RECURRENT READING FOR SINGLE-VECTOR RETRIEVAL

Anonymous authors
Paper under double-blind review

ABSTRACT

In reasoning-intensive retrieval, a relevant document may share an underlying principle with the query rather than its words, requiring several points of contact between query and document. Late-interaction models provide this capacity but incur additional cost at retrieval time. We ask whether a single vector representation can instead be trained to encode this information. In our aligned-score configurations with untied output projections, we observe *slot locking*: corresponding query and document reads attain the highest similarity, and the additional reads partly compensate for a weakened base embedding. We introduce Recurrent Reading for Single-Vector Retrieval (RLI), which reads token states with a recurrent head and matches each query read to its best document read through a projection shared across reads. We find that late interaction is more useful as a training signal than as a scoring rule. At retrieval time, RLI ranks documents using one inner product between the packed query and document vectors. Across four backbones from 0.6B to 4B and three model families, removing the reads from the trained model costs 3.9 to 5.2 nDCG@10. On BRIGHT, using original queries without query-time generation or reranking, a 4B RLI averages 42.42 ± 0.88 nDCG@10 over three seeds (best 43.41), 4.1 points above the strongest published retriever-only baseline in our comparison and 3.6 above the same recipe without reads (38.80 ± 1.06).

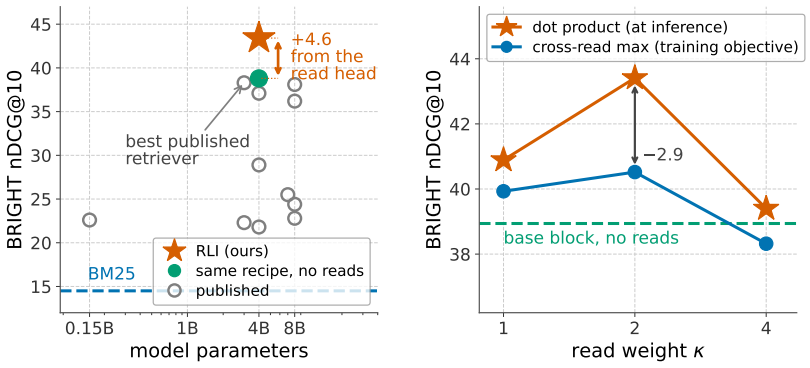


Figure 1: **Left:** BRIGHT nDCG@10 against model size for retrievers evaluated alone on original queries. The star is the best RLI run and the green circle the three-seed mean of the same recipe without reads, so the marked gap compares one run against a mean. **Right:** RLI scored with each aggregator at each read weight κ; the dashed line is the base block alone.

1 INTRODUCTION

Dense retrievers (Karpukhin et al., 2020; Izacard et al., 2021; Wang et al., 2022; Zhang et al., 2025) encode a query and a document independently into single vectors and rank documents by the inner product. Because document vectors can be precomputed, this design makes first-stage retrieval cheap at web scale, and it has become the standard architecture for embedding models built on large

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.

language models. Recent progress has come mainly through training data: synthetic reasoning-intensive corpora (Shao et al., 2025; Das et al., 2025) and graded relevance annotation (Chen et al., 2026). Yet dense retrievers still degrade sharply when relevance is not a matter of surface correspondence. On BRIGHT (Su et al., 2025), a benchmark of such queries, embedding models that lead BEIR (Thakur et al., 2021) and MTEB (Muennighoff et al., 2023) lose most of their accuracy.

BRIGHT queries are not harder paraphrases. A coding query describes a data-transformation problem and is answered by the documentation of a function the query never names; a competition mathematics query is answered by a different problem that happens to use the same theorem. Recognising the relevant passage means combining several facts rather than matching one. Late interaction (Khattab & Zaharia, 2020; Santhanam et al., 2022b) supplies the several points of contact this calls for by keeping one vector per token, but pays for them at search time with a token-level index and a dedicated search engine (Santhanam et al., 2022a). Most reported progress on BRIGHT instead leaves the encoder unchanged and adds machinery around it, a language model that rewrites the query or a cross-encoder that reranks the candidates, and those gains shrink substantially once the rewriting is removed (Sun et al., 2025). We investigate whether changing the training objective can improve reasoning-intensive retrieval while retaining a single-vector representation.

A central obstacle lies in the training objective rather than in the single-vector serving form itself. The natural way to give a dense retriever more capacity is to let a text produce several components instead of one, as multi-vector retrieval does (Humeau et al., 2019; Luan et al., 2021; Zhang et al., 2022). Those methods keep the components separate and aggregate them at search time; keeping a single-vector index instead means concatenating them, and their inner product then binds component t of the query to component t of the document by index, an arbitrary label, rather than by content, so the objective never supervises the similarity between component t of one text and component u of another (§3.2). We call this property *slot locking*. It has a structural and an empirical part. Structurally, the aligned score contains no cross-read similarity term (Observation 1). Empirically, with untied output projections, the same-index document read is the best match for every query read on all monitored positive pairs. We diagnose the empirical part; §4.3 shows that tying the projections removes it, although the structural part still holds. For the untied configurations, rescoring with the cross-read maximum also leaves nDCG@10 unchanged on every BRIGHT split (§4.3). Its cost is not that the extra components sit idle but that they substitute: protecting the base embedding with its own loss raises its score and removes almost exactly as much from what the components contribute (Figure 2).

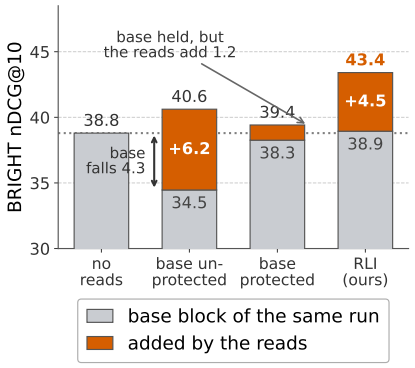


Figure 2: Base-block score and the amount the reads add, for configurations A, B, C and F of Figure 5.

