdot v_{\text{rank}}(\{1, \ldots, M\} \mid x, y) \;\le\; 1,$$

and equals one on cascade-successful labeled requests. The result is an axiomatically grounded, normalized rank-weighted credit signal that across-sample reallocation can use to adapt retriever quotas.

I Limitations

R4Rec coordinates frozen components and thus inherits their ceiling: if no retriever in the bank surfaces the target item, neither reallocation nor refinement can recover it, and achievable recall is bounded by the union of the bank's outputs. Our reranker-aligned Shapley credit rewards retrievers for surfacing the gold item conditioned on cascade success, but does not directly credit contributions to candidate-set composition (e.g., supplying distinguishable candidates), even though our decomposition suggests that much of the gain arises from more resolvable candidate sets; principled yet tractable credit assignment for such set-level effects remains open. The same controller updates can in principle continue after deployment using attributed item-level implicit feedback such as clicks or purchases. In this work, however, we freeze the controllers during offline evaluation for a controlled comparison, and leave a full study of continual adaptation under delayed, noisy, and potentially biased production feedback to future work. Refinement further presupposes text-queryable retrievers in the bank.