RIMU: RANKING-AWARE DENSE RETRIEVAL ADAPTATION VIA CONVEX OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval underpins many modern agents, but dense retrievers built with frozen encoders do not automatically improve as verified query feedback accumulates. Query–corpus misalignment may therefore persist even when the correct target for a query is known. We introduce RIMU, a lightweight method that corrects stored corpus vectors based on the ranking of gold targets relative to their competitors. RIMU computes minimum-displacement updates through one convex quadratic program per query, then renormalizes the vectors to preserve cosine retrieval. Across ten retrieval pools and five frozen encoders, it improves N@10 over the frozen index by 13.4–27.1%. With Qwen3-Embedding-8B, it achieves the highest N@10 on seven of ten pools. Learning curves show gains as feedback accumulates, while gain-retention analyses assess whether later updates preserve earlier improvements. Updates take 2.3–27.3 ms per query on CPU, without encoder retraining or additional index entries.

1 INTRODUCTION

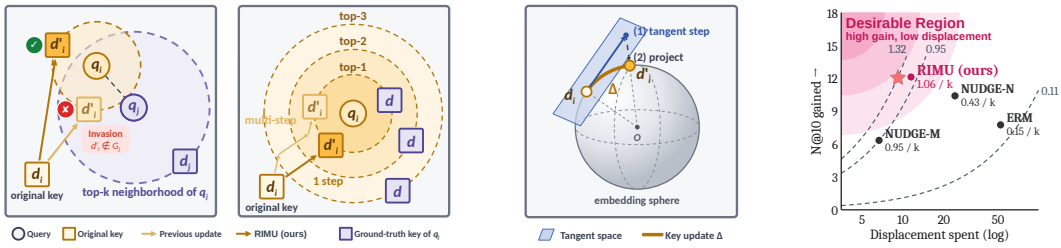
Retrieval systems commonly use lexical or embedding-based search to produce a top-K candidate set, optionally rerank it, and pass the selected results to an LLM (Karpukhin et al., 2020; Nogueira & Cho, 2019; Lewis et al., 2020). The LLM may then answer the query, invoke a retrieved tool, or search again after observing intermediate results (Qin et al., 2024; Yao et al., 2023). Yet the same information need can be expressed in different ways (Ma et al., 2023; Wang et al., 2023), while deployed retrievers serve application-specific corpora and changing query workloads (Zeighami et al., 2025; Hu et al., 2026). Matching user queries to relevant corpus representations therefore remains a challenge for reliable retrieval.

Query rewriting and query expansion can reduce query–corpus mismatch (Ma et al., 2023; Gao et al., 2023; Wang et al., 2023), as can corpus expansion (Nogueira et al., 2019). However, query-side transformations are typically repeated at inference time rather than retained as improvements from verified interactions (Hu et al., 2026). An auxiliary feedback memory can retain past corrections, but requires additional storage and retrieval as interactions accumulate (Bang et al., 2026). Repeatedly fine-tuning an embedding model or reranker also incurs training cost (Zeighami et al., 2025; Sun et al., 2025). Updating vectors already stored in the corpus offers a lightweight way to retain feedback within the existing retrieval interface (Zeighami et al., 2025; Hu et al., 2026).

Lightweight adaptation takes several forms: NUDGE and ERM update corpus representations (Zeighami et al., 2025; Hu et al., 2026), embedding adapters learn transformations over representations (Yoon et al., 2024), and online RAG systems use feedback to adapt reranking or later inference (Sun et al., 2025; Bai et al., 2025; Bang et al., 2026). For methods that edit stored corpus vectors, a central question remains: how far should a vector move, and in which direction, to repair a ranking without disrupting other queries? NUDGE maximizes alignment with positive queries under movement or normalization constraints, while ERM incorporates validated expansion signals into document keys. Neither explicitly determines the update from the relative scores of gold and competing items across the queries affected by that key.

We propose RIMU (Ranking-Informed Minimal Updates), a per-query convex optimizer that updates corpus vectors from verified retrieval experience. RIMU uses score gaps against retrieved competitors to guide ranking correction, while auxiliary ownership constraints discourage interference with already observed, similar queries. Figure 1(a) illustrates the ranking objective and this soft

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.



(a) (b) (c)

Figure 1: **Update requirements and cost.** (a) *Left*: moving a key toward its query may bring it into another query's top-k results (*retrieval intrusion*). For similar observed queries, equation 5 penalizes violations of an ownership preference through slack, without guaranteeing rank preservation. *Right*: equation 3 sets the required rank against each competitor, determining how far the key must move. (b) Updates move unit keys in the tangent space before projecting them back onto the sphere equation 4. (c) Gain versus displacement cost (keys written $\times$ mean angular displacement, in thousands of degree-keys; log axis). Dashed curves show equal gain per unit cost. Results use QWEN3-EMBEDDING-8B across ten pools and five folds. We plot the 26 pool–fold cells where all four methods wrote keys and omit the 24 cells where at least one returned the frozen index. *Adaptor* writes no keys and is excluded from this axis.

preference. A minimum-displacement objective chooses how far to move each key while penalizing unresolved ranking violations. After solving the dual program, we compute each update in the key's tangent space and renormalize the result to unit length, as illustrated in Figure 1(b). We evaluate the rankings after normalization because it can alter the margins used by the optimization.

Our contributions are as follows:

- We formulate each corpus-vector correction as a rank-aware convex optimization problem, using observed gold–competitor comparisons to determine the direction and magnitude of the update. Corrections are applied as a greedy sequence over arriving feedback rather than solved jointly.
- We analyze the role of normalization in continual vector updates. We show that repeated unnormalized updates can change retrieval rankings through norm growth, even when an irrelevant vector remains less similar by cosine.
- We introduce RIMU, which computes minimum-displacement updates in the tangent space and renormalizes vectors before writing them to the index. Auxiliary ownership constraints penalize interference among observed, similar queries, without guaranteeing their rankings.
- We evaluate RIMU on ten retrieval pools, measuring testing accuracy, learning curves, retention of earlier gains, and update cost. It achieves the best N@10 on 7 of 10 pools with Qwen3-Embedding-8B and 8 of 10 with BGE-large-en-v1.5.

2 Related Work

Query–corpus mismatch has been addressed through query expansion (Ma et al., 2023; Gao et al., 2023; Wang et al., 2023), predicted queries added to document representations (Nogueira et al., 2019), and reranking (Nogueira & Cho, 2019; Sun et al., 2023). These methods generally act on individual requests or a static index. Feedback-driven methods instead adapt retrieval from observed interactions, for example by training embedding adapters (Yoon et al., 2024), updating corpus vectors, or storing corrections in an auxiliary memory (Bang et al., 2026). Among direct vector updates, NUDGE moves relevant vectors toward positive queries (Zeighami et al., 2025), while ERM incorporates validated query expansions into document keys (Hu et al., 2026). Our method additionally constrains relative rankings and limits the interference that an update can cause for similar, previously observed queries.

Our optimization also relates to passive-aggressive learning, which finds a minimum-norm update subject to a margin constraint (Crammer et al., 2006). Here, however, the updated variables are

shared corpus vectors whose rankings affect other queries. Earlier relevance-feedback methods likewise move document vectors toward successful queries (Rocchio, 1971; Brauen, 1971), but do not explicitly constrain the resulting cross-query ranking changes. Model editing controls interference during parameter updates (Meng et al., 2022; 2023); our setting instead edits retrieved items in a shared index.

3 MINIMUM-CHANGE KEY UPDATES

3.1 PROBLEM DEFINITION

We are given target vectors $\{d_t\}_{t\in T}$, unit-norm, and a stream of training queries q_i each carrying a verified gold set $\mathcal{G}_i \subseteq T$. Retrieval ranks targets by $s(q_i, d_t) = q_i^\top d_t$ and returns the top k; write $\mathcal{N}_i$ for the non-gold targets in the top k that q_i actually retrieved, and $V = \bigcup_i \mathcal{G}_i$ for the targets any update is allowed to touch. The queries are fixed and the target vectors are the free variables, so the problem is to choose $\{d'_t\}$ that place as many golds as possible inside the top k,

$$\max_{\{d'_t\}} \sum_i \sum_{g\in\mathcal{G}_i} \mathbb{I}\big[\mathrm{rank}_{q_i}(d'_g) \le k\big], \tag{1}$$

counting retrieved golds rather than fully correct queries. At $k = 10$, this is a recall-oriented proxy for R@10; it weights queries by their gold-set size rather than averaging their individual recalls. It is not the C@10 objective, which counts a query as correct only when all its golds are in the top ten. Two things make this the wrong problem to attack directly: the objective is discrete, and a target moved for one query is read by every other query that retrieves it. The rest of this section replaces equation 1 with a convex surrogate that keeps the second fact in view.

We write each updated target as

$$d'_t \;=\; d^0_t + \Delta_t, \tag{2}$$

where Δ_t is its displacement. Only targets identified as gold in the observed feedback are writable; all other corpus vectors remain fixed. We impose ranking, ownership, and tangency constraints, then use a minimum-displacement objective to choose the updates.

Constraint 1: Ranking constraint. The first constraint encourages correction of the observed retrieval failure. At the start of cache b, let $s_b(q,t) = q^\top d_t^{(b)}$ be the score of target t for query q. A query is *triggered* when, at these cache-entry scores, its highest-scoring gold is not ranked first, some gold lies outside the top k, or that gold leads the strongest non-gold by less than 0.05; untriggered queries emit no rows and write nothing. For each triggered query and pair $(g,n) \in \mathcal{G}(q)\times\mathcal{N}(q)$ of a gold and a retrieved non-gold competitor, the implemented ranking row is

$$q^\top \Delta_g \;\ge\; m - [s_b(q,g) - s_b(q,n)] - \xi_{gn}, \qquad \xi_{gn} \ge 0. \tag{3}$$

Here d^0_g is the current vector from which Δ_g is taken, whereas the score gap on the right remains fixed at cache entry. If an earlier query has changed a shared key, the row need not express the current absolute margin. The slack variable ξ_{gn} allows a ranking constraint to be violated when all constraints cannot be satisfied simultaneously. Thus, this constraint encourages the gold target to move only as much as needed to recover the desired ordering rather than requiring every ranking relation to be satisfied exactly.

Constraint 2: Tangency. Retrieval uses unit-normalized target vectors. Enforcing $\|d^0_t + \Delta_t\|_2 = 1$ directly would make the optimization non-convex. We instead restrict each displacement to the tangent space at its current target vector, as illustrated in Figure 1(b):

$$(d^0_t)^\top \Delta_t = 0. \tag{4}$$

This linear constraint retains convexity. After solving, we write back $(d^0_t + \Delta_t)/\|d^0_t + \Delta_t\|_2$, also shown in Figure 1(b). Since $\|d^0_t + \Delta_t\|_2 = \sqrt{1 + \|\Delta_t\|_2^2}$, normalization changes the optimized scores, even though its effect is second-order for small updates. Measured over the keys the update writes, the typical one turns 9.4° and keeps 0.977 of the score the solver computed for it (Table 4).

Normalization matters especially when a target is updated repeatedly. Without it, even perfectly tangential updates increase the stored vector's norm. Under inner-product scoring, that growth can eventually make an irrelevant target outrank a relevant one despite having lower cosine similarity:

Proposition 1 (Norm-induced ranking reversal). *Let $\|q\| = \|h\| = \|z^{(0)}\| = 1$, where h is relevant to q and $z^{(0)}$ is not. Suppose the unnormalized updates satisfy $z^{(t+1)} = z^{(t)} + \Delta^{(t)}$ and $(z^{(t)})^\top \Delta^{(t)} = 0$. Define $S_T = \sum_{t<T} \|\Delta^{(t)}\|_2^2$ and $c_T = q^\top z^{(T)}/\|z^{(T)}\|_2$. If $0 < c_T < q^\top h$, the irrelevant target nevertheless outranks h by inner product whenever*

$$S_T > \left(\frac{q^\top h}{c_T}\right)^2 - 1.$$

Proposition 1 shows that norm growth *can reverse rankings* under the stated conditions; it does not imply that every unnormalized update harms retrieval. Table 5 provides empirical evidence of the risk: without post-update normalization, the tangent variant drops from 66.05 to 14.46 N@10 on ToolRet/cust. and from 56.16 to 5.04 on ToolRet/web, while improving on some other pools. We renormalize after every write to preserve cosine retrieval; Appendix B.4 gives the proof and full comparison.

Constraint 3: Ownership. A ranking correction for one query may move its gold target toward another query with a different gold target, causing retrieval intrusion. We add an ownership preference to discourage this movement. For an updated target g owned by q_{own} and a potentially affected query q_j, we prefer

$$(q_{\text{own}} - q_j)^\top \left(d_g^0 + \Delta_g\right) \;\geq\; -\eta_{gj}, \qquad \eta_{gj} \geq 0. \tag{5}$$

The implemented ownership margin is zero. The slack η_{gj} makes this a preference rather than a requirement: a violation is priced rather than forbidden. We set the ownership row price to $C_{\text{own}} = 0.05\,C$ throughout. Ranking row prices also depend on competitor scores, so this ratio does not impose a strict priority between individual ranking and ownership rows. We construct ownership rows only for already observed, similar queries in the cache that also triggered an update and have disjoint gold sets — cosine at least 0.5 to the query being solved, and an owner-minus-foreign gap of at most 0.15 — where intrusion is most likely; Appendix G.1 shows that this is where intrusion lands, and what pressing the row harder costs. This auxiliary preference targets interference among these observed queries; it guarantees neither their rankings nor those of unseen queries. Table 2 shows localized reductions in invasion and retention of training gains, with little change in overall testing N@10.

Objective. We seek the smallest update that improves the observed rankings:

$$\min_{\{\Delta_t\},\, \xi \geq 0,\, \eta \geq 0} \quad \frac{1}{2}\sum_t \|\Delta_t\|_2^2 + C\sum_{g,n} w_{gn}\xi_{gn} + C_{\text{own}}\sum_{g,j}\eta_{gj}, \tag{6}$$

subject to the ranking, tangency, and ownership constraints above. For each query, w_{gn} is the softmax of the cached competitor scores $s_b(q,n)$ over $n \in \mathcal{N}(q)$, with temperature 0.05; it is the same for every gold g of that query. Thus the ranking row price is Cw_{gn}. The displacement term keeps updated keys close to their original representations. As NUDGE also motivates, moving a key too far toward a positive query can sacrifice the information in its original embedding (Zeighami et al., 2025). We therefore prefer a small correction that improves ranking while limiting movement toward competing queries. The slack term allows ranking violations when correcting them would require excessive displacement; a smaller C places more weight on preserving the original vectors. The quadratic objective and linear constraints give a convex quadratic program.