To remove slot locking we propose Recurrent Reading for Single-Vector Retrieval (RLI), which changes the objective the encoder is trained through while preserving the single-vector retrieval interface (§3). The components are produced by a recurrent read head, a gated recurrent unit (Cho et al., 2014) that attends over the token states T times, computing the attention at step t from the state accumulated so far; we call its outputs *reads*. Training scores a pair by late interaction over the reads, matching each query read against its best document read. This objective depends on the document reads only as a set, so the scoring function prescribes no fixed slot correspondence; whether one arises is then an empirical question. A second change is necessary for the first to take effect: the read projections must be tied, or the reads occupy unrelated random subspaces, the monitored positive pairs stay diagonally matched, and the new objective does not improve the retrieval score. A natural consequence of training through late interaction would be to retrieve with it, and that is what we expected. The opposite holds: the plain dot product of the concatenated vector outperforms the objective the model was trained through, on every seed and at every read weight. In our experiments, late interaction improves training, while the aligned dot product gives better retrieval scores at inference. One possible explanation is that maximum-based supervision encourages useful individual reads whose aligned similarities can subsequently be combined.

Our experiments separate the effects of projection sharing and the training objective. With base-block protection retained, tying the output projections under the aligned objective raises nDCG@10 from 39.42 to 41.51; adding late-interaction supervision raises it to 43.41 (single runs at a shared seed; RLI's three-seed mean is 42.42, §4.4). Across four backbones spanning 0.6B to 4B and three model families, removing the reads from the trained model reduces nDCG@10 by 3.9 to 5.2 points. On the 4B backbone, fixed pooling adds 0.87 points over its own base block, compared with 4.47 for RLI. These results suggest that additional pooled components alone do not explain the gain. Across three seeds, RLI averages 42.42 ± 0.88, compared with 38.80 ± 1.06 for the same recipe trained without reads ($p = 0.011$, §4.2). A single learned read already reaches 42.46; slot locking decides whether further reads add or degrade (§4.5).

Contributions. (i) We identify *slot locking* in single-vector retrievers built from several pooled components and measure it with an off-diagonal argmax rate, a read cosine and a rescoring test (§3.2, §4.3). (ii) We show that tying the read projections and training through late interaction over reads remove it, while retrieval keeps one vector per document and one inner product (§3, §4.4). (iii) We find that this training objective is a worse scorer than the plain inner product at every read weight and on all four backbones (§4.5, Appendix F). (iv) On BRIGHT without query-time generation or reranking, a 4B RLI averages 42.42 ± 0.88 nDCG@10 over three seeds, 3.6 above the same recipe without reads and above every published retriever-only model we compare against (§4.2).

2 Related Work

Reasoning-intensive retrieval. BRIGHT (Su et al., 2025) showed that retrievers strong on BEIR (Thakur et al., 2021) and MTEB (Muennighoff et al., 2023) fail when relevance requires inference. Most responses change the training data: synthetic hard queries (Shao et al., 2025), LLM reasoning trajectories (Das et al., 2025) and graded, reasoning-weighted pairs (Chen et al., 2026). We hold the data fixed and change the objective, which is complementary. Pipelines that rewrite the query and rerank (Sun et al., 2025; Chen et al., 2026) follow a different protocol; we compare only retrievers on original queries and list pipelines in Appendix C.

Late interaction and multiple vectors. ColBERT (Khattab & Zaharia, 2020; Santhanam et al., 2022b) sums, over query tokens, the maximum similarity to any document token, and needs a token-level index and a dedicated engine (Santhanam et al., 2022a); Reason-ModernColBERT (LightOn AI, 2025) applies it to reasoning retrieval. We apply the same function to a few learned reads, only during training. Multi-view representations (Luan et al., 2021; Zhang et al., 2022), Poly-encoders (Humeau et al., 2019) and MetaEmbed (Xiao et al., 2026) keep several vectors and aggregate them at inference, so their objectives are permutation-invariant and slot locking does not arise. TCT-ColBERT (Lin et al., 2020) and MUVERA (Dhulipala et al., 2024) also pair late interaction with single-vector search, by distillation and by a fixed-dimensional encoding. Our head resembles recurrent attention readers (Vinyals et al., 2015; Sukhbaatar et al., 2015), and Slot Attention (Locatello et al., 2020) and PMA (Lee et al., 2019) also pool several vectors; Slot Attention resolves arbitrary slot order by Hungarian matching, and our maximum plays that role across query and document reads.

3 Method

RLI changes what a dense retriever is trained *through* while preserving the single-vector retrieval interface. This section defines the recurrent read head and the packed vector it produces (§3.1), describes slot locking under the aligned scoring objective (§3.2), and gives the late-interaction objective that removes it (§3.3). The deployed model is the packed vector scored by an inner product; the maximum appears only in the loss. Figure 3 places the three serving forms side by side.

3.1 The recurrent read head and the packed vector

A retrieval model maps a text x to a vector $\phi(x)$ and ranks a corpus by $\langle \phi(q), \phi(p) \rangle$. Let f_θ be a decoder-only backbone producing token states $H \in \mathbb{R}^{L \times d}$ under a padding mask μ. The conven-

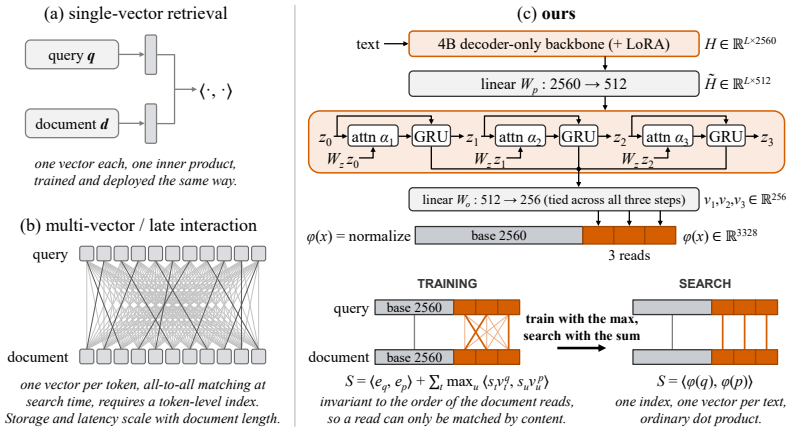


Figure 3: **(a)** Single-vector retrieval: one vector per text, one inner product. **(b)** Late interaction: one vector per token, all-to-all matching at search time over a token-level index. **(c)** RLI: a recurrent head reads the token states $T{=}3$ times through one tied projection W_o; training matches each query read to its best document read, search uses the inner product of the packed vectors over one index.

tional construction takes the last non-padding state, $e = h_\ell$, and normalises it; we retain this vector, call it the *base block*, and add T learned reads alongside it. Throughout, $\bar{u} = u/\|u\|$.

The head has parameters $\Theta = \{W_p, W_z, \mathrm{GRU}, W_o, s\}$ and consumes token states the backbone has already computed, so each text is encoded once. With $\tilde{H} = HW_p^\top$ projected into a working space of width $m \ll d$ and $z_0 = W_p e$, for $t = 1, \dots, T$:

$$a_{t,i} = \frac{\langle \tilde{h}_i,\ W_z z_{t-1} \rangle}{\sqrt{m}} + \log \mu_i, \qquad \alpha_t = \mathrm{softmax}(a_t), \tag{1}$$

$$r_t = \sum\nolimits_i \alpha_{t,i}\, \tilde{h}_i, \quad z_t = \mathrm{GRU}(r_t,\ z_{t-1}), \qquad v_t = W_o\, z_t \in \mathbb{R}^c. \tag{2}$$

Equation 1 is a single-query attention whose query vector is a linear function of the accumulated state, and equation 2 folds the resulting read back into it. Read t therefore attends conditioned on what reads 1 through $t-1$ retrieved, so the map $H \mapsto (v_1, \dots, v_T)$ does not factorise into T independent functions. The recurrence acts *at* pooling rather than after it, unlike latent reasoning over an already-pooled query representation (Jin et al., 2026); §4.4 isolates this against a matched set of fixed pooling functions. A single W_o is shared across all T steps, which matters for the objective below and which we return to in §3.2.

The packed vector. The representation concatenates the base block with the scaled reads and normalises the result, $\phi(x) = \frac{1}{Z}\,[\,\bar{e}\,;\, s_1\bar{v}_1\,;\,\dots\,;\, s_T\bar{v}_T\,] \in \mathbb{R}^{d+Tc}$, with s learned and each block normalised before scaling; because every block is a unit vector the normaliser is the global constant $Z = (1 + \sum_t s_t^2)^{1/2}$, and

$$\langle \phi(q), \phi(p) \rangle = \frac{1}{Z^2}\Big(\langle \bar{e}_q, \bar{e}_p \rangle + \sum\nolimits_t s_t^2\, \langle \bar{v}_t^q, \bar{v}_t^p \rangle \Big). \tag{3}$$

This is also the retrieval rule: every document is encoded once into $\phi(x)$ and stored in a single index, and documents are ranked by this inner product. The read blocks carry a global constant κ, folded into the stored vectors at index time (§4.1), which by equation 3 weights read t by $(\kappa s_t)^2$. Query time therefore keeps the shape of dense retrieval, one vector per document and one inner product, with no per-token structure, but it is not free: the stored vector is $d + Tc = 3{,}328$ wide against $d = 2{,}560$, so memory traffic and inner-product arithmetic grow by about 30%.

3.2 Slot Locking

Write $\nu_t = s_t\bar{v}_t$ and $S_{\mathrm{dot}}(q,p) = \langle \bar{e}_q, \bar{e}_p \rangle + \sum_t \langle \nu_t^q, \nu_t^p \rangle = Z^2 \langle \phi(q), \phi(p) \rangle$, which ranks identically to equation 3.

Observation 1 (Slot locking)**.** *Under S_{dot}, for any loss $\mathcal{L}$ that depends on a query–passage pair only through its score, $\partial\mathcal{L}/\partial\nu_t^q \in \mathrm{span}\{\nu_t^p\}_p$.*

This follows from $\partial S_{\mathrm{dot}}/\partial\nu_t^q = \nu_t^p$ and the chain rule. The statement is about the *direction* of the gradient, not its magnitude: for InfoNCE the coefficients are the softmax weights, which depend on the full score and therefore on every read, so the reads are coupled during optimisation. What the observation excludes is narrower and is what matters here: the score contains no cross-read similarity term $\langle\nu_t^q, \nu_u^p\rangle$ for $u \neq t$, so cross-read agreement is never itself optimised. The cross-read geometry is left to initialisation and to whatever the shared backbone and head induce, and the slot index, an arbitrary label, is the only thing the score uses to bind query reads to document reads. That this leaves the extra reads substituting for the base embedding rather than adding to it is an empirical finding (Figure 2, Figure 5), not a corollary.

A second mechanism is architectural. With untied $W_o^{(t)}$ the reads occupy unrelated random subspaces, and we measure the aligned pairing attaining the cross-read maximum from the first diagnostic step onwards, whatever the loss. The two mechanisms are not interchangeable. Replacing S_{dot} while keeping separate projections leaves the maximum pinned to the diagonal, since the aligned pairing already attains it. Tying while keeping S_{dot} leaves Observation 1 in force, so the loss still never rewards cross-read agreement; it does not follow that none arises, because the shared projection is a property of the head rather than of the objective, and §4.3 finds that it alone moves the reads into a common space. §4.4 reports the measurement that separates these cases.

3.3 Late-interaction supervision over reads

Training scores a pair by matching each query read to its best document read,

$$S_{\max}(q,p) = \langle \bar{e}_q, \bar{e}_p \rangle + \sum\nolimits_t \max_u \langle \nu_t^q,\, \nu_u^p \rangle. \tag{4}$$

Three properties matter. $S_{\max}$ depends on the document reads only as a multiset, so relabelling them leaves this term unchanged and slot identity cannot be expressed in it. The full objective is not permutation-invariant: the lexical term targets read 1 and the diversity term refers to consecutive reads, both by construction. $S_{\max}$ dominates the inference score pointwise, $S_{\max} \geq S_{\mathrm{dot}}$, with equality exactly when the maximum is attained by the aligned term in every comparison. And $\partial S_{\max}/\partial\nu_u^p$ is non-zero only for the pairs that attain the maximum, so training determines which read binds where instead of presupposing it. The base term is kept outside the maximum on purpose: it anchors the shared geometry and is never permuted. The loss is the InfoNCE form (Oord et al., 2018) with $S_{\max}$ in place of the inner product, over a batch of queries each with one positive and $G-1$ hard negatives, with in-batch negatives gathered across devices, temperature τ, and a per-sample weight w_q, the reasoning-intensity weight of Chen et al. (2026) computed from the ratio of the losses of the original query and of an offline chain-of-thought (CoT). The CoT is never used at inference.