3.2 Optimization and tangent-space update

We solve the convex program in equation 6 through its dual, whose variables λ_r correspond to the ranking and ownership constraints. Eliminating the tangency equalities gives each target update in the form

$$\Delta_t = \sum_r \lambda_r \sigma_{r,t} P_t v_r, \qquad P_t = I - d_t d_t^\top. \tag{7}$$

Here v_r is the direction specified by constraint r, $\sigma_{r,t}$ is its coefficient for target t, and P_t projects onto the tangent space at the current unit vector d_t. Hence $d_t^\top \Delta_t = 0$ by construction. After solving for the dual variables, we write the target back to the unit sphere:

$$d_t' = \frac{d_t + \Delta_t}{\|d_t + \Delta_t\|_2}. \tag{8}$$



Figure 2: **One update step.** Left: queries are retrieved against the current index $K^{(t)}$, and their updates produce $K^{(t+1)}$ for later queries. Right: each query forms ranking, ownership, and tangency constraints over its gold keys; other corpus vectors remain fixed. The resulting updates are renormalized before being written to the index. The illustration shows top-2 retrieval.

The dual derivation and solver details are in Appendix B.

3.3 The update algorithm

We process verified feedback in arrival order, using caches of at most B queries (seed-0 shuffled training order in the experiments). At cache entry, we retrieve all queries and cache their candidate identities, gold scores, and competitor scores. These retrieval quantities and the set of triggered queries remain fixed throughout the cache; there is no re-retrieval between query updates.

The implementation visits coupling components in order of their first triggered query, and visits queries within each component in arrival order. Queries are linked when they share a gold or one query's gold is another's retrieved competitor. This graph determines traversal only: every query still has its own QP. Distinct components have disjoint writable keys; their order does not change the resulting writes. Before each solve, the current gold vectors are read from the mutable index to construct the tangent space and ownership rows. Ranking rows retain the cache-entry score gaps specified with equation 3. Thus, the QPs do not all start from an immutable shared vector state.

Each solved update is immediately renormalized and written back. If several queries share a gold, the later solve reads the earlier normalized write, then replaces that key with its own normalized result. Updates are neither averaged nor merged from independently solved snapshot copies; a later write can undo an earlier ranking gain. Other keys remain fixed during that solve, and the next cache retrieves from the resulting index.

The main method solves one QP per query. A joint QP over coupled queries is also possible, but produced weaker ranking gains in our experiments; Appendix C.4 reports this comparison. The full procedure is given in Algorithm 1. Figure 2 gives the whole of one step: the cache is retrieved against a single index state, each query becomes one instance of equation 6 over its own gold keys, and the solved displacement is renormalized and written back.

3.4 Complexity

With V and the comparison pairs fixed, equation 6 is a convex quadratic program, even if all observed queries and corpus items are included. A global solve can nevertheless be expensive: exhaustive comparisons yield up to $O(Q\gamma|T|)$ ranking rows for Q queries, at most γ golds per query, and $|T|$ corpus items.

Our per-query program instead has $R = O(\gamma(k + p))$ inequality rows, where k is the number of retrieved competitors and $p \leq B$ is the number of relevant queries in the cache. Its dual kernel requires $O(R^2)$ storage and $O(R^2)$ work per dense solver iteration. Convexity is a property of each such program and not of the update sequence: the programs are solved greedily in arrival order, a later write can move a key an earlier one set, and the renormalization that follows the solve is outside the program. Constructing the kernel and applying the update still depend on the embedding dimension d; retrieving candidates also depends on the index size and search method. Appendix B gives the derivation and detailed cost.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Encoders. The main text reports results with two frozen encoders: QWEN3-EMBEDDING-8B ($d = 4096$) (Zhang et al., 2025) and BGE-LARGE-EN-V1.5 ($d = 1024$) (Xiao et al., 2024). Neither is fine-tuned. Query and corpus embeddings are L^2-normalized before adaptation, so their initial inner products equal cosine similarities. Both encoders use the same corpora, folds, and method hyper-parameters, without encoder-specific retuning. Results with additional encoders are provided in Appendix D.4.

Benchmarks. We evaluate retrieval over three types of indexed items: skills, tools, and documents. For skill retrieval, we use SRA (Su et al., 2026); for tool retrieval, we use ToolRet's (Shi et al., 2025) three disjoint pools (customized, code, and web); and for document retrieval, we use MAIR's (Sun et al., 2024) six official domains. This yields ten retrieval pools in total. Each query is evaluated against its designated pool rather than a union of corpora, which would introduce candidates outside the corresponding task. Per-pool corpus and query counts are given in Appendix A.

Evaluation protocol. All methods and encoders use identical five-fold splits. Training queries are presented in a seed-0 shuffled order, held fixed for sequential updates. We report the mean and standard deviation across folds. No hyper-parameter is tuned on testing queries; configurations are listed in Appendix A.

Baselines. For a direct comparison, we focus on lightweight methods that adapt embedding-based retrieval. **NUDGE-M** and **NUDGE-N** (Zeighami et al., 2025) update stored corpus vectors, as does **ERM** (Hu et al., 2026); **Adaptor**, the query-side linear adapter of Sanjeev & Troynikov (2024), instead leaves the index alone and maps the query. We reproduce ERM following its original paper, run NUDGE with the official `nudge-ft` package, and use the authors' code and default settings for Adaptor. NUDGE's inner train/dev split for selecting $\eta^\star$ follows the reference harness's deterministic hash. Its updated keys are unnormalized and are scored by dot product, as required by the implementation. Query expansion (Ma et al., 2023; Wang et al., 2023; Gao et al., 2023), corpus expansion (Nogueira et al., 2019), external memory, and reranking (Nogueira et al., 2020) can be used alongside corpus-vector updates; we therefore do not treat them as direct baselines for the update itself.

Evaluation measures. We report testing N@10 as the primary retrieval metric; R@10 and C@10 appear in Appendix D.4. C@10 requires every gold target of a query to appear in the top ten.

Held-out accuracy alone does not show whether later updates undo earlier corrections. We therefore measure *first-fold gain retention*. After updating on fold 1, we evaluate its queries and record their gain g_1 in N@10 over the frozen index. After updating on folds $2, \ldots, \ell$, we reevaluate the same queries and record $g_1^{(\ell)}$. We plot $100\, g_1^{(\ell)}/g_1$: 100% means the initial gain is preserved, and values above 100% mean later updates improve it further. We omit runs whose first-fold gain is below one N@10 point; absolute changes are reported alongside retention to avoid overinterpreting ratios with small denominators.

To examine how performance changes with feedback volume, we report testing N@10 over nested subsets containing 5% to 100% of the training queries. We also count *invasions*: updated non-gold keys that move from below a query's gold target to above it in the ranking. Finally, update time measures the CPU wall-clock time required to modify the index, excluding encoding and evaluation. Further definitions and measurements are in Appendices G.1, F.1 and C.5.

4.2 BASELINE RESULTS

Retrieval performance. Table 1 reports N@10 on the ten pools for two frozen encoders. Read down a column: every method is scored against the same frozen index, and the last column averages each method's gain over that index across the ten pools. Ours is the largest in both panels, $+13.4\%$ under QWEN3-EMBEDDING-8B and $+27.1\%$ under BGE-LARGE-EN-V1.5, against $+9.6\%$ and $+20.6\%$ for the best baseline; the query-side adapter is negative in both. Per pool, our method

Table 1: **N@10 across ten pools and two frozen encoders**, with ToolRet resolved into its three disjoint pools. Panels (a) and (b) differ only in the encoder: corpora, 5-fold splits, hyper-parameters and evaluation harness are identical, and nothing was retuned on (b). Cells report fold mean ± s.d.; bold marks the best mean within a panel column. ERM is reproduced from its paper; NUDGE and Adaptor use the authors' implementations (Section 4.1). Adaptor is the query-only linear adapter of Sanjeev & Troynikov (2024); it leaves the index unchanged.

Method	SRA	ToolRet			MAIR						$\overline{\Delta}$ %
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
(a) QWEN3-EMBEDDING-8B ($d = 4096$)											
Frozen	61.3±0.4	49.8±1.6	53.2±1.5	46.6±0.7	49.5±2.1	45.6±1.0	49.3±3.0	40.0±4.6	**44.9**±3.2	**45.2**±1.0	*ref.*
ERM	81.6±0.7	59.6±2.2	51.0±2.0	55.9±0.5	55.6±2.8	42.1±2.2	46.3±2.3	30.7±3.6	41.7±2.8	41.8±0.9	+2.9
NUDGE-M	87.0±0.6	50.1±2.8	53.2±1.5	50.2±0.2	51.7±2.9	45.6±1.0	50.4±2.5	40.2±4.6	44.9±3.2	45.2±1.0	+5.8
NUDGE-N	88.2±0.6	62.4±2.5	53.2±1.5	55.0±0.6	55.8±2.8	44.9±2.4	49.0±1.8	40.0±4.6	44.9±3.2	44.2±2.2	+9.6
Adaptor	**91.9**±0.9	59.3±1.4	51.5±1.2	48.8±0.7	52.1±3.4	33.8±2.1	41.2±2.8	18.9±3.2	31.9±2.9	30.8±2.4	−8.0
RIMU *(ours)*	89.2±0.5	**66.0**±1.2	**58.7**±1.3	**56.2**±0.9	**58.0**±3.7	**49.3**±1.3	**51.1**±3.0	**40.4**±4.6	43.9±3.1	43.7±0.8	+13.4
(b) BGE-LARGE-EN-V1.5 ($d = 1024$)											
Frozen	52.5±0.6	33.6±2.3	34.2±1.9	37.4±0.8	32.5±2.6	30.1±1.6	55.5±1.7	49.0±3.1	**40.0**±1.5	48.4±2.8	*ref.*
ERM	76.6±0.5	50.7±2.1	37.7±1.7	48.1±1.3	42.4±2.9	28.7±2.1	53.3±1.7	44.7±3.5	39.3±1.6	43.7±2.5	+13.7
NUDGE-M	79.4±0.6	44.8±1.9	34.1±2.0	42.7±0.4	34.0±3.6	30.0±1.5	55.6±1.6	49.3±3.1	**40.0**±1.5	**48.4**±2.8	+10.3
NUDGE-N	82.1±0.7	54.2±2.8	38.4±0.9	49.1±0.9	45.0±2.5	32.2±3.6	55.5±1.7	48.9±2.6	**40.0**±1.5	**48.4**±2.8	+20.6
Adaptor	79.9±1.1	53.9±1.7	38.8±1.7	50.1±1.7	34.3±0.8	17.6±3.9	39.9±1.5	14.4±1.6	20.9±3.0	30.6±1.7	−6.0
RIMU *(ours)*	**82.6**±0.9	**61.2**±2.6	**46.2**±1.1	**51.7**±0.6	**47.8**±2.1	**33.3**±1.9	**56.0**±1.7	**49.7**±2.9	39.8±1.5	48.0±3.2	+27.1

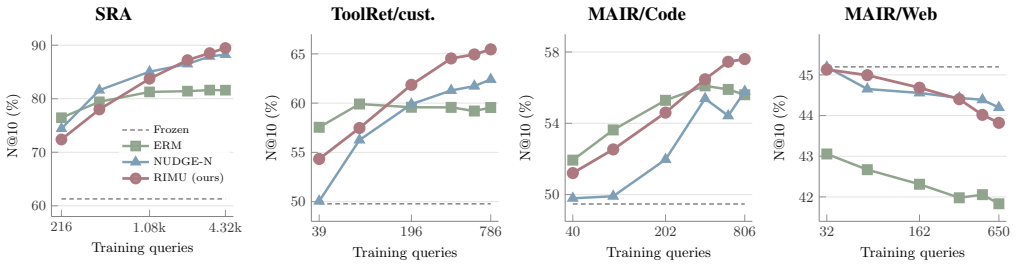


Figure 3: **Learning curves with increasing feedback.** Testing N@10 on four representative pools as nested training-query subsets grow from 5% to 100%, averaged over five folds; dashed lines mark the frozen index. Of the three methods plotted, RIMU finishes highest on the first three pools, gaining 17.1, 11.1, and 6.4 points, while all of them finish below frozen on MAIR/Web. Additional feedback can improve retrieval, but its benefit is not guaranteed; results for all ten pools appear in Figure 6.

achieves the highest mean N@10 in seven pools with Qwen and eight with BGE, and improves over the frozen index in eight pools under each encoder. The gains extend across skill, tool, and document retrieval, suggesting that the update is not specific to one encoder or corpus type. The exceptions matter: on MAIR/Academic and MAIR/Web, the frozen index remains better than all adapted methods. Some of NUDGE's cells equal the frozen index exactly, because its inner validation returns a step size too small to reorder anything.

Learning from additional queries. More feedback does not always improve testing retrieval (Figure 3). Our method keeps gaining on SRA, ToolRet/customized and MAIR/Code as the training set grows, but declines on MAIR/Academic and MAIR/Web, ending 1.0 and 1.5 points below the frozen index with Qwen. This is not particular to us — ERM ends 9.4 points below frozen on MAIR/Medical.

Two things can make added feedback unprofitable. The first is *invasion*: a written key that no testing query wanted enters its top ten and pushes a gold out. The second is that the feedback and the testing queries are simply about different things — only gold targets are writable (§3.1), so a testing query

whose golds no training query claims cannot be reached by any write, and the only effect an update can have on it is the first kind. Which regime a pool is in is measurable rather than assumed: on MAIR/Academic only 6.1% of testing queries share a gold with any training query, and a testing query there is no closer to a training query that shares its gold than to one that does not (0.707 against 0.714), whereas on SRA the figures are 96.6% and 0.889 against 0.760. Across the ten pools that overlap predicts the gain ($r = 0.93$) better than query similarity does ($r = 0.64$).

Update time. Adapting the index takes 14.6 seconds per fold on average and 7.8 at the median, on eight CPU cores with no GPU at any point (Figure 4). Two of the three index-side baselines are faster — ERM writes in 0.5 seconds and NUDGE-M in 4.2 — and the third, NUDGE-N, takes 25.7; fitting a query-side adapter is an order of magnitude above all of them. The cost is low because the dual's size follows the row count alone, not the embedding dimension or the index (§3.4). What the methods pay for differs in kind: our cost tracks the number of training queries while NUDGE's tracks the corpus, since selecting its single step size rescores the whole index once per candidate value. Appendix F.1 gives the per-pool times.

4.3 Retention of Earlier Gains

An update may improve the queries that prompted it and then lose that improvement as more feedback arrives. To measure this we treat the five folds as an arrival sequence: after adapting to the first fold we record its gain over the frozen index, then incorporate each later fold and evaluate the first fold again. Because the evaluated queries were used for adaptation, this measures retention rather than testing generalization.

Figure 1(c) shows what the displacement buys. Across the ten pools ours spends a fifth of ERM's angular budget and just under half of NUDGE-N's, and gains more than either: less movement and more retrieval at the same time, rather than a trade between them. That comparison puts each method at its own setting, so it cannot separate spending less from repairing less; swept and matched on the training N@10 each method reaches, the ordering holds and widens, and on SRA neither NUDGE variant reaches ours at any step size (Appendix F.2). Our update keeps 97.1% of what it first delivered on average and never less than 90.5% on any pool, where ERM keeps 92.4% and falls to 79.4%, and NUDGE-N — on the four pools where it trains at all — keeps 81.9% and falls to 53.8% (Figure 5; all ten pools in Figure 7 and the per-fold numbers in Appendix E). We attribute this to the minimum-displacement objective: a key moved only as far as the observed ranking requires is less likely to be undone by a later write. The attribution is consistent with the displacement each method spends, but the ablation in §4.4 varies the constraints under the ordinary testing split and does not repeat the retention measurement, so we do not claim it as a direct test.

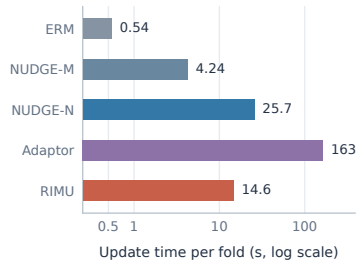


Figure 4: **What the update costs.** Mean seconds to adapt the index for one fold over the ten pools; all five methods timed back to back in one process on one CPU node (8 threads, no GPU), timing only the call that produces the new index. Note the logarithmic axis. Per-pool times in Appendix F.1.

	SRA	**ToolRet**		**MAIR**	mean	
		cust.	code	Legal	bm$^-$	full
training	$+0.28$	$+0.77$	$+1.36$	$+0.35$	$+0.69$	$+0.69$
testing	$+0.37$	$+0.37$	$+0.07$	-0.20	$+0.15$	$+0.13$
invasions	-12	-11	-12	-12	-12	-1

Table 2: **Ownership on guarded queries.** Each cell is the paired difference between RIMU and the same method with the ownership price set to zero, over 12 (pool, fold) runs. **bm$^-$** keeps only the queries the row is emitted for; **full** is every query of the same pools. The row holds the training gain ($t = +5.1$, 12 of 12 runs) and reduces invasion among the queries it guards ($t = -3.6$, 10 of 12), while testing N@10 does not move ($t = +1.4$). The two *mean* columns are what the row is for; invasions are per query, scaled by 10^3. Per-pool detail on both is in Appendix G.2.

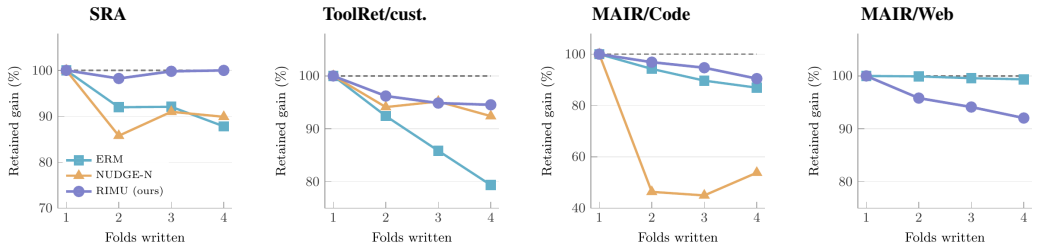


Figure 5: **What each method keeps.** After ℓ folds have been written, the first fold is scored on its *own* queries again; the axis is the share of its original gain over the frozen index that survives, so 100% is no forgetting and the scale does not depend on how much a method had to lose. RIMU ends at 100.0, 94.5, 90.5 and 92.0 on these four pools against ERM's 87.8, 79.4, 86.9 and 99.4. NUDGE-N gives back nearly half of its first-fold gain on MAIR/CODE and has nothing to retain on MAIR/WEB, where it returns the frozen index; ERM is ahead only there, sitting at the in-sample ceiling with no headroom to lose. All ten pools in Figure 7.

Table 3: **Ablation of RIMU's constraints.** Testing and training N@10 for four representative pools and the ten-pool mean, averaged over five folds with Qwen3-Embedding-8B; each variant removes one condition. Removing ownership reduces training N@10 by 0.37 points but has little measurable testing effect (-0.050 points), whereas removing tangency affects both. Ownership primarily protects observed queries in this evaluation; full results appear in Appendix D.2.

	Test					Training				
Variant	SRA	T/web	M/Code	M/Web	*mean*	SRA	T/web	M/Code	M/Web	*mean*
RIMU *(ours)*	89.2	56.2	58.0	43.7	55.64	92.9	73.8	76.2	72.8	77.31
− ownership equation 5	88.7	56.2	58.6	43.6	55.59	92.5	73.8	76.1	72.6	76.95
− tangency equation 4	88.5	53.4	57.0	44.7	55.23	91.7	59.6	67.8	62.7	67.73
ranking rows equation 3 only	88.1	53.4	57.8	44.4	55.22	90.9	59.0	66.7	62.2	66.87

4.4 ABLATION STUDY

Table 3 removes one condition at a time and reports both testing and training N@10.

Ranking constraints account for most of the gain: retaining them alone improves testing N@10 by 12.6% over the frozen index, compared with 13.4% for the full method. Removing tangency reduces training N@10 by 9.6 points and testing N@10 by 0.41 points, showing its importance when updates are renormalized before retrieval.

Ownership has a smaller, more localized effect. Removing it changes overall testing N@10 by only -0.05 points, but reduces training N@10 by 0.37 points (45 of 50 pool–fold runs). On queries selected by the ownership condition, Table 2 shows a 0.69-point training gain and 0.012 fewer invasions per testing query. Thus, ownership helps protect the queries it guards, but this benefit does not translate into a clear overall testing N@10 gain. Per-pool results appear in Appendix G.2.

5 CONCLUSION

We introduced RIMU, a corpus-aware method for continually adapting a frozen dense retriever to verified query experience. Rather than moving a gold vector solely toward its associated query, RIMU uses its ranking against competing corpus items to determine the update. A minimum-displacement convex program keeps the update local, while tangent-space optimization followed by renormalization preserves the unit-norm retrieval protocol. We also examined ownership constraints as a way to reduce retrieval intrusion, where an update for one query harms another. Across the evaluated pools, RIMU outperforms existing lightweight continual-update methods at modest adaptation cost.

AI USE STATEMENT

Generative AI tools assisted with literature search and summarization, language editing, and the development of analysis and plotting code. The authors reviewed and verified AI-assisted text, citations, and code. The research question, convex formulation, experimental design, interpretation of results, and final claims are the authors' own. The authors take full responsibility for the content of this work.

ETHICS STATEMENT

This work studies continual adaptation of dense retrieval using existing benchmark datasets and pretrained embedding models. It does not involve human-subject research or the collection of new personal data. Incorrect or adversarial feedback could redirect corpus vectors and degrade retrieval for later queries, including queries seeking tools or documents. The ownership constraints address interference among observed queries but do not guarantee correct rankings for unseen queries. Our offline evaluation does not establish robustness to malicious feedback or suitability for safety-critical deployment.

REPRODUCIBILITY STATEMENT

Section 3 specifies the ranking, ownership, and tangency constraints, the convex objective, and the normalized write-back procedure. Appendix B provides the dual derivation and the analysis of normalization, and Algorithm 1 in Appendix C states the update in full. Section 4.1 describes the frozen encoders, five-fold evaluation protocol, baselines, and metrics; Appendix A reports the retrieval pools and fixed hyperparameters. Appendices E and F.1 detail the retention and update-time evaluations. We will release the implementation, configurations, split-generation and evaluation scripts, and plotting code to facilitate reproduction. The released package is generated from the working tree rather than copied by hand, and the build fails unless the generated code reproduces the reported arm and each ablation bit for bit.

REFERENCES

Yu Bai, Yukai Miao, Dawei Wang, Li Chen, Fei Long, Rundi Zhai, Dan Li, Yanyu Ren, Tianfeng Liu, Hongtao Xie, Ce Yang, and Xuhui Cai. DMA: Online RAG alignment with human feedback. *arXiv preprint arXiv:2511.04880*, 2025. doi: 10.48550/arXiv.2511.04880. URL https://arxiv.org/abs/2511.04880.

Jihwan Bang, Seunghan Yang, Kyuhong Shim, Simyung Chang, Juntae Lee, and Sungha Choi. Feedback adaptation for retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 28472–28485. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1419. URL https://aclanthology.org/2026.findings-acl.1419/.

Thomas L. Brauen. Document vector modification. In Gerard Salton (ed.), *The SMART Retrieval System: Experiments in Automatic Document Processing*, pp. 456–484. Prentice-Hall, Englewood Cliffs, NJ, 1971.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006. URL https://www.jmlr.org/papers/v7/crammer06a.html.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pp. 1762–1777. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.99. URL https://aclanthology.org/2023.acl-long.99/.

Yuntong Hu, Sha Li, Naren Ramakrishnan, and Liang Zhao. RAG without forgetting: Continual query-infused key memory. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Z8svqD3pmI.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020. doi: 10.18653/v1/2020.emnlp-main.550.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen-tau Yih, Tim Rocktaschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*, 2023. URL https://arxiv.org/abs/2308.03281.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5303–5315. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.322. URL https://aclanthology.org/2023.emnlp-main.322/.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2202.05262.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.07229.

Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*, 2019. doi: 10.48550/arXiv.1901.04085. URL https://arxiv.org/abs/1901.04085.

Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*, 2019. doi: 10.48550/arXiv.1904.08375. URL https://arxiv.org/abs/1904.08375.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. Document ranking with a pretrained sequence-to-sequence model. In *Findings of EMNLP*, pp. 708–718, 2020. doi: 10.18653/v1/2020.findings-emnlp.63.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.16789.

J. J. Rocchio. Relevance feedback in information retrieval. In Gerard Salton (ed.), *The SMART Retrieval System: Experiments in Automatic Document Processing*, pp. 313–323. Prentice-Hall, 1971.

Suvansh Sanjeev and Anton Troynikov. Embedding adapters. Technical report, Chroma, May 2024. URL https://research.trychroma.com/embedding-adapters.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24497–24524, 2025. doi: 10.18653/v1/2025.findings-acl.1258.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill retrieval augmentation for agentic AI. *arXiv preprint arXiv:2604.24594*, 2026.

Jiashuo Sun, Xianrui Zhong, Sizhe Zhou, and Jiawei Han. DynamicRAG: Leveraging outputs of large language model as feedback for dynamic reranking in retrieval-augmented generation. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-5620. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/f69d2246a85e28c3e63dee3daf8d4af8-Abstract-Conference.html.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14918–14937. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.923. URL https://aclanthology.org/2023.emnlp-main.923/.

Weiwei Sun, Zhengliang Shi, Jiulong Wu, Lingyong Yan, Xinyu Ma, Yiding Liu, Min Cao, Dawei Yin, and Zhaochun Ren. MAIR: A massive benchmark for evaluating instructed retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14044–14067, 2024. doi: 10.18653/v1/2024.emnlp-main.778.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022. URL https://arxiv.org/abs/2212.03533.

Liang Wang, Nan Yang, and Furu Wei. Query2doc: Query expansion with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9414–9423. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.585. URL https://aclanthology.org/2023.emnlp-main.585/.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed resources for general Chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. doi: 10.1145/3626772.3657878. URL https://arxiv.org/abs/2309.07597.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629.

Jinsung Yoon, Yanfei Chen, Sercan Arik, and Tomas Pfister. Search-adaptor: Embedding customization for information retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 12230–12247. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.661. URL https://aclanthology.org/2024.acl-long.661/.

Sepanta Zeighami, Zac Wellmer, and Aditya Parameswaran. NUDGE: Lightweight non-parametric fine-tuning of embeddings for retrieval. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=MYw74B77KQ.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

A Pools and settings

Pools. SRA (26,262 targets, 1,080 testing queries per fold); ToolRet customized (3,367 / 196), code (3,794 / 349) and web (37,292 / 1,045); and the six MAIR domains (23,463 to 111,351 targets each). ToolRet is kept disjoint because each query names the one pool it belongs to and scoring against the 44,453-target union invents roughly ten times the distractors. MAIR's 52 pooled tasks carry a published task-to-domain map and each domain is scored against its own targets only, which is sound here because *no* gold crosses a domain boundary — 0 of 175,370 checked.

	SRA	ToolRet			MAIR						mean
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
keys written	298	632	776	3431	411	986	395	1322	914	2647	1181
turn θ, degrees											
mean	14.9	9.9	7.0	10.2	10.2	9.1	7.0	6.8	11.3	7.3	9.4
median	12.2	8.1	5.3	8.3	7.3	5.9	6.0	5.6	10.4	6.4	7.5
p90	31.7	20.1	15.2	21.5	24.5	21.9	13.8	13.5	21.2	14.2	19.8
p99	42.6	36.1	28.8	35.6	36.1	32.6	21.3	21.5	31.7	22.2	30.8
$\cos\theta$ kept	0.949	0.976	0.986	0.974	0.972	0.976	0.989	0.990	0.972	0.988	0.977
share $\theta < 15^\circ$	56%	78%	90%	77%	75%	75%	92%	93%	70%	91%	80%

Table 4: **How far a written key turns, and what the write keeps.** Over the keys the update actually wrote, three folds of each pool, QWEN3-EMBEDDING-8B. Because the displacement is perpendicular to the key by construction, the stored vector is exactly $(d+\Delta)\cos\theta$ for a turn of θ, so $\cos\theta$ is the fraction of the score the solver computed that survives renormalization — not a bound on it. The typical key turns 9.4° and keeps 0.977 of its solved score; 80% of keys turn less than 15°. The tail is where the cost sits: the ninetieth percentile is 19.8° and the ninety-ninth reaches 43° on SRA, the pool that also moves the most. This is the constant behind the second-order argument of §3.1.