Auxiliary objectives. Five auxiliary terms accompany the loss, each addressing a failure mode we observed before introducing it: *base-block retention* applies the same contrastive loss to the base block alone, without which it degrades into the substitution behaviour of §1; *reasoning-anchor distillation* makes the base geometry predictive of the rewrite's embedding through a head discarded at inference; *lexical fidelity* requires the first read to preserve an exact-token channel; *read diversity* penalises overlap between consecutive query reads under an entropy floor; and *usage balance* keeps the assignment of query reads to document reads spread across the T document reads. Appendix A gives each term in full and Table 4 its weight. Setting $s = 0$ recovers the base-only scoring function. The read-free training baseline additionally removes the lexical-fidelity, read-diversity and usage-balance losses, while retaining base-block retention and reasoning-anchor distillation (Appendix A).

4 Experiments

Our experiments answer four questions. **RQ1:** does reading at pooling improve a strong single-vector recipe when backbone, data and every other setting are held fixed (§4.2)? **RQ2:** what does slot locking cost, and does removing it let extra reads add rather than degrade (§4.3, §4.4, §4.5)? **RQ3:** which design choices matter at training and at search time: the objective, projection tying,

Table 1: **BRIGHT, per split, retriever-only on original queries** (nDCG@10). Best values among the published baselines and the best RLI run are shown in bold. Appendix B adds six further published baselines and groups the rows by system type. Excluding Pony (§4.2), the three-seed mean of RLI is 42.32 against 40.41 for the strongest baseline.

	StackExchange							Coding		Theorem-based			
Model	**Bio.**	**Earth.**	**Econ.**	**Psy.**	**Rob.**	**Stack.**	**Sus.**	**Leet.**	**Pony**	**AoPS**	**TheoQ.**	**TheoT.**	**Avg.**
BM25	18.9	27.2	14.9	12.5	13.6	18.4	15.0	24.4	7.9	6.2	10.4	4.9	14.5
Qwen3-Embedding-4B	17.8	34.7	16.9	23.3	12.5	16.2	16.8	35.7	1.4	9.8	35.5	41.5	21.8
Qwen3-Embedding-8B	21.0	33.0	18.4	26.1	15.7	19.4	17.3	33.8	1.2	9.4	39.2	39.3	22.8
llama-nemotron-embed-3b-v2	31.1	36.7	22.3	28.4	18.0	18.4	20.3	32.1	6.8	12.1	25.1	16.5	22.3
Reason-ModernColBERT (150M)	33.3	41.0	24.9	30.7	21.1	20.6	20.3	31.1	8.5	9.2	19.5	11.2	22.6
ReasonIR-8B	26.2	31.4	23.3	30.0	18.0	23.9	20.5	35.0	10.5	14.7	31.9	27.2	24.4
RaDeR-gte-Qwen2-7B	34.6	38.9	22.1	33.0	14.8	22.5	23.7	37.3	5.0	10.2	28.4	35.1	25.5
Seed-1.5-Embedding	34.8	46.9	23.4	31.6	19.1	25.4	21.0	**43.2**	4.9	12.2	33.3	30.5	27.2
DIVER-Retriever	41.8	43.7	21.7	35.3	21.0	21.2	25.1	37.6	13.2	10.7	38.4	37.3	28.9
ReasonEmbed-Llama-3.1-8B	55.4	56.2	35.2	48.5	32.1	37.3	41.1	28.8	16.8	9.1	37.9	36.6	36.2
ReasonEmbed-Qwen3-4B	55.4	54.5	34.9	46.9	34.0	36.1	37.4	34.5	13.6	11.3	**41.4**	45.1	37.1
ReasonEmbed-Qwen3-8B	55.5	56.6	36.2	47.4	**35.3**	36.6	39.1	33.6	16.4	12.5	**41.4**	47.2	38.1
llama-nv-embed-reasoning-3b	63.4	60.2	**39.5**	45.5	32.6	34.0	**43.3**	37.5	15.0	10.5	39.5	38.5	38.3
RLI	**70.30**	**64.26**	38.51	**49.03**	33.93	**41.55**	41.18	29.34	**45.47**	**19.04**	39.07	**49.25**	**43.41**
RLI, 3 seeds	68.54	63.06	36.92	47.43	34.27	40.92	40.18	27.78	43.51	19.22	38.56	48.61	42.42 ± 0.88
our recipe, no reads (one run)	63.02	59.33	35.15	40.55	28.03	36.64	37.97	15.65	47.74	13.65	40.24	47.23	38.77

the inference scorer, the serving form and the number of reads (§4.4, §4.5)? **RQ4:** does the result transfer across model scales, backbone families and document lengths (§4.6)?

4.1 Experimental setup

Benchmark. We evaluate on BRIGHT (Su et al., 2025): 12 splits, 1,384 queries, and corpora ranging from 7.9K to 414K documents. Queries are used as given, without rewriting or expansion, and we report nDCG@10. Texts are encoded with a maximum length of 8192 tokens and the benchmark's task instruction. Full details are given in Appendix A.

Baselines. Table 1 compares RLI against the published retriever-only field on BRIGHT. We include the lexical baseline BM25; Qwen3-Embedding-4B and 8B (Zhang et al., 2025) and llama-nemotron-embed-3b-v2 (NVIDIA, 2026b); the late-interaction model Reason-ModernColBERT (LightOn AI, 2025); and the reasoning-specialised retrievers ReasonIR-8B (Shao et al., 2025), RaDeR-gte-Qwen2-7B (Das et al., 2025), Seed-1.5-Embedding (ByteDance Seed, 2025), DIVER (Sun et al., 2025), ReasonEmbed models (Chen et al., 2026) and llama-nv-embed-reasoning-3b (NVIDIA, 2026b). MRE-T1, which reports 39.6, is not in this table because it generates a reasoning chain at query time and so falls outside the retriever-only protocol; it appears in Table 8. Appendix A gives the source of every published figure.