Our settings. The released configuration uses ranking margin $m = 0.01$, ranking price $C = 10$, ownership price $C_{\text{own}} = 0.5$, $k = 10$ retrieved targets, and a cache of $B = 64$ queries. It retains at most ten golds per query; when there are more, it keeps those ranked highest by the frozen index. Ranking-row prices use a softmax over cached competitor scores with temperature 0.05. A query triggers an update according to the cache-entry rule in Algorithm 1, with a 0.05 score-gap threshold. Ownership rows compare disjoint-gold queries in the cache with cosine at least 0.5 and an owner-minus-foreign score gap at most 0.15; their preferred margin is zero. The reported solver uses tangent updates, immediate unit-norm write-back, and the seed-0 shuffled training order.

B DUAL DERIVATION AND COMPUTATIONAL DETAILS

This section supplies the derivation behind equation 7, the per-query cost omitted from the main text, and the proof and empirical check for norm-induced ranking reversals.

B.1 ELIMINATING THE TANGENCY EQUALITIES

Write each row r of equation 3 and equation 5 in the common form $\sum_t \sigma_{r,t}\, v_r^\top \Delta_t \geq \beta_r - \xi_r$, where v_r is the query (for a ranking row) or the difference of two queries (for an ownership row), and $\sigma_{r,t} \in \{+1, -1, 0\}$ records whether target t enters that row as the gold, as the competitor, or not at all. With multipliers $\lambda_r \geq 0$ on these rows and free μ_t on the tangency equalities equation 4, the Lagrangian of equation 6 is stationary in Δ_t when

$$\Delta_t \;=\; \sum_r \lambda_r\, \sigma_{r,t}\, v_r \;+\; \mu_t d_t, \tag{9}$$

and stationary in ξ_r when $\lambda_r = C_r - \nu_r$ with $\nu_r \geq 0$ the multiplier of $\xi_r \geq 0$, i.e. exactly when $0 \leq \lambda_r \leq C_r$. The slack variables therefore do not survive into the dual: they become the box.

The tangency equalities do not survive either. Substituting equation 9 into $d_t^\top \Delta_t = 0$ and using $\|d_t\|_2 = 1$ determines μ_t outright,

$$\mu_t \;=\; -\sum_r \lambda_r\, \sigma_{r,t}\, A_{rt}, \qquad A_{rt} = v_r^\top d_t, \tag{10}$$

and putting it back gives equation 7. There is no iteration over the equalities and no penalty parameter for them: their coefficients are computed while forming the dual kernel.

B.2 THE DUAL SIZE IS INDEPENDENT OF THE EMBEDDING AND CORPUS DIMENSIONS

Eliminating Δ leaves a dual in λ alone,

$$\max_{0\le\lambda_r\le C_r} \; -\tfrac{1}{2}\,\lambda^\top Q\lambda \;+\; \lambda^\top\beta, \qquad Q_{ab} = \sum_t \sigma_{a,t}\,\sigma_{b,t}\; v_a^\top P_t v_b, \tag{11}$$

with one bounded variable per row. Because P_t is an orthogonal projector, the kernel needs no $d \times d$ matrix: it is the query Gram corrected by one rank-one term per target,

$$v_a^\top P_t v_b \;=\; G_{ab} - A_{at}A_{bt}, \qquad G_{ab} = v_a^\top v_b. \tag{12}$$

Forming G and A requires inner products in dimension d, but once they are built, each dual iteration depends on the row count R rather than on d or the corpus size. Equation equation 12 gives this separation; §3.4 uses it to distinguish kernel construction from the solve. Dropping the tangency rows sets $A = 0$ and returns the kernel to the plain query Gram, so the two geometries differ by exactly this correction and by nothing else.

The tangent equality needs no iterative constraint solver: its projector appears inside the kernel. The stored vector still requires the explicit renormalization in equation 8. Since Q is a sum of positive semidefinite terms and the box is compact and non-empty, equation 11 always has a solution, with no duality gap against equation 6.

B.3 ROW COUNTS AND THE PER-CACHE COST

Fix one query: at most γ golds and k retrieved competitors, so $|\mathcal{G}| \le \gamma$ writable keys. equation 3 contributes γk rows and equation 5 at most γ per neighbouring query it is compared against, of which there are $p \le B$ above the cosine cut, so $R = O(\gamma(k+p))$; equation 4 is eliminated in closed form rather than solved. The primal has $|\mathcal{G}|d$ unknowns, but we never form it — the dual has one bounded variable per row and its kernel is the $R \times R$ kernel in equation 11, so the solve is at most $O(R^2)$ in memory and per iteration, *independent of* d *and of* $|T|$: a 367,160-target index and a 3,367-target one give the same dual.

The corpus enters through retrieval at $O(|T|d)$ per query. Per cache of B queries,

$$\underbrace{O(B\,|T|\,d)}_{\text{retrieval}} \;+\; \underbrace{O\big(B\,(R+p^2)\,d\big)}_{\text{kernels and writes}} \;+\; \underbrace{O\big(B\,\tau R^2\big)}_{\text{duals}}, \tag{13}$$

with τ the number of dual iterations. The middle term collects everything that still touches d once retrieval is done: the query Gram G over the $1+p$ queries a row can name, at $O(p^2 d)$; its tangency correction A, at $O(p\gamma d)$; and reading the dual solution back, one scaled query vector per active row, at $O(Rd)$. Since $R = O(\gamma(k+p))$ the last of these absorbs the second, and only the query Gram is left beside it.

The dual term is the dense form, which is what the implementation uses at the row counts reported here, and it is an upper bound rather than the necessary work: two rows interact only when they name a common writable key, so the kernel is structurally sparse, and past a few hundred rows the implementation assembles it as a sparse product whose cost follows the number of shared keys instead. τ is likewise a cap, with early exit once the largest coordinate move falls below tolerance.

For a fixed per-query row bound R, the total solve cost scales linearly with the number of queries; R itself can increase with the number p of ownership neighbours in the cache. One QP per query does not make the update sequence independent: two queries are coupled when they share a writable gold, and also when one query's gold sits in the other's retrieved list, since the second query's rows are then stated against a key the first one may have moved. Algorithm 1 therefore uses the fixed component traversal and within-component arrival order specified in Section 3.3; arbitrary reordering or parallel solves need not produce the same index.

Retrieval dominates whenever $|T|$ is large, which is the regime the method is for. Over the ten pools of Table 12 the measured update time correlates at 0.97 with the product of index size and query count — what the retrieval term alone predicts — against 0.29 with the index size and 0.80 with the query count taken separately.

Table 5: Effect of geometric and metric normalization. Values are N@10 multiplied by 100. **RIMU (cosine)** uses tangent-space updates followed by normalization and cosine retrieval. **RIMU-dot** removes the tangency constraint and uses the dot-product formulation. **RIMU-tan-dot** retains tangency but omits post-update normalization. The final row reports the mean difference from RIMU (cosine).

	SRA	**ToolRet**			**MAIR**					$\overline{\Delta}$ vs. RIMU
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
Frozen	61.29	49.78	53.16	46.57	49.47	45.60	49.26	40.02	44.90	45.20	–
RIMU *(ours)*, cosine	89.17	66.05	58.68	56.16	57.96	49.29	51.12	40.42	43.86	43.68	–
RIMU-dot	88.14	66.92	51.03	33.30	58.09	39.25	49.36	39.12	36.41	42.59	−5.22
RIMU-tan-dot	90.36	14.46	57.42	5.04	59.29	48.94	51.15	40.32	42.47	43.34	−10.36

B.4 Norm growth and post-update normalization

Proof of Proposition 1. Orthogonality gives $\|z^{(T)}\|^2 = 1 + S_T$. Thus the irrelevant key has inner-product score $q^\top z^{(T)} = c_T\sqrt{1+S_T}$, which exceeds $q^\top h$ under the stated condition. Yet its cosine similarity c_T remains lower than that of h. □

Normalization is not merely an implementation detail in vector adaptation. For adapted query and document embeddings, raw dot-product scoring satisfies

$$\tilde{q}^\top \tilde{d} = \|\tilde{q}\|_2 \, \|\tilde{d}\|_2 \, \cos(\tilde{q}, \tilde{d}).$$

Without norm control, ranking therefore depends not only on semantic direction but also on vector magnitude. An adapter may improve its optimization objective by increasing selected vector norms rather than learning better angular alignment. Such magnitude effects can accidentally correlate with relevance on one benchmark while severely distorting rankings on another.

Constraining an update to the tangent space does not by itself preserve normalization. For a unit vector d and a tangent update satisfying $d^\top \Delta = 0$,

$$\|d + \Delta\|_2^2 = \|d\|_2^2 + 2d^\top \Delta + \|\Delta\|_2^2 = 1 + \|\Delta\|_2^2.$$

Thus, every nonzero finite tangent step increases the norm unless it is followed by an explicit retraction or renormalization,

$$d' = \frac{d + \Delta}{\|d + \Delta\|_2}.$$

Tangency controls the first-order direction of movement; it does not replace post-update normalization.

Table 5 reveals two distinct failure patterns. Removing both tangency and cosine normalization produces a broad degradation: RIMU-dot underperforms normalized RIMU on eight of ten pools, with an average loss of 5.22 points and a 22.86-point loss on ToolRet/web. Retaining tangency without renormalization is substantially less predictable. RIMU-tan-dot slightly exceeds normalized RIMU on SRA and MAIR/Code, but collapses to 14.46 on ToolRet/customized and 5.04 on ToolRet/web, respectively 35.32 and 41.53 points below the frozen retriever. Its mean loss of 10.36 points is driven almost entirely by these catastrophic failures.

This behavior was reproduced in a separate three-arm rank-gap diagnostic. The unnormalized tangent variant (`tangent_noren`) reached 94.46 on SRA but only 20.38 on ToolRet/customized. The same benchmark-dependent instability appearing in two independent evaluations indicates that it is not an isolated optimization accident.

These results do not imply that every unnormalized adapter must fail on every dataset: indeed, the unnormalized variants can outperform the normalized method on selected pools. Rather, they show that performance without normalization is unreliable across domains and can conceal severe worst-case failures behind competitive results on favorable benchmarks. The mechanism is not specific to RIMU; any adapter that changes embedding norms while using raw dot-product scores exposes retrieval to the same uncontrolled magnitude effect. We therefore treat output normalization, post-update retraction, and consistency between the optimized and deployed similarity metrics as design

Algorithm 1 RIMU: one pass over a stream of verified retrievals

Require: initial unit-norm keys D; stream of queries with verified gold sets; cache size B; base ranking price C for equation 3 and $C_{\text{own}} = 0.05\,C$ for equation 5
Ensure: updated unit-norm keys K, differing from D only on written golds

$K \leftarrow D$
for each cache Q_b of at most B queries, in arrival order **do**
  Cache $s_b(q,t) \leftarrow q^\top K_t$, top-$k$ sets T_q, and $\mathcal{N}_q \leftarrow T_q \setminus \mathcal{G}_q$ for all $q \in Q_b$
  Select triggered queries using these cached scores ▷ criterion below
  Link triggered queries sharing a gold or a gold–competitor key; form components
  for each component, ordered by its first query's arrival **do**
    for each query q in the component, in arrival order **do**
      Read current gold vectors $d_g^0 \leftarrow K_g$, $g \in \mathcal{G}_q$
      Form ranking rows from cached gaps $s_b(q,g) - s_b(q,n)$; form tangency rows at d_g^0
      Form ownership rows using d_g^0 and eligible triggered queries in the cache
      Solve the per-query dual over $\mathcal{G}_q$ alone
      for $g \in \mathcal{G}_q$ **do**
        $K_g \leftarrow (d_g^0 + \Delta_g)/\|d_g^0 + \Delta_g\|_2$ ▷ immediate write; visible to later solves
      end for
    end for
  end for
end for
return K

choices that preserve the intended cosine metric and avoid this source of instability; the proposition does not establish that they are necessary for good retrieval on every dataset.

B.5 Prices

The box caps are the only per-row parameters. Ranking rows carry $C_r = Cw_{gn}$, using the cached-score weights in equation 6, and ownership rows their own cap $C_{\text{own}} = 0.05\,C$, the value used in every run reported here; a row that is already satisfied leaves $\lambda_r = 0$ and contributes nothing to equation 7, which is why the set of targets that actually move is a subset of the writable set rather than all of it.

C Update procedure and its design choices

The trigger uses cache-entry scores. A query with at least one non-gold competitor is selected if its highest-scoring gold is not ranked first, any gold lies outside the top k, or the highest-scoring gold exceeds the highest-scoring non-gold competitor by less than 0.05. Shared-key writes are serialized in the order above. Ownership comparisons in the release implementation use other triggered queries with disjoint gold sets, query cosine at least 0.5, and current owner-minus-foreign score at most 0.15; their preferred margin is zero.

The remaining comparisons isolate choices in this procedure. They change how rows are relaxed or when the index is refreshed while keeping the reported per-query solver as the reference point.

C.1 Relaxing by penalty rather than by dropping rows

We implemented the outer loop *solve; if infeasible, drop the binding row; re-solve* and it is worse than the soft form: dropping the strongest unmet row and retrying costs -0.87 to -3.61 N@10 across the benchmarks. The loop commits to a discrete choice — this row, not that one — on the basis of the current iterate, whereas the penalty lets every row give a little and lets the quadratic term decide how much.

C.2 THE CACHE: WHAT REFRESHING PER COMPONENT ACTUALLY CHANGES

As a diagnostic, we refresh retrieval after each coupling component instead of once at cache entry. This variant is separate from the reported method; it changes later components' candidates and cached score gaps, not the per-query QP itself.

Components are small — 1.6 queries each on SRA, 1.4 on MAIR/Code — so one fold of SRA re-reads the index $2{,}683$ times where a single-snapshot cache would read it 68 times. Of those refreshes, 99.0% return the *identical* competitor set (94.3% on MAIR/Code, the one pool where the protocol gains anything at all, $+0.19$); when a set does change, the Jaccard overlap is still 0.79. Across twenty cells the two forms differ by -0.19 to $+0.19$, which is fold noise.

The choice only becomes visible when B grows. Materialising a whole $1{,}399$-query stream against one snapshot moves $1{,}698$ targets into some query's top-10 that no constraint had ever mentioned, and the competitor sets it solved for retain a Jaccard overlap of only 0.757 with the sets that actually obtain. Dropping the cache entirely costs up to 8.3 N@10 on SRA and 4.3 on MAIR/Code — and on three of the ten pools it does not run at all, because the components of an uncapped stream merge into one block whose dual reaches $948{,}246$ rows and 3.3 TiB.

C.3 WHAT THE UPDATE COSTS BYSTANDERS

Rank is zero-sum: lifting one target past another lowers the other. Attributing every testing query that loses ground over a cache, 97.8% are cases where the query's *own* gold never moved and a different query's written gold rose past it; only 1.7% are targets shared between a training and a testing query that moved against the latter. Invaders are separated from bystanders by alignment, not magnitude: $\cos(q_{\text{test}}, \Delta_g)$ is 0.27 for invaders against 0.13 for written keys that did not invade, and $\cos(q_{\text{test}}, q_{\text{train}})$ is 0.65 against 0.39. The damage is generalisation spilling onto near-duplicate queries, not a degradation of the index: across 775 written keys the mean pairwise cosine rises only from 0.436 to 0.460, and each key travels 23–32% of the angular distance to its own query — $6.5°$ to $13.9°$ — rather than collapsing onto it.

C.4 JOINTLY SOLVING A COUPLING COMPONENT

The reported method visits coupling components but solves one QP per query. As a controlled alternative, we solve all queries in a component jointly. The graph links two queries in a cache when

- **they share a gold:** $\mathcal{G}_i \cap \mathcal{G}_j \neq \emptyset$, so both may write the same corpus vector; or
- **one's gold is the other's competitor:** $(\mathcal{G}_i \cap \mathcal{N}_j) \cup (\mathcal{G}_j \cap \mathcal{N}_i) \neq \emptyset$, so a vector written for one is ranked as a non-gold by the other.

Take the connected components of that graph and solve one program per component. The two conditions give the joint program two powers the per-query solve does not have: a shared key's displacement is priced against every query that claims it at once instead of being overwritten by whichever query is served last, and a rival's gold, being a variable rather than a constant, can be pushed *down* as well as out-run.

Neither power pays for itself here. Adding the grouping back costs 0.07 testing N@10 over the fifty (pool, fold) cells and 0.21 on the training queries, both against it; scoring the earlier folds of an arrival stream again after later folds are written — the measurement that would favour it, since overwriting is exactly what it prevents — it is 0.38 *worse*, with the per-query solve ahead on seven of the ten pools. What it does buy is a small reduction in invasions, concentrated where one would expect: testing queries are bucketed by their similarity to the nearest training query that shares none of their golds, and removing the grouping adds 0.022 invasions per query in the least similar bucket against 0.123 in the most. That is the mechanism working as designed, on a quantity that turns out to cost the retrieval almost nothing — fewer than one invasion in twenty displaces a gold from a top ten.

The grouping is also the more complicated object. A component of n queries has $R = O(n\gamma k + n^2\gamma)$ rows against $O(\gamma(k+p))$ for a single query, its kernel is quadratic in the component size, and its members cannot be solved independently of one another. That said, the two forms cost about the same in practice at this scale: timed on one node, the per-query solve is faster on five of the nine

	SRA	ToolRet			MAIR						mean
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
N@10	89.06	65.99	58.64	56.30	58.02	49.54	51.12	40.30	43.85	43.79	
s.d. order	0.354	0.432	0.253	0.162	0.613	0.326	0.092	0.256	0.118	0.134	0.274
s.d. fold	0.426	1.460	1.370	0.966	3.468	1.299	3.069	4.419	3.079	0.832	2.039
ratio	1.2	3.4	5.4	6.0	5.7	4.0	33.4	17.2	26.2	6.2	7.4

Table 6: **The arrival order is not what the result rests on.** Five orders of the same feedback — the published one and seeds 1 to 4 — over ten pools and five folds, QWEN3-EMBEDDING-8B, with everything but the order held fixed. **s.d. order** is the spread over the five orders within a fold; **s.d. fold** is the spread over the five folds within an order, which is the quantity the paper already reports after every cell. Order moves the ten-pool mean between 55.55 and 55.72, a standard deviation of 0.069, against a gain over the frozen index of 13.4. Averaged over pools it is 7.4 times smaller than the fold-to-fold spread a reader is already asked to accept. NUDGE-N was run at every seed as a control: it does not consume arrival order, and it is identical to the digit on all 20 of its comparable cells, so the spread above is the order rather than the harness.

pools and slower on four, for a mean difference of 0.7 seconds per fold, because solving each query alone pays a fixed setup per query that a component otherwise amortises. We report the per-query solve as the method for its simplicity and its behaviour, not for its speed, and keep this variant here.

C.5 SENSITIVITY TO ARRIVAL ORDER

The per-query writes make the feedback order part of the method. We compare the reported seed-0 order with four additional shuffles over the same ten pools and five folds, holding every other setting fixed.

Two things make the table below mean what it says. The seed reaches nothing in the solve path except the shuffle — the solver classes store it and never draw from it — so changing it changes the arrival order and nothing else. And NUDGE-N, which does not consume arrival order at all, was run at every seed as a control: it is identical to the digit on every comparable cell, so the spread reported here is the order and not the harness.

D ADDITIONAL RETRIEVAL RESULTS

This section expands the main N@10 table in four directions: pool interpretation, the full constraint ablation, learning curves, and encoder and metric robustness. These are the same runs and splits described in Section 4.1; no additional tuning is introduced here.

D.1 READING THE MAIN RESULT TABLE

The three ToolRet columns are not a decomposition. Each restricts the corpus to one pool and every query is scored only against the pool it names, so these are three separate retrieval problems, each easier than the union. Differences must be read down a column, never across columns and never against an aggregate-ToolRet number: an earlier version of this table scored *web* against a 36,373-target pool and its frozen row read 51.0, where the full pool reads 46.6. Nothing here mixes the two.

NUDGE's step size collapses on one cell, and only there. NUDGE writes $d_t \leftarrow d_t - \eta^\star g_t$ with unit g_t and a *single* scalar step size $\eta^\star$ — written γ in their paper, renamed here because γ bounds the golds per query above — chosen on an inner dev split, so the whole method reduces to that one number. On ToolRet/code under QWEN3-EMBEDDING-8B both variants select $\eta^\star = 4.2 \times 10^{-7}$ — seven orders of magnitude below the 3.73 the same code selects on SRA — and the five folds reproduce the frozen row to every digit we compute ($+0.0000$ each). The keys are written; the step is simply too small to reorder anything.

Table 7: **Ablation, all ten pools.** The four pools of Table 3 plus the six omitted there, same runs and same folds. *mean* is over all ten and $\overline{\Delta}\%$ is the mean per-pool gain over the frozen index. The frozen row is the same to one decimal in both panels because it is the same index and the two splits are random partitions of one query set; the underlying means differ by at most 0.007.

Variant	SRA	ToolRet			MAIR						*mean*	$\overline{\Delta}\%$
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web		
(a) testing-set N@10												
Frozen index	61.3	49.8	53.2	46.6	49.5	45.6	49.3	40.0	44.9	45.2	48.52	*ref.*
RIMU *(ours)*	89.2	66.0	58.7	56.2	58.0	49.3	51.1	40.4	43.9	43.7	55.64	+13.4
− ownership equation 5	88.7	65.7	58.4	56.2	58.6	49.5	51.1	40.2	43.9	43.6	55.59	+13.3
− tangency equation 4	88.5	65.3	58.5	53.4	57.0	49.0	50.7	40.5	44.6	44.7	55.23	+12.6
ranking rows equation 3 only	88.1	64.8	58.4	53.4	57.8	49.1	50.7	40.6	44.7	44.4	55.22	+12.6
(b) training-set N@10												
Frozen index	61.3	49.8	53.2	46.6	49.5	45.6	49.3	40.0	44.9	45.2	48.52	*ref.*
RIMU *(ours)*	92.9	79.8	78.2	73.8	76.2	74.6	77.1	75.7	72.0	72.8	77.31	+60.2
− ownership equation 5	92.5	79.3	76.7	73.8	76.1	74.4	76.8	75.5	71.8	72.6	76.95	+59.5
− tangency equation 4	91.7	74.4	69.2	59.6	67.8	62.6	67.4	61.6	60.4	62.7	67.73	+39.5
ranking rows equation 3 only	90.9	73.0	66.3	59.0	66.7	62.0	67.0	61.4	60.2	62.2	66.87	+37.8

The pooled MAIR run — the six domains as a single 367,160-key index, which is how earlier versions of this table reported them — fails a different way and should not be read as the same one. There $\eta^\star$ is not degenerate (2.9×10^{-2} for NUDGE-M) and 213,882 keys move; the per-fold deltas are $-0.047, +0.013, +0.003, +0.073, -0.060$, which average to a cell that prints as unchanged. That is an update with no effect, not a refusal to update. Panel (b) shows the collapse is not a property of the benchmark: under BGE-LARGE-EN-V1.5 the same ToolRet/code pool draws a live step from NUDGE-M, which then moves the score slightly the wrong way (34.1 against a frozen 34.2). The degenerate $\eta^\star$ belongs to the encoder–pool pair.

Splitting MAIR is what makes its result readable. Pooled into one column, MAIR reads 44.7 frozen against 47.0 for RIMU — a +2.3 that says nothing about where it came from. Split, the six domains do not behave alike at all. Every method gains on *Code*, RIMU by +8.5 under QWEN3-EMBEDDING-8B and +15.2 under BGE-LARGE-EN-V1.5. On *Legal* RIMU gains +3.7 and is the only method that gains at all under QWEN, though NUDGE-N also gains there under BGE (+2.1). On *Academic* nothing moves under either encoder — frozen, NUDGE-M and NUDGE-N print the same cell to the decimal — and on *Web* the same holds under BGE while NUDGE-N gives up 1.0 under QWEN. ERM and RIMU come out below frozen on both domains. The pooled number was two opposite behaviours averaging into a small positive.

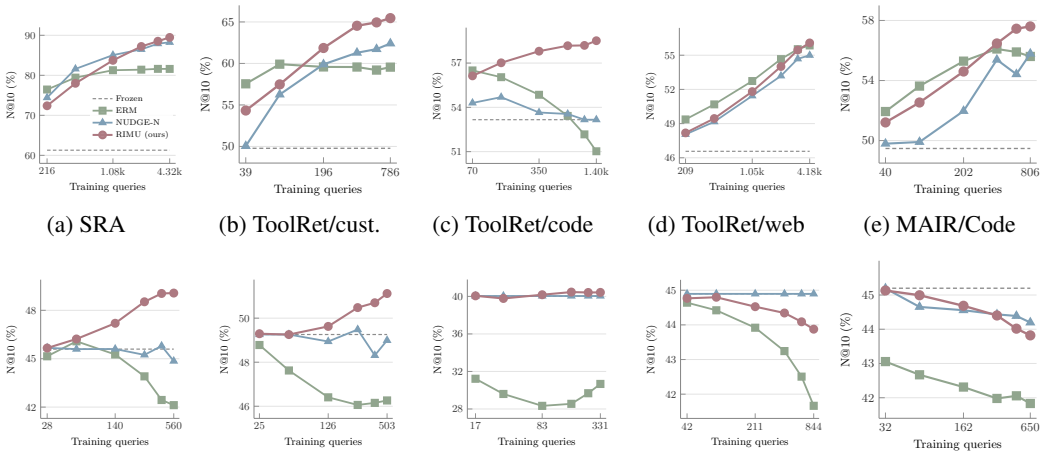
ERM can lose to the frozen index. It is below frozen on five of the six MAIR domains, and on *Medical* it reads 30.67 against a frozen 40.02 — a 9.35-point loss that the pooled column had hidden as a 1.5-point one. A target-side memory that moves keys toward a weighted query direction has no mechanism to stop when the move is not needed; RIMU's minimum-norm objective is exactly that mechanism, and it is why RIMU loses at most 1.5 where ERM loses 9.4.

D.2 CONSTRAINT ABLATION ACROSS ALL TEN POOLS

Table 3 displays four representative pools. The complete ten-pool breakdown in Table 7 uses the same runs and folds, so it tests whether the mean hides a large pool-specific reversal.

D.3 LEARNING CURVES ACROSS ALL TEN POOLS

Figure 6 extends the four curves in Figure 3 to every pool. It separates gains that continue with more feedback from the MAIR/Academic and MAIR/Web cases where RIMU ends below the frozen index.



(a) SRA (b) ToolRet/cust. (c) ToolRet/code (d) ToolRet/web (e) MAIR/Code

(f) MAIR/Legal (g) MAIR/Fin. (h) MAIR/Med. (i) MAIR/Acad. (j) MAIR/Web

Figure 6: **Learning curves, all ten pools.** The four in Figure 3 plus the six omitted there. Added feedback is not uniformly an improvement and the failure is not evenly distributed: ERM ends below where it started on five pools — on ToolRet/code it crosses from above the frozen index to below it, losing 5.5 — while ours declines on two, MAIR/Academic (-0.9) and MAIR/Web (-1.3), the same two pools where no method beats the frozen index in Table 1. NUDGE-N is flat to the digit on MAIR/Academic and MAIR/Medical at every split, its global step size having collapsed.