Training data. All configurations are trained on a mixture of public reasoning-retrieval data (Appendix A). The mixture consists of ReasonEmbed (Chen et al., 2026) and ReasonRank (Liu et al., 2026). Three domains are oversampled (Appendix A).

Model and training. The backbone is a 4B decoder-only model (Yang et al., 2025) fine-tuned on MS MARCO (Bajaj et al., 2016); we deliberately do not start from a reasoning-specialised retriever, so the measured effect is attributable to the method rather than the initialisation. Low-rank adapters (Hu et al., 2021) are trained jointly with the read head for one epoch, with $m = 512$, $c = 256$ and $T = 3$, giving a stored vector of 3,328 dimensions and 3.28M head parameters. We select $\kappa \in \{1, 2, 4\}$ on 1,000 queries sampled at random from the ReasonIR training data (Shao et al., 2025), which is not in our training mixture, ranking each query against the positives and hard negatives pooled over the sample. This selects $\kappa = 2$, which we use for all splits, queries and configurations; Table 9 reports BRIGHT at every κ as a sensitivity analysis.

4.2 Main results

Table 1 reports the per-split comparison. Across three seeds RLI averages 42.42 nDCG@10, 4.1 points above llama-nv-embed-reasoning-3b, the strongest published retriever, and 5.3 above ReasonEmbed-Qwen3-4B, which shares our backbone family and parameter count; its best run reaches 43.41 and ranks first on seven of the twelve splits (six for the seed mean). Against our own

recipe trained without reads, the comparison that isolates the method, RLI-4B improves on ten of the twelve splits, the exceptions being Pony and TheoremQA-Questions. Same-recipe seed variance on this benchmark is close to one nDCG@10 point, so we trained the identical recipe three times. The three runs average 42.42 ± 0.88; the read-free recipe, measured the same way, averages 38.80 ± 1.06. The reads are therefore worth 3.62 points, and a Welch test confirms the gap ($t = 4.55$, $p = 0.011$, 95% CI $[+1.38, +5.86]$; Appendix D). The gains are broad rather than concentrated: seven of the twelve splits improve by more than four points over the read-free baseline, led by LeetCode, Psychology and Biology. Pony, where we lead the field by the most, is inherited from the training mixture rather than the reads: the read-free baseline already scores 47.74 there against 22.0 for the best external system (GritLM-7B, Table 6), and the reads lower it. Excluding Pony therefore enlarges the effect of the reads, from $+3.6$ to $+4.4$ (42.32 against 37.95), while narrowing the margin over published retrievers: RLI remains first, 1.9 points above llama-nv-embed-reasoning-3b (40.41). LeetCode is our weakest relative result, 29.34 against Seed-1.5-Embedding's 43.2. There the read-free recipe scores only 15.65, below BM25; this split is trained with groups of two and duplicated twelve times (Appendix A), and we have not isolated which setting causes the drop. The reads recover 13.69 points, consistent with the exact-token channel the first read is trained to keep (Table 14). Excluding both splits where the read-free recipe is anomalous, Pony and LeetCode, the reads add $+3.6$ (43.77 against 40.18), unchanged from all twelve splits. Mean Recall@100 is 76.6, with substantial variation across splits (Table 14).

4.3 Slot locking is measurable, and tying is what unlocks it

Table 2 reports off-diagonal argmax rates and read cosines; the following analysis reports the corresponding changes in retrieval performance. Slot locking makes two predictions testable on a trained model without retraining it. If reads bind by position, then for a positive pair the best-matching document read for query read t is read t itself; we measure this as the *off-diagonal argmax rate* over every (query, gold passage) pair. A zero rate on gold pairs does not by itself imply that rescoring with the cross-read maximum of equation 4 preserves the ranking, since a non-gold passage may still gain from an off-diagonal match; we therefore also report Δ, the change in nDCG@10 when the model is rescored with the cross-read maximum, in the text alongside each configuration. The rate is also not invariant to reparameterisation: applying an orthogonal R_t to read t of queries and documents alike leaves every aligned score, and hence retrieval, unchanged while altering the cross-read similarities. It therefore characterises the cross-read geometry a run happens to arrive at, not retrieval quality, which is why we read it alongside Δ and the read cosine rather than on its own. Table 2 reports these for five models trained with the same backbone, data and recipe.

Table 2: **Off-diagonal argmax rate** on BRIGHT at inference, averaged over splits (0% = slot-locked); the end-of-training rate on training pairs, in Table 16, is a different population and runs higher. Read cos. is the mean cosine between a document's three reads. Configurations follow Figure 5.

	Obj.	Tied	Off-diag.	Cos.
B	dot	no	0.00%	-0.04
C	dot	no	0.00%	-0.03
E	max	no	0.00%	-0.00
G	dot	yes	10.3%	0.84
F	max	yes	$\mathbf{37.1\%}$	$\mathbf{0.88}$

For B and C the rate is 0.00%: in none of the $23{,}388$ comparisons does a query read prefer a non-aligned document read, and rescoring with the maximum changes nDCG@10 by $+0.00$ on every split. The same holds for E, which trains through the maximum but with untied projections: the maximum alone leaves the lock in place. Tying alone breaks it. Configuration G keeps the aligned objective and ties the projections, and its three reads are no longer near-orthogonal: their mean cosine is 0.84, against -0.04 to -0.00 for B, C and E, and 10.3% of pairs bind off the diagonal, the maximum and the sum disagreeing at inference by -1.93. It is therefore the shared projection rather than the objective that moves the reads into one common space, where a query read can be matched by whichever document read carries the closest content. Observation 1 allows this: the loss failing to reward cross-read agreement does not prevent the head from producing it. What the objective decides is how much of that freedom is used, and F uses most of it, binding off the diagonal on 37.1% of pairs with the two scorers disagreeing by -2.89. Protecting the base block (B to C; Figures 2 and 5) raises its score from 34.46 to 38.26 and reduces what the reads add from $+6.16$ to $+1.16$, while the total falls by 1.20 (40.62 to 39.42). Under the aligned objective, the reads

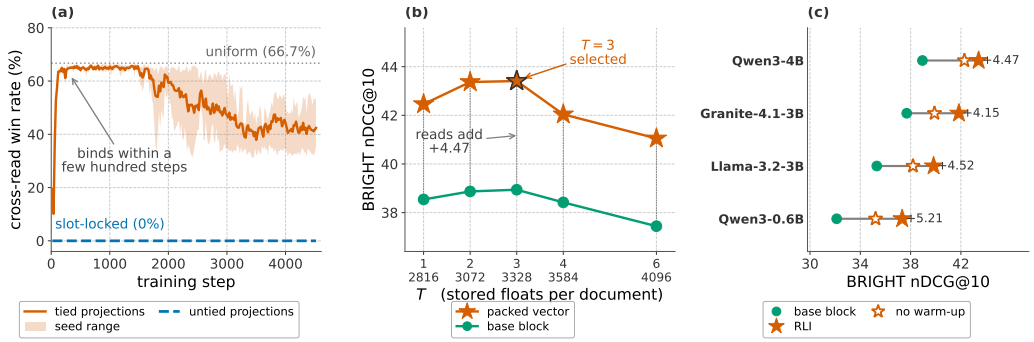


Figure 4: **(a)** Cross-read win rate during training, i.e. the off-diagonal argmax rate on positive training pairs (0% = slot-locked, 66.7% = uniform at T=3); mean of three runs, band their range. **(b)** BRIGHT nDCG@10 against the number of reads T (stored floats per document under each tick), packed vector and its base block. **(c)** Four backbones: circle, base block; filled star, packed vector; hollow star, no MS MARCO warm-up; labels, what the reads add. Numbers in Appendix F.

compensate for a weakened base rather than add to it; under RLI, the base is protected and the reads still add $+4.47$.

4.4 Where the gain comes from

Figure 5 and Table 5 compare configurations trained from the same checkpoint on the same mixture and with the same seed, the one on which RLI scored best. With base-block protection retained, tying the output projections under the aligned objective increases the score from 39.42 (C) to 41.51 (G). Replacing the aligned objective with the cross-read maximum increases it to 43.41 (F). These matched-seed differences are $+2.09$ and $+1.90$; against RLI's three-seed mean the second is $+0.91$, about one seed standard deviation. The corresponding within-model contributions of the reads are $+1.16$, $+3.13$ and $+4.47$.

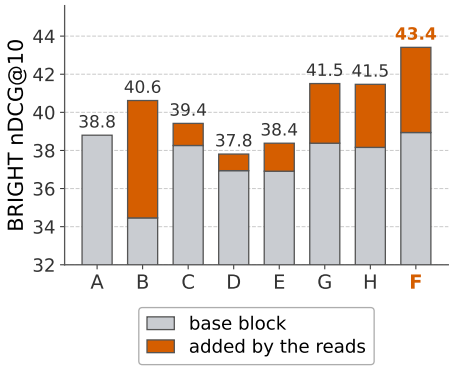


Figure 5: Base-block score and the amount the reads add. **A** no reads; **B**, **C** aligned loss, base unprotected and protected; **D** fixed pooling; **E** maximum, untied W_o; **G** aligned, tied; **H** maximum and tied, reads not recurrent; **F** RLI. Full numbers in Table 5.

Configuration H replaces the recurrent carry with a fixed initial state and learned per-step attention offsets (Appendix A). It scores 41.46, 1.95 below F at the same seed and 0.96 below RLI's three-seed mean. Configuration E retains untied output projections under the maximum objective and scores 38.37. Its off-diagonal argmax rate is 0.00% at all monitored training steps. Consequently, $S_{\max} = S_{\text{dot}}$ on the monitored positive pairs. We did not measure this equality on negative pairs, so this diagnostic does not establish equivalence of the complete training losses. The fixed-pooling control scores 37.81, with its pooled components adding $+0.87$ over its own base block, compared with $+4.47$ for F. "Reads add" measures the difference between the packed-vector score and the base-block score of the same trained model. It measures reliance on the reads, whereas the training ablations assess the effects of changing the model and objective. The three-seed comparison against read-free training is reported in §4.2.

4.5 Design choices at search time

Table 9 scores the same trained model with every aggregator and read weight from one encoding pass. The cross-read maximum, the objective the model was trained through, is the *worse* inference scorer at every weight, by 2.89 nDCG@10 at the deployed $\kappa = 2$ (43.41 against 40.52); we had

Table 3: **BRIGHT long documents** (Recall@1, eight splits of unsplit web pages). RLI-4B is one run at the passage-setting κ. Best in bold. †Reasons at query time.

Model	Avg.	Bio.	Earth	Econ.	Psy.	Rob.	Stack	Sus.	Pony
BM25	11.4	10.7	15.4	10.7	8.4	7.4	22.2	10.7	5.4
gte-Qwen1.5-7B-instruct	27.8	39.2	36.1	25.7	42.3	21.3	**23.5**	33.1	1.3
inf-retriever-v1-pro	30.5	44.1	42.2	31.4	43.1	20.8	21.4	41.0	0.4
MRE-T1†	35.1	46.5	**46.0**	34.5	52.7	27.7	22.2	45.2	6.3
RLI (ours)	**37.1**	**50.9**	40.4	**39.6**	**55.9**	**28.2**	19.7	**51.0**	**10.7**

expected the opposite. The two differ in which document read each query read meets: the dot product keeps the aligned pairing, while the maximum takes the best available document read and may reuse one for several query reads, so a passage weak on its aligned match can repair it elsewhere. Under the diagnostic read weights used in Figure 6, this pattern occurs on 17% of queries, compared with 6% for the opposite pattern. These diagnostic weights differ from those used by the deployed scorer (Appendix E).

Figure 4(b) varies the number of reads T while keeping the remaining training settings fixed. The output projection W_o is shared across reads; increasing T increases both the stored vector width and the number of recurrent steps. The score at $T = 3$ exceeds that at $T = 4$ by 1.37 points and that at $T = 6$ by 2.36 points. A paired bootstrap over the $1{,}384$ queries does not distinguish $T = 3$ from $T = 2$ ($+0.04$, 95% CI $[-0.49, +0.57]$), while $T = 2$ stores 7.7% fewer floats per document. For $T > 3$, retrieval performance decreases while the end-of-training off-diagonal rate approaches the uniform-assignment rate. At $T = 1$ there is no cross-read matching; this configuration scores 42.46, within seed variance of the three-read mean (42.42 ± 0.88), so most of the gain over read-free training comes from one learned read. Slot locking decides whether further reads help: three untied reads trained through the aligned score fall to 39.42 (C), while under RLI three reads reach 43.41 at the same seed. Despite a mean cosine of 0.88, the reads are not copies: permuting them removes the gold from the top ten in 23.4% of cases (Table 10).