D.4 Five frozen encoders and three retrieval metrics

Table 1 reports two encoders. To test whether its ordering depends on those backbones, we run the same grid on three more: E5-LARGE-V2 (Wang et al., 2022), GTE-LARGE (Li et al., 2023) and QWEN3-EMBEDDING-0.6B (Zhang et al., 2025). Corpora, folds, hyper-parameters and evaluation are identical throughout and nothing was retuned per encoder. The baselines were run through the same container and virtual environment as the published rows, so a difference between panels is a difference between encoders and not between software stacks.

Tables 8, 9, and 10 report N@10, per-gold R@10, and all-golds-per-query C@10, respectively. Reading all three avoids treating a gain in average rank as proof that every gold for a query reaches the top ten. The same pool can also respond differently to the same update under different frozen encoders. Each cell stacks the fold mean above its standard deviation.

Table 8: **N@10 across five frozen encoders**, same ten pools, same 5-fold splits, same hyperparameters — nothing was retuned per encoder. The two panels of Table 1 are the third and fifth panels here, so all five can be read together. The final column is each method's mean gain over that encoder's own frozen index, which is the comparison the table is for: a method that only works on one backbone shows up as a column that changes sign between panels.

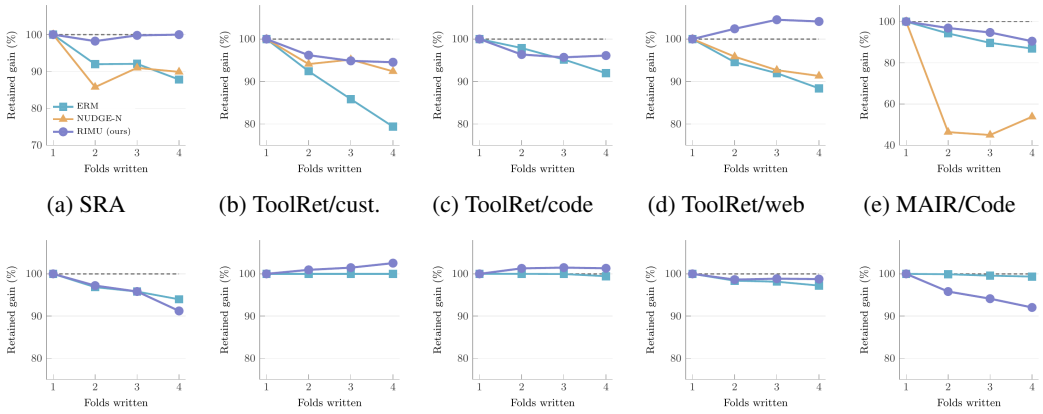
Method	SRA	ToolRet			MAIR						$\overline{\Delta}$ %
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
E5-LARGE-V2 ($d = 1024$)											
Frozen	47.2±0.5	37.6±2.0	31.2±1.2	36.2±1.5	32.5±3.4	29.6±1.5	59.7±2.7	44.8±2.0	**46.3**±1.8	43.6±2.2	*ref.*
ERM	76.1±0.8	49.9±2.2	27.7±0.9	39.6±1.2	40.6±1.5	21.5±2.3	49.9±1.1	25.0±3.3	34.1±1.0	34.8±1.9	−1.8
NUDGE-M	79.4±0.8	48.0±2.1	27.8±1.8	40.0±1.3	34.5±2.5	29.6±1.5	59.9±2.8	44.8±2.0	46.3±1.8	**43.6**±2.2	+10.2
NUDGE-N	80.6±0.7	50.7±2.4	30.8±1.6	42.6±1.5	45.3±1.4	29.6±1.5	59.2±2.3	44.8±2.0	46.3±1.8	**43.6**±2.2	+16.1
Adaptor	**80.8**±1.0	54.0±2.4	37.8±1.7	**46.6**±2.9	33.8±1.4	15.1±1.1	45.2±1.9	7.6±1.1	25.0±3.1	25.5±3.0	−7.6
RIMU *(ours)*	80.2±1.0	**58.6**±3.7	**40.2**±0.8	42.4±1.5	**48.4**±2.6	**31.7**±2.5	**60.2**±2.7	**44.8**±2.6	42.9±1.5	42.7±2.3	+21.9
GTE-LARGE ($d = 1024$)											
Frozen	53.8±0.7	36.5±2.4	35.1±1.6	34.1±0.8	25.4±3.0	30.3±1.7	46.3±1.3	48.2±3.1	**41.5**±1.3	**46.5**±1.8	*ref.*
ERM	75.2±0.6	50.0±2.6	34.6±2.1	40.2±1.1	38.9±1.5	28.9±2.1	39.4±2.6	38.9±2.7	40.4±1.1	42.2±1.4	+9.6
NUDGE-M	78.0±1.0	44.1±2.5	32.0±3.8	37.6±1.3	29.9±2.9	30.2±1.6	**46.9**±1.7	**49.0**±2.8	**41.5**±1.3	46.1±2.2	+8.7
NUDGE-N	81.0±1.0	51.9±2.8	36.7±1.7	42.1±1.1	40.1±2.0	32.8±3.5	46.3±1.3	48.3±3.2	**41.5**±1.3	46.5±1.8	+18.7
Adaptor	79.4±1.2	55.3±2.7	39.5±2.2	45.0±2.6	32.3±2.8	20.3±2.8	42.3±2.5	12.7±2.2	24.1±3.4	31.9±2.9	−1.8
RIMU *(ours)*	**81.9**±0.9	**61.0**±2.5	**45.4**±0.6	**46.9**±1.1	**43.9**±3.6	**33.5**±1.7	46.4±2.3	48.6±3.1	40.9±1.3	45.7±2.0	+26.7
BGE-LARGE-EN-V1.5 ($d = 1024$)											
Frozen	52.5±0.6	33.6±2.3	34.2±1.9	37.4±0.8	32.5±2.6	30.1±1.6	55.5±1.7	49.0±3.1	**40.0**±1.5	48.4±2.8	*ref.*
ERM	76.6±0.5	50.7±2.1	37.7±1.7	48.1±1.3	42.4±2.9	28.7±2.1	53.3±1.7	44.7±3.5	39.3±1.6	43.7±2.5	+13.7
NUDGE-M	79.4±0.6	44.8±1.9	34.1±2.0	42.7±0.4	34.0±3.6	30.0±1.5	55.6±1.6	49.3±3.1	**40.0**±1.5	**48.4**±2.8	+10.3
NUDGE-N	82.1±0.7	54.2±2.8	38.4±0.9	49.1±0.9	45.0±2.5	32.2±3.6	55.5±1.7	48.9±2.6	**40.0**±1.5	**48.4**±2.8	+20.6
Adaptor	79.9±1.1	53.9±1.7	38.8±1.7	50.1±1.7	34.3±0.8	17.6±3.9	39.9±1.5	14.4±1.6	20.9±3.0	30.6±1.7	−6.0
RIMU *(ours)*	**82.6**±0.9	**61.2**±2.6	**46.2**±1.1	**51.7**±0.6	**47.8**±2.1	**33.3**±1.9	**56.0**±1.7	**49.7**±2.9	39.8±1.5	48.0±3.2	+27.1
QWEN3-EMBEDDING-0.6B ($d = 1024$)											
Frozen	43.4±0.9	45.3±0.7	37.8±1.8	42.5±1.0	47.8±1.7	25.4±2.1	25.8±3.7	18.4±3.1	**32.2**±2.0	18.3±3.1	*ref.*
ERM	75.1±0.2	53.5±1.6	37.7±1.2	53.1±1.1	53.1±2.5	19.1±1.9	22.7±2.4	17.8±3.1	27.7±1.7	14.8±2.5	+5.3
NUDGE-M	85.0±0.5	45.5±0.4	38.0±1.8	46.2±0.8	50.1±2.3	25.4±2.1	27.7±3.6	18.4±3.1	31.8±2.1	18.2±3.0	+11.6
NUDGE-N	84.7±0.7	56.1±2.4	41.5±1.9	53.1±0.8	54.4±2.8	25.4±2.1	28.7±3.7	16.0±3.3	**32.2**±2.0	18.3±3.1	+16.5
Adaptor	**90.9**±0.5	61.4±2.0	**46.8**±1.9	47.5±1.6	47.3±1.6	24.6±2.0	**31.1**±1.8	8.8±0.4	26.2±2.1	**27.4**±0.9	+17.6
RIMU *(ours)*	86.8±0.4	**62.0**±0.7	46.7±1.6	**54.2**±1.4	**56.0**±3.3	**27.7**±2.9	28.9±3.7	**19.8**±3.7	31.1±2.1	17.5±2.9	+22.6
QWEN3-EMBEDDING-8B ($d = 4096$)											
Frozen	61.3±0.4	49.8±1.6	53.2±1.5	46.6±0.7	49.5±2.1	45.6±1.0	49.3±3.0	40.0±4.6	**44.9**±3.2	**45.2**±1.0	*ref.*
ERM	81.6±0.7	59.6±2.2	51.0±2.0	55.9±0.5	55.6±2.8	42.1±2.2	46.3±2.3	30.7±3.6	41.7±2.8	41.8±0.9	+2.9
NUDGE-M	87.0±0.6	50.1±2.8	53.2±1.5	50.2±0.2	51.7±2.9	45.6±1.0	50.4±2.5	40.2±4.6	44.9±3.2	45.2±1.0	+5.8
NUDGE-N	88.2±0.6	62.4±2.5	53.2±1.5	55.0±0.6	55.8±2.8	44.9±2.4	49.0±1.8	40.0±4.6	44.9±3.2	44.2±2.2	+9.6
Adaptor	**91.9**±0.9	59.3±1.4	51.5±1.2	48.8±0.7	52.1±3.4	33.8±2.1	41.2±2.8	18.9±3.2	31.9±2.9	30.8±2.4	−8.0
RIMU *(ours)*	89.2±0.5	**66.0**±1.2	**58.7**±1.3	**56.2**±0.9	**58.0**±3.7	**49.3**±1.3	**51.1**±3.0	**40.4**±4.6	43.9±3.1	43.7±0.8	+13.4

Table 9: **R@10 across five frozen encoders**, same ten pools, same 5-fold splits, same hyperparameters — nothing was retuned per encoder. The third and fifth panels are the two encoders of Table 1, read at R@10 instead of N@10. The final column is each method's mean gain over that encoder's own frozen index, which is the comparison the table is for: a method that only works on one backbone shows up as a column that changes sign between panels.

Method	SRA	ToolRet			MAIR						$\overline{\Delta}$ %
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
E5-LARGE-V2 ($d = 1024$)											
Frozen	58.7±1.1	45.3±1.8	40.9±0.9	46.3±1.6	43.5±3.9	31.5±2.3	**73.5**±3.2	**44.9**±0.5	**62.4**±2.3	**49.7**±1.1	*ref.*
ERM	89.9±1.0	58.2±2.0	32.0±1.4	47.4±1.6	53.2±2.6	31.2±2.5	63.3±2.3	23.8±3.6	48.7±1.5	41.2±2.0	−1.6
NUDGE-M	89.1±0.7	55.7±2.3	31.9±4.8	49.9±1.4	45.1±5.4	31.5±2.3	73.4±3.0	**44.9**±0.5	**62.4**±2.3	49.7±1.1	+6.4
NUDGE-N	90.8±0.7	59.9±2.2	38.0±3.8	52.5±1.7	60.0±3.3	31.5±2.3	72.9±2.7	**44.9**±0.5	**62.4**±2.3	49.7±1.1	+13.1
Adaptor	90.4±1.0	61.9±2.0	**51.9**±2.8	**57.2**±2.7	46.2±0.9	19.7±1.8	60.5±3.8	8.0±1.5	36.7±5.6	28.1±2.7	−7.5
RIMU *(ours)*	**91.5**±1.0	**65.7**±3.8	49.7±0.9	54.8±1.5	**63.1**±5.0	**35.9**±2.8	73.5±3.6	44.8±1.9	57.4±1.5	49.2±1.1	+19.0
GTE-LARGE ($d = 1024$)											
Frozen	67.4±0.9	45.0±3.0	46.9±2.8	45.2±0.8	34.8±4.3	31.6±2.1	58.0±2.3	47.9±3.0	**56.9**±2.5	**52.4**±2.1	*ref.*
ERM	89.6±0.5	62.2±1.8	43.8±3.4	49.8±1.7	50.4±2.5	34.5±2.5	47.6±2.3	39.4±3.4	56.0±2.2	49.3±2.6	+8.5
NUDGE-M	88.7±0.9	52.5±2.2	41.3±9.3	49.1±1.2	38.8±5.0	31.6±2.1	**58.4**±2.4	**48.2**±3.0	**56.9**±2.5	52.2±2.1	+5.7
NUDGE-N	92.3±0.7	63.5±2.4	47.4±1.8	53.2±1.2	52.7±3.5	37.9±4.8	58.0±2.3	47.9±3.0	**56.9**±2.5	52.4±2.1	+16.8
Adaptor	90.0±0.7	62.7±3.1	52.7±3.6	56.0±1.9	44.9±3.7	26.6±3.0	53.6±4.0	13.1±3.2	36.3±5.2	33.6±2.4	−3.0
RIMU *(ours)*	**92.3**±0.8	**71.9**±2.1	**57.5**±1.3	**59.4**±0.9	**56.9**±5.6	**38.5**±2.1	57.7±2.6	48.0±3.5	56.2±2.6	52.1±2.5	+23.4
BGE-LARGE-EN-V1.5 ($d = 1024$)											
Frozen	65.9±1.0	41.7±3.0	45.7±2.7	48.3±1.1	42.8±4.0	31.3±2.1	68.4±3.1	48.2±1.9	54.3±3.5	**53.3**±2.0	*ref.*
ERM	89.7±0.5	63.7±1.4	47.9±1.4	58.9±1.4	56.5±4.4	33.6±2.4	67.8±2.9	45.0±3.2	54.4±3.6	49.6±3.1	+14.0
NUDGE-M	89.1±0.6	53.5±2.0	45.7±2.7	53.9±0.6	43.6±7.7	31.3±2.0	68.7±2.6	48.3±1.8	54.3±3.5	**53.3**±2.0	+7.7
NUDGE-N	92.7±0.8	66.2±2.4	48.3±1.5	59.7±0.9	58.0±4.2	36.4±3.8	68.4±3.1	47.7±2.1	54.3±3.5	**53.3**±2.0	+17.9
Adaptor	89.7±1.2	62.4±2.3	52.3±2.7	60.4±1.2	45.6±1.4	22.9±6.0	53.6±3.3	20.1±2.7	31.0±3.7	33.3±1.1	−5.6
RIMU *(ours)*	**92.8**±0.8	**72.5**±2.5	**57.7**±1.2	**63.8**±0.8	**60.6**±4.3	**38.0**±2.3	**69.1**±2.8	**48.5**±2.2	**54.5**±3.6	52.8±2.3	+23.7
QWEN3-EMBEDDING-0.6B ($d = 1024$)											
Frozen	50.0±1.1	55.5±1.9	51.1±1.6	53.9±1.0	60.8±3.0	30.1±2.1	34.8±3.0	**19.0**±5.5	**47.3**±2.4	21.8±2.5	*ref.*
ERM	86.8±0.4	66.5±2.0	49.5±1.9	64.7±1.3	67.7±4.1	27.7±2.4	28.3±2.6	15.0±3.6	41.6±2.0	16.1±2.3	+3.6
NUDGE-M	93.2±0.3	55.7±1.7	52.2±2.5	58.2±1.5	63.2±4.1	30.1±2.1	37.2±2.6	**19.0**±5.5	46.7±1.9	21.7±2.4	+10.6
NUDGE-N	93.9±0.7	67.1±1.5	56.0±1.8	64.7±1.1	69.0±5.0	30.1±2.1	38.3±3.2	14.1±3.5	**47.3**±2.4	21.8±2.5	+13.6
Adaptor	**96.9**±0.3	69.9±1.7	**64.2**±2.1	60.3±1.4	61.2±2.7	32.4±1.3	**42.9**±3.0	9.1±1.6	40.7±2.9	**30.9**±1.8	+16.4
RIMU *(ours)*	95.5±0.6	**71.7**±1.5	59.6±1.2	**66.0**±0.9	**71.6**±4.2	**34.8**±2.9	38.5±3.1	18.9±4.7	45.7±2.5	21.0±2.0	+19.6
QWEN3-EMBEDDING-8B ($d = 4096$)											
Frozen	73.1±0.4	60.1±1.6	71.4±0.9	58.0±0.4	62.8±2.3	50.1±1.5	62.7±3.2	39.7±5.8	**62.4**±3.9	**52.7**±1.5	*ref.*
ERM	91.5±0.6	71.4±1.1	71.1±2.6	66.8±0.4	71.4±3.6	52.2±1.5	57.7±2.8	26.8±1.7	60.4±3.5	49.3±1.5	+2.6
NUDGE-M	93.6±0.3	61.2±5.2	71.4±0.9	61.2±0.8	66.4±4.3	50.1±1.5	64.3±4.2	40.0±5.8	**62.4**±3.9	52.7±1.4	+4.4
NUDGE-N	95.9±0.5	72.5±1.9	71.4±0.9	65.9±0.5	71.3±3.2	51.8±1.8	61.8±3.2	39.7±5.8	**62.4**±3.9	52.1±2.6	+8.0
Adaptor	**97.6**±0.5	68.7±1.7	70.4±1.7	61.6±0.5	66.4±4.3	44.2±2.7	58.0±3.1	23.3±4.7	50.8±3.6	36.0±3.5	−5.3
RIMU *(ours)*	96.4±0.7	**74.2**±1.5	**76.1**±1.1	**67.8**±0.7	**72.5**±2.7	**56.4**±0.9	**64.9**±3.5	**40.1**±5.9	62.0±3.6	52.3±1.3	+11.0

Table 10: **C@10 across five frozen encoders**, same ten pools, same 5-fold splits, same hyperparameters — nothing was retuned per encoder. The third and fifth panels are the two encoders of Table 1, read at C@10 instead of N@10. C@10 is all-or-nothing — a query counts only when the *whole* gold set is inside the top ten — so it separates the methods most on the multi-gold pools. The final column is each method's mean gain over that encoder's own frozen index, which is the comparison the table is for: a method that only works on one backbone shows up as a column that changes sign between panels.

Method	**SRA**	**ToolRet**			**MAIR**						$\overline{\Delta}$ %
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
E5-LARGE-V2 ($d = 1024$)											
Frozen	50.8±0.8	28.8±2.0	38.2±1.1	36.5±1.4	37.3±4.3	28.3±2.5	**69.5**±3.3	**39.9**±1.2	**57.3**±2.6	**43.6**±1.3	*ref.*
ERM	86.6±1.3	47.6±2.7	31.2±1.6	38.2±1.6	50.3±2.4	29.9±2.2	59.6±2.2	20.0±3.5	44.1±2.7	37.3±2.2	+6.1
NUDGE-M	87.2±1.1	40.9±3.2	31.0±4.0	39.5±1.3	39.3±4.6	28.3±2.5	69.3±3.1	**39.9**±1.2	**57.3**±2.6	**43.6**±1.3	+10.8
NUDGE-N	88.9±1.1	46.2±3.6	36.4±3.1	41.9±2.1	56.4±3.4	28.3±2.5	69.0±3.1	**39.9**±1.2	**57.3**±2.6	**43.6**±1.3	+19.6
Adaptor	87.8±1.3	47.9±2.6	**50.9**±3.1	**46.4**±2.9	40.5±1.8	17.7±1.8	58.0±4.0	6.8±2.0	32.4±5.4	24.3±2.1	−1.7
RIMU *(ours)*	**89.3**±1.1	**50.4**±4.5	48.2±1.4	43.9±1.6	**59.6**±5.6	**31.4**±2.9	69.3±4.0	39.6±1.8	52.3±3.0	43.3±1.0	+25.8
GTE-LARGE ($d = 1024$)											
Frozen	60.6±1.1	31.2±3.4	44.5±2.9	34.7±1.0	28.9±4.8	28.3±2.5	55.0±2.2	**40.6**±3.5	**51.9**±2.9	**44.9**±2.3	*ref.*
ERM	86.5±1.1	50.9±2.5	42.8±3.5	39.5±1.6	47.6±2.6	31.7±2.6	45.8±2.6	32.8±3.5	50.7±2.7	43.3±3.0	+15.1
NUDGE-M	86.4±1.3	40.5±1.9	39.6±8.7	38.1±1.7	33.5±4.3	28.3±2.5	**55.3**±2.3	**40.6**±3.5	**51.9**±2.9	44.8±2.3	+8.7
NUDGE-N	90.3±1.0	50.4±3.4	45.9±1.5	41.7±1.6	48.7±4.2	**33.4**±3.6	55.0±2.2	**40.6**±3.5	**51.9**±2.9	**44.9**±2.3	+22.1
Adaptor	86.1±1.1	47.4±3.3	51.6±3.7	45.1±1.8	39.4±4.3	24.6±3.0	51.0±3.7	10.4±3.2	32.4±5.0	28.2±2.3	+0.6
RIMU *(ours)*	**90.6**±0.8	**55.9**±2.8	**56.0**±1.3	**48.0**±0.9	**53.2**±5.9	**33.4**±2.0	54.5±2.6	40.3±3.8	51.1±3.1	44.8±2.6	+29.2
BGE-LARGE-EN-V1.5 ($d = 1024$)											
Frozen	59.1±1.4	27.6±3.6	43.2±2.5	37.3±1.1	34.9±4.4	28.1±2.1	64.6±2.6	41.3±2.3	49.0±3.9	**45.7**±2.1	*ref.*
ERM	85.9±1.0	51.5±1.7	46.7±1.5	48.1±1.6	53.3±4.9	30.9±2.1	64.1±2.8	39.1±3.1	48.7±4.1	43.5±3.2	+22.0
NUDGE-M	87.0±1.1	41.5±1.6	43.2±2.5	42.5±1.4	38.4±6.6	28.1±2.1	64.7±2.4	41.3±2.3	49.0±3.9	**45.7**±2.1	+12.2
NUDGE-N	90.9±1.0	53.8±2.5	47.2±1.4	48.5±1.0	55.2±4.5	32.1±3.1	64.6±2.6	40.6±2.7	49.0±3.9	**45.7**±2.1	+25.9
Adaptor	86.0±1.6	48.1±3.2	51.2±2.9	48.7±0.9	39.7±2.1	20.9±5.8	50.7±3.6	17.9±2.8	26.7±3.4	28.6±1.1	−0.4
RIMU *(ours)*	**91.2**±1.2	**56.0**±3.3	**56.3**±1.3	**52.3**±1.0	**58.0**±4.6	**32.9**±2.1	**65.3**±2.6	**41.5**±2.8	**49.1**±4.0	45.3±2.4	+31.2
QWEN3-EMBEDDING-0.6B ($d = 1024$)											
Frozen	44.1±1.1	40.1±2.3	46.6±1.5	42.1±0.5	53.4±3.2	25.3±2.6	31.6±2.3	**15.5**±5.7	**43.0**±3.3	19.0±1.8	*ref.*
ERM	80.3±0.5	52.4±1.8	48.2±2.2	53.8±1.4	64.6±4.3	25.0±2.3	27.0±2.5	11.3±3.2	37.3±2.8	13.8±2.1	+8.2
NUDGE-M	90.7±0.8	40.2±2.2	47.8±2.4	46.4±1.3	56.0±4.3	25.3±2.6	34.0±2.4	**15.5**±5.7	42.3±3.1	18.8±1.7	+12.8
NUDGE-N	90.6±0.8	53.6±1.7	53.5±1.9	53.9±1.0	64.9±6.3	25.3±2.6	35.1±2.6	11.1±3.1	**43.0**±3.3	19.0±1.8	+18.6
Adaptor	**94.3**±0.7	54.6±3.4	**62.5**±2.4	48.7±1.1	55.7±3.5	29.0±1.6	**40.2**±3.2	6.5±1.1	37.0±3.2	**26.6**±1.9	+21.4
RIMU *(ours)*	92.4±0.7	**55.9**±1.9	58.3±1.5	**55.1**±0.8	**69.6**±4.5	**29.4**±3.0	35.6±2.8	15.0±5.0	41.2±3.2	18.3±1.5	+25.3
QWEN3-EMBEDDING-8B ($d = 4096$)											
Frozen	66.9±0.7	43.1±1.3	66.7±1.1	46.4±0.4	55.3±2.3	43.9±1.4	58.8±3.0	32.4±5.8	**57.2**±4.0	**45.8**±1.1	*ref.*
ERM	86.3±0.5	54.4±1.1	69.8±2.9	55.3±0.4	68.6±4.3	46.7±1.4	54.5±2.2	21.7±1.9	55.2±3.7	43.6±1.5	+6.1
NUDGE-M	91.4±0.7	45.0±2.5	66.7±1.1	49.0±1.0	59.0±4.5	43.9±1.4	60.9±3.9	32.4±5.8	**57.2**±4.0	**45.8**±1.1	+5.7
NUDGE-N	92.8±0.5	54.9±1.6	66.7±1.1	54.2±0.3	67.2±5.0	45.9±2.3	58.4±2.7	32.4±5.8	**57.2**±4.0	45.4±1.9	+10.8
Adaptor	**95.6**±1.0	54.6±1.4	68.7±1.8	49.7±0.8	62.1±5.5	40.1±2.9	55.0±2.8	19.1±4.4	46.6±3.4	31.7±3.0	−1.3
RIMU *(ours)*	93.6±0.7	**55.9**±1.4	**74.9**±1.4	**56.9**±0.5	**70.6**±2.8	**48.6**±1.3	**61.2**±3.3	**32.6**±6.3	56.9±3.6	45.7±1.1	+14.7



(a) SRA (b) ToolRet/cust. (c) ToolRet/code (d) ToolRet/web (e) MAIR/Code

(f) MAIR/Legal (g) MAIR/Fin. (h) MAIR/Med. (i) MAIR/Acad. (j) MAIR/Web

Figure 7: **Forgetting, all ten pools.** The four in Figure 5 plus the six omitted there, the same quantity: the share of the first fold's own gain over the frozen index that is still there after ℓ folds have been written. RIMU ends above 90% on every pool (mean 97.1, median 97.4, lowest 90.5) and at or above 100% on three of them, where the later folds reinforced the first rather than overwriting it; it is the higher of the two methods that train everywhere on eight pools of ten. ERM declines on all ten (mean 92.4, lowest 79.4) and is flat only where it sits at the in-sample ceiling, which leaves it nothing to lose. NUDGE-N appears on four pools: on the other six (TOOLRET/CODE, MAIR/LEGAL, MAIR/FIN., MAIR/MED., MAIR/ACAD., MAIR/WEB) there is no gain to retain: it returns the frozen index, to within 0.1 point, rather than training, and a flat line would credit inaction as perfect memory. Where it does train it is the least stable (mean 81.9), losing nearly half of its first-fold gain on MAIR/CODE. Per-fold numbers in Table 11.

E RETENTION, FOLD BY FOLD

Intrusion concerns other queries at the time of a write. Retention asks whether later feedback undoes a gain on queries already served. Held-out testing results alone do not expose that loss, so we revisit the first fold after each subsequent fold is written.

The five folds are read as an arrival stream. After the first ℓ folds are written, the index is scored on each earlier fold separately; fold 5 is never trained on and stays as the ordinary testing column. Figure 7 plots this as the share of each pool's first-fold gain that survives, and Table 11 reports the first fold's own N@10 as three further folds pass through the same index. Both ends are in-sample, so the change measures retention rather than a generalization gap. The testing column can move in the opposite direction: on TOOLRET/WEB, ERM loses 5.1 points on the queries it had already served ($90.21 \rightarrow 85.10$) while its testing score *rises* by 3.2 ($52.41 \rightarrow 55.63$). A reader with only the testing column would conclude that more feedback helped.

Averaged over the folds that had later writes, ERM loses 1.79 points — 2.45 over the seven pools where it is not already at its in-sample ceiling — and NUDGE-N 1.91 on the four pools where it trains at all, while the minimum-change update loses 0.65 and is positive on three pools of ten. That is consistent with the minimum-displacement objective and the auxiliary ownership row, though later queries can still overwrite a shared gold. The mechanisms differ and the table does not pool them — ERM and NUDGE fit the whole training set at once, so their decay is interference from later data, whereas the update here is sequential and its decay would be overwriting — but the symptom is the one a cache operator would notice.

Two rows are flagged rather than dropped. NUDGE-N reduces to a single global step size $\eta^\star$ chosen on an inner dev split, and that selection can return $\eta^\star = 0$, in which case the write is the identity and the cell reproduces the frozen row exactly. That is what happens on six of the ten pools here; on three of them it begins to train once the fourth fold arrives, so the flat line is an absence of action rather than retention. It is the same mechanism as the collapsed step size reported in Appendix D.1, at the end of its range rather than near it. And ERM sits at 97.4–100.0 in-sample on MAIR/FINANCE,

Table 11: **What the update keeps.** The five folds are an arrival stream; after the first ℓ folds are written the index is scored on each earlier fold separately. Rows $\ell{=}1 \ldots 4$ are the *first* fold's own N@10 as three further folds pass through the same index, so they are in-sample at both ends and the change cannot be a generalization gap. $\bar{\Delta}$ averages that change over every fold that had later writes. The testing rows are fold 5, never trained on. ERM and NUDGE fit the whole training set at once, so a decay there is interference from later data; RIMU is sequential, so a decay would be overwriting. † the first fold had no gain to keep: NUDGE-N returned the frozen index rather than training, either throughout or until the fourth fold arrived, so the row is not a retention result and is left out of the mean. [c] ERM is at 97.4–100.0 in-sample, leaving no headroom to lose.

	SRA	**ToolRet**			**MAIR**						mean $\bar{\Delta}$
		cust.	code	web	Code	Legal	Fin.	Med.	Acad.	Web	
ERM											
first fold, $\ell{=}1$	88.76	85.97	98.70	90.21	93.15	92.72	100.00	99.74	94.69	99.77	
$\ell{=}2$	86.55	83.36	97.76	87.81	90.60	91.18	100.00	99.74	93.83	99.72	
$\ell{=}3$	86.58	81.08	96.57	86.67	88.56	90.68	100.00	99.72	93.71	99.53	
$\ell{=}4$	85.40	78.85	95.13	85.10	87.34	89.79	100.00	99.42	93.22	99.42	
$\bar{\Delta}$	-1.93	-4.01	-1.85	-3.16	-3.52	-1.62	-0.21[c]	-0.23[c]	-1.03	-0.37[c]	**-1.79**
testing, $\ell{=}1$	80.49	59.94	52.97	52.41	55.42	46.26	45.69	29.86	48.44	42.84	
testing, $\ell{=}4$	80.41	59.07	47.93	55.63	54.55	42.28	44.95	35.40	45.87	41.88	
NUDGE-N											
first fold, $\ell{=}1$	95.73	85.15	54.42	80.18	91.22	44.21	44.36	42.53	41.80	44.26	
$\ell{=}2$	90.83	83.16	54.42	78.77	68.42	44.21	44.36	42.53	41.80	44.26	
$\ell{=}3$	92.63	83.51	54.42	77.68	67.83	44.21	44.36	42.53	41.80	44.26	
$\ell{=}4$	92.25	82.59	54.42	77.22	71.59	85.96	89.09	42.53	41.80	92.03	
$\bar{\Delta}$	-0.92	-1.37	–[†]	-1.50	-3.86	–[†]	–[†]	–[†]	–[†]	–[†]	**-1.91**
testing, $\ell{=}1$	85.05	61.34	51.26	51.62	55.65	46.05	49.59	41.96	50.23	45.68	
testing, $\ell{=}4$	87.27	60.29	51.26	54.38	54.69	46.14	50.64	41.96	50.23	40.71	
RIMU *(ours)*											
first fold, $\ell{=}1$	93.25	80.63	79.00	72.93	76.19	74.56	75.16	77.50	71.91	73.96	
$\ell{=}2$	92.68	79.52	78.11	73.57	75.33	73.71	75.45	77.96	71.50	72.71	
$\ell{=}3$	93.18	79.13	77.95	74.15	74.74	73.29	75.61	78.02	71.57	72.21	
$\ell{=}4$	93.24	79.03	78.05	74.03	73.57	71.89	75.94	77.97	71.54	71.59	
$\bar{\Delta}$	+0.04	-0.49	-0.29	+0.34	-1.84	-1.84	+0.28	-0.65	-0.49	-1.59	**-0.65**
testing, $\ell{=}1$	84.37	63.18	55.82	50.90	56.46	48.09	49.63	40.99	49.70	45.37	
testing, $\ell{=}4$	89.01	65.45	57.08	55.37	57.72	48.57	50.90	42.36	48.89	44.76	

MAIR/MEDICAL and MAIR/WEB, so it has no headroom to lose — on the second its testing score is 35.40 against a frozen 41.96.

F EFFICIENCY AND MATCHED-REPAIR COMPARISONS

Wall-clock update time and displacement measure different costs. We first report the per-pool time behind the main-text summary, then compare methods after matching how much training ranking repair they achieved.

F.1 UPDATE TIME BY POOL

Figure 4 gives the mean time across pools. Table 12 breaks it down over all ten pools, alongside corpus size and the number of training queries. RIMU's per-query time ranges from 2.3 to 27.3 ms; its total time tracks query count more closely than index size, whereas NUDGE-N's grid search is more sensitive to corpus size.

F.2 DISPLACEMENT AT MATCHED TRAINING REPAIR

Comparing methods at their own settings cannot separate the claim from its cheapest alternative explanation. A method tuned down both moves less and repairs less, so "moves less, disturbs less"

Table 12: **Time to adapt the index, per pool**, seconds for one fold under QWEN3-EMBEDDING-8B; the ten pools behind Figure 4. All five were timed back to back in a single process on one CPU node (8 threads, no GPU), and only the call that produces the new index is timed — retrieval and evaluation are excluded. The frozen arm returns the index untouched and costs under 0.01 s everywhere, so it is omitted. The last row is our update's cost per training query; the pools are divided across two aligned panels for legibility. The two rows to read against each other are **Targets** and **Train q**: our cost correlates with the training queries at 0.80 and with the corpus at 0.29, while NUDGE-N is the other way round (-0.18 and 0.97), since its γ search rescores the whole index once per candidate. On a large index with little feedback that inverts the ordering — on MAIR/ACAD. we are $3.7\times$ cheaper than NUDGE-N, on SRA $1.9\times$ more expensive.

	SRA	**ToolRet**			**MAIR**		
		cust.	code	web	Code	Legal	Fin.
Targets	26,262	3,367	3,794	37,292	23,463	52,743	49,083
Train q	4,319	785	1,399	4,183	805	560	503
ERM	0.46	0.12	0.13	0.80	0.20	0.48	0.32
NUDGE-M	11.3	0.20	0.33	2.65	1.38	3.07	2.65
NUDGE-N	13.3	1.45	1.73	20.8	8.89	20.5	21.0
Adaptor	240.9	71.3	75.4	276.3	47.1	111.5	27.0
RIMU *(ours)*	26.0	2.25	3.15	49.2	6.46	9.10	5.98
ms per query	6.0	2.9	2.3	11.8	8.0	16.2	11.9

	MAIR			mean	rel.×
	Med.	Acad.	Web		
Targets	30,371	111,351	100,149		
Train q	331	845	650		
ERM	0.53	0.71	1.69	0.5	0.04
NUDGE-M	2.13	8.52	10.1	4.2	0.29
NUDGE-N	11.0	79.9	78.5	25.7	1.76
Adaptor	200.1	43.2	536.7	163.0	11.17
RIMU *(ours)*	4.55	21.5	17.7	14.6	1.00
ms per query	13.7	25.4	27.3		

is also what an under-tuned baseline looks like. Our claim is about an objective — the smallest displacement that repairs the observed ranking — and an objective can only be judged at equal repair.

Each method's own knob is therefore swept over four pools and three folds, and every setting records the training N@10 it reached, the displacement it spent, the testing N@10 it generalised to, and the strict invasion count it caused. Each curve is interpolated to a common training N@10, the median of our own settings on that cell, and the other three read off there. Matching is on training N@10 because that is the work the method was asked to do; matching on testing would assume the conclusion.

Table 13 distinguishes an unreachable target from an inefficient update. Neither NUDGE variant reaches RIMU's matched training score on SRA within its sweep. In the cells where NUDGE-N does reach that score, RIMU uses less displacement and causes fewer invasions; the caption gives the paired differences and the number of comparable cells.

G OWNERSHIP AND CROSS-QUERY INTERFERENCE

The main-text ownership ablation has a small overall testing effect. The two analyses below check where intrusion occurs and then evaluate the row on the queries it was designed to guard, alongside the full pools. These diagnostics probe the row's scope; they do not establish a guarantee for testing rankings.

	SRA	ToolRet		MAIR
		cust.	code	Legal
RIMU *(ours)*				
displacement	4.4	6.2	5.3	8.1
testing N@10	89.4	66.5	59.4	49.1
invasions per query	0.36	0.77	0.77	0.86
NUDGE-N				
displacement	*o.r.*	16.9	8.5	33.0
testing N@10	*o.r.*	63.1	53.8	42.9
invasions per query	*o.r.*	1.17	1.07	1.44
NUDGE-M				
displacement	*o.r.*	*o.r.*	9.8	18.4
testing N@10	*o.r.*	*o.r.*	30.3	32.2
invasions per query	*o.r.*	*o.r.*	4.77	4.23

Table 13: **What the update costs at the same repair.** Each method's own knob is swept over the same four pools and three folds, and each curve is interpolated to a common training N@10 — the median of our own settings on that cell — so that every method has repaired the observed rankings to the same degree before anything else is compared. Displacement is in thousands of degree-keys, invasions are per testing query under the strict definition. *o.r.* (out of reach) means the baseline's entire grid falls short of that training N@10: on SRA neither NUDGE variant reaches it at any γ, so the comparison there is not that they spend their displacement worse but that they cannot do the work. Over the 9 cells where NUDGE-N does reach it, we spend 70% less displacement ($t = -4.43$), score 12.7 more N@10 ($t = 2.15$) and cause 46% fewer invasions ($t = -8.11$). This is the comparison the "moves less" claim needs: a method tuned down moves less and repairs less, and matching on repair removes that explanation.

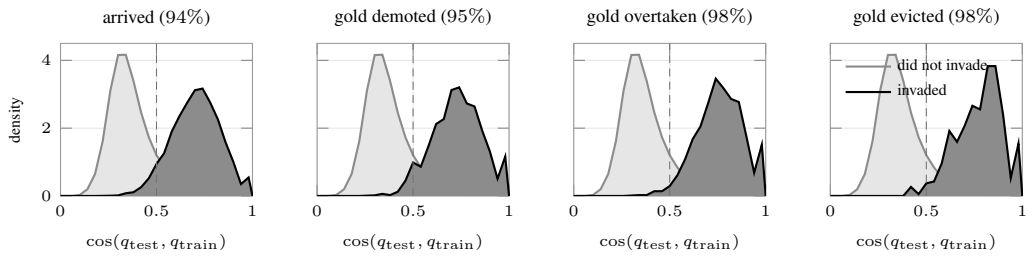


Figure 8: **Query similarity and retrieval intrusion.** Across four pools and three folds, panels group testing-query–written-key pairs by the effect of an update, from entering the top ten to displacing a gold. The x-axis measures similarity to the training query that moved the key; the dashed line marks the ownership threshold of 0.5. The share above this threshold rises from 94.4% of arrivals to 98.1% of gold evictions, compared with 11.3% of pairs causing no damage.

G.1 Similarity of the queries affected by an update

The ownership row in equation 5 applies to a nearby query claiming a disjoint gold set. The cosine threshold is 0.5. To check whether this criterion targets relevant interference, we measure how similar an affected testing query is to the training query that moved the intruding key.

A written key that arrives in a testing query's top ten need not have cost that query anything: it may have displaced another non-gold. The panels therefore grade each arrival by what it did — a gold of that query fell a place, this key overtook a gold, a gold left the top ten — and ask, for each grade, where on the similarity axis it came from. If the threshold were aimed at the wrong quantity, the harder grades would spread out rather than concentrate above it.

	SRA	ToolRet		MAIR	all four
		cust.	code	Legal	
(a) on **benchmark**$^{-}$					
training N@10	+0.28	+0.77	+1.36	+0.35	+0.69
testing N@10	+0.37	+0.37	+0.07	-0.20	+0.15
invasions per query	-0.012	-0.011	-0.012	-0.012	-0.012
(b) over the full pool					
training N@10	+0.55	+0.53	+1.45	+0.21	+0.69
testing N@10	+0.42	+0.22	+0.18	-0.31	+0.13
invasions per query	-0.010	+0.020	+0.000	-0.014	-0.001

Table 14: **The ownership row, scored where it acts and everywhere.** Each cell is the paired difference between the method with the row and the same method with its price set to zero, averaged over three folds. **Benchmark**$^{-}$ keeps the training and testing queries that the row is emitted for: a query with some training query within $\cos 0.5$ of it claiming a disjoint gold set. The rule is the method's emission condition and consults no outcome. Over the twelve (pool, fold) cells the row is worth $+0.69$ training N@10 ($t = +5.1$, 12 of 12 cells) and -0.012 invasions per query ($t = -3.6$, 10 of 12) on benchmark^{-}, against -0.001 invasions ($t = -0.2$) on the full pools. Testing N@10 moves $+0.15$ with a fold-to-fold spread of 0.38, which is not a difference: restricting the evaluation to the protected queries does not recover it, and this is the limit of what the row is shown to do.

G.2 Effect on guarded queries and the full pool

Table 3 scores the ownership row over every testing query and finds a $+0.05$ N@10 mean difference, too small to establish an overall ranking benefit. The guarded queries are a minority of each pool, so we also measure the effect within that subgroup.

Benchmark^{-} removes one of them. It keeps the training and testing queries the row is emitted for — a query with some training query within $\cos 0.5$ of it claiming a disjoint gold set — and the rule is the method's own emission condition, applied without consulting any outcome. The full pools are reported beside it, since a subgroup shown alone is not evidence. Table 14 gives both per pool; the body table carries the same measurement summarised.

On benchmark^{-}, the row reduces invasions by 0.012 per query and raises training N@10 by 0.69 on average. Its $+0.15$ testing N@10 has substantial fold-to-fold variation, while the full-pool invasion change is close to zero. The evidence supports a narrower claim: the row reduces observed interference near similar, disjoint-gold queries, without a clear overall testing ranking gain.