4.6 Scale, backbone and document length

Figure 4(c) and Table 16 report the backbones. Qwen3-0.6B reaches 35.21 without the MS MARCO warm-up and 37.33 with it. An instruction-tuned 3B Granite model (IBM Research, 2026) with no embedding pre-training reaches 39.90 plain and 41.84 warmed, and Llama-3.2-3B, the base of the strongest published retriever, reaches 38.19 from the recipe alone against that model's 38.3, which adds embedding pre-training and a larger synthetic set over the same corpora. On our own backbone the warm-up is worth 1.1. In every case the reads contribute between $+3.9$ and $+5.2$ over the base block of the same run; these are within-model removal effects, since we did not train matched read-free recipes on the other three backbones, so they establish that the model depends on the reads across backbones rather than a gain of that size over matched baseline training. Across the four tested backbones, the end-of-training off-diagonal argmax rate lies between 54% and 65%, and the aligned dot product outperforms the cross-read maximum at inference. BRIGHT also provides a long-document setting for its eight StackExchange and Pony splits, where the corpus is unsplit web pages and the metric is Recall@1 (Su et al., 2025). Evaluated there with nothing changed, RLI-4B reaches 37.1, above every single-model entry, including MRE-T1 (35.1), which reasons at query time (Table 3).

5 Conclusion

We studied slot locking in dense retrievers with several pooled components, separating the aligned score's lack of cross-read terms from the diagonal matching of untied projections. Two changes remove it, and they are not symmetric. Tying the read projections places the reads in one shared space: still trained through an inner product, a tenth of comparisons bind off the diagonal and BRIGHT nDCG@10 rises from 39.4 to 41.5. Late-interaction supervision over a recurrent head needs that space, and adds $+1.9$ at a matched seed ($+0.9$ against the three-seed mean of 42.4); fixed pooling on the same budget adds $+0.9$ to its own base block. One learned read already reaches 42.5; removing slot locking keeps up to three reads from degrading it.

USE OF LARGE LANGUAGE MODELS

We used generative AI tools as coding assistants for the experimental pipeline, including data loading and evaluation scripts, and to improve the paper's readability. We also used them to review the clarity and consistency of methodological descriptions and interpretations of results. The authors reviewed and tested all AI-assisted code, verified the resulting text, and take full responsibility for the paper's content.

REFERENCES

BAAI. Bge-reasoner. https://github.com/VectorSpaceLab/agentic-search/tree/main/BGE-Reasoner, 2025.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

ByteDance Seed. Seed1.5-embedding. https://huggingface.co/ByteDance-Seed/Seed1.5-Embedding, 2025.

Jianlyu Chen, Junwei Lan, Chaofan Li, Defu Lian, and Zheng Liu. Reasonembed: Enhanced text embeddings for reasoning-intensive document retrieval. pp. 1203–1221, 2026.

Kyunghyun Cho, Bart Van Merriënboer, Çağlar Gulçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1724–1734, 2014.

Debrup Das, Sam O'Nuallain, and Razieh Rahimi. Rader: Reasoning-aware dense retrieval models. pp. 19981–20008, 2025.

Laxman Dhulipala, Majid Hadian, Rajesh Jayaram, Jason Lee, and Vahab Mirrokni. Muvera: Multi-vector retrieval via fixed dimensional encoding. *Advances in Neural Information Processing Systems*, 37:101042–101073, 2024.

Forward AI Labs. Mira reasoning retrieval. https://wanggz.github.io/mira-reasoning-retrieval, 2026a.

Forward AI Labs. Mre-t1. https://huggingface.co/ForwardAILabs/MRE-T1, 2026b.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. 2021.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. 2019.

IBM Research. Granite 4.1 language models. https://huggingface.co/collections/ibm-granite/granite-41-language-models, 2026. Accessed: 2026-07-25.

INF. Inf-x-retriever. https://yaoyichen.github.io/INF-X-Retriever/, 2025.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*, 2021.

Jiajie Jin, Yanzhao Zhang, Mingxin Li, Dingkun Long, Pengjun Xie, Yutao Zhu, and Zhicheng Dou. Laser: Internalizing explicit reasoning into latent space for dense retrieval. *arXiv preprint arXiv:2603.01425*, 2026.

Vladimir Karpukhin, Barlas Oguz, 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.

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 39–48, 2020.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pp. 3744–3753. PMLR, 2019.

LightOn AI. Reason-ModernColBERT. `https://huggingface.co/lightonai/Reason-ModernColBERT`, 2025.

Sheng-Chieh Lin, Jheng-Hong Yang, and Jimmy Lin. Distilling dense representations for ranking using tightly-coupled teachers. *arXiv preprint arXiv:2010.11386*, 2020.

Wenhan Liu, Xinyu Ma, Weiwei Sun, Yutao Zhu, Yuchen Li, Dawei Yin, and Zhicheng Dou. Reasonrank: Empowering passage ranking with strong reasoning ability. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22007–22031, 2026.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. volume 33, pp. 11525–11538, 2020.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 2014–2037, 2023.

NVIDIA. Nemo retriever agentic retrieval on bright. `https://github.com/NVIDIA/NeMo-Retriever/blob/main/retrieval-bench/submissions/bright_agentic.md`, 2026a.

NVIDIA. llama-nv-embed-reasoning-3b. `https://huggingface.co/nvidia/llama-nv-embed-reasoning-3b`, 2026b.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

RakanLabs. Rakanembed4b. `https://rakanlabs.com/work/rakanembed`, 2026.

Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. Plaid: an efficient engine for late interaction retrieval. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 1747–1756, 2022a.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3715–3734, 2022b.

Rulin Shao, Rui Qiao, Varsha Kishore, Niklas Muennighoff, Xi Victoria Lin, Daniela Rus, Bryan Kian Hsiang Low, Sewon Min, Wen-tau Yih, Pang Wei Koh, et al. Reasonir: Training retrievers for reasoning tasks. *arXiv preprint arXiv:2504.20595*, 2025.

Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Liu Haisu, Quan Shi, Zachary Siegel, Michael Tang, et al. Bright: A realistic and challenging benchmark for reasoning-intensive retrieval. In *International Conference on Learning Representations*, volume 2025, pp. 48941–48991, 2025.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. volume 28, 2015.

Duolin Sun, Meixiu Long, Dan Yang, Junjie Wang, Yecheng Luo, Yue Shen, Jian Wang, Hualei Zhou, Chunxiao Guo, Peng Wei, et al. Diver: A multi-stage approach for reasoning-intensive information retrieval. *arXiv preprint arXiv:2508.07995*, 2025.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. 2021.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. 2015.

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.

Zilin Xiao, Qi Ma, Mengting Gu, Chun-cheng Chen, Xintao Chen, Vicente Ordonez, and Vijai Mohan. Metaembed: Scaling multimodal retrieval at test-time with flexible late interaction. In *International Conference on Learning Representations*, volume 2026, pp. 23619–23638, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Zhang, Yaobo Liang, Ming Gong, Daxin Jiang, and Nan Duan. Multi-view document representation learning for open-domain dense retrieval. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5990–6000, 2022.

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

A Training and evaluation details

Training mixture. We train on two public releases, Chen et al. (2026) and Liu et al. (2026), and the same mixture is shared byte-for-byte by every configuration in the paper. Each record carries a query, its offline chain-of-thought rewrite (used only by the reasoning-intensity weight w_q and the anchor loss, never at inference), one positive and a pool of hard negatives from which $G-1$ are drawn, and the batch size and group size the record is trained with, following the per-domain settings that ship with the ReasonEmbed release. Batches are formed within one file ("same dataset within batch"), so in-batch negatives come from the same file; for the twelve per-domain files this is also the same domain, while the pooled file spans all twelve.

Mixture composition and domain oversampling. A mixed file pooling ReasonEmbed examples across all twelve domains accounts for 46% of the raw training rows. We oversample the LeetCode, AoPS and Pony files by factors of 12, 4 and 4, respectively. The table below reports each file's fraction of training rows before and after oversampling.

	Copies	Raw share	Effective share
Shares are fractions of training rows.			
LeetCode	$\times 12$	4.9%	34.2%
AoPS	$\times 4$	4.2%	9.7%
Pony	$\times 4$	1.9%	4.3%
all other domains	$\times 1$	–	–

Within this mixture the reads are what lift LeetCode further, from 15.65 for the read-free recipe to 29.34 (Table 1); Pony's strength, by contrast, is inherited from the mixture rather than from the reads, which is why §4.2 attributes it there.

Corpus exposure. ReasonEmbed's half of the mixture is synthesised over the BRIGHT corpora with the benchmark's labelled passages removed, so our encoder has read unlabelled text from the evaluation corpora. This is not an advantage over the models we compare against: the three ReasonEmbed entries in Table 1 are trained on that same release, and llama-nv-embed-reasoning-3b (NVIDIA, 2026b) is trained on ReasonEmbed, ReasonAug and ReasonRank together with a further synthetic set generated from the same BRIGHT corpora. No configuration is trained on the relevance judgements. The deployed model has a single evaluation-time constant, the global scalar κ, selected on held-out ReasonIR queries rather than on BRIGHT (§4.1); Table 9 reports every value. The comparison that isolates the reads remains the one against our own read-free recipe, which shares the mixture, backbone and schedule.

Sources for the published baselines. The per-split figures of Tables 1 and 6 are those published by each source, as collated in the llama-nv-embed-reasoning-3b model card (NVIDIA, 2026b), except Contriever, OpenAI, Voyage, Google and GritLM-7B, which appear only in Table 6 and are as reported by Shao et al. (2025) under the same protocol. For DIVER-Retriever we use the release that reports per split; its later revision reports 31.9 on average without a per-split breakdown (Sun et al., 2025). BGE-Reasoner-Embed is from Chen et al. (2026); the pipeline systems of Table 8 are BGE-Reasoner (BAAI, 2025), DIVER (Sun et al., 2025), NeMo agentic retrieval (NVIDIA, 2026a), RakanEmbed4B (RakanLabs, 2026), INF-X-Retriever (INF, 2025), MRE-T1 (Forward AI Labs, 2026b) and Mira (Forward AI Labs, 2026a). The field spans BM25, general-purpose embedding models (Zhang et al., 2025), the late-interaction Reason-ModernColBERT (LightOn AI, 2025), and retrievers trained for reasoning (Shao et al., 2025; Das et al., 2025; Sun et al., 2025; Chen et al., 2026).

The auxiliary objectives. Writing $\bar{e}$ for the normalised base block, $\nu_t = s_t \bar{v}_t$ for read t, α_t for its attention distribution over the L token states, and $\mathcal{B}$ for the batch, the five auxiliary terms of §3.3 are as follows. *Base-block retention* is the same InfoNCE loss with $S_{\max}$ of equation 4 replaced by $\langle \bar{e}_q, \bar{e}_p \rangle$, so the base block is trained as a standalone retriever. *Reasoning-anchor distillation* uses a two-layer head g ($2560 \to 4096 \to 2560$) and

$$\mathcal{L}_{\text{anchor}} = \sum_{q \in \mathcal{B}} \Big(1 - \langle \widehat{a}_q,\, \mathrm{sg}[\bar{e}_{q'}] \rangle + \big\| \widehat{a}_q - \mathrm{sg}[\bar{e}_{q'}] \big\|_2^2 \Big), \qquad \widehat{a}_q = \overline{\bar{e}_q + g(\bar{e}_q)}, \tag{5}$$

where q' is the offline chain-of-thought rewrite of q, encoded by the same model, and $\mathrm{sg}[\cdot]$ is a stop-gradient, so the rewrite branch contributes no gradient to the backbone and g is discarded at inference. *Lexical fidelity* matches the distribution induced by the first read over a training group $\mathcal{P}(q)$ to a token-overlap teacher ρ,

$$\mathcal{L}_{\text{lex}} = \sum_{q \in \mathcal{B}} \mathrm{KL}\Big(\mathrm{softmax}_{p \in \mathcal{P}(q)} \big(\rho(q,p)/\tau_{\text{lex}} \big) \;\Big\|\; \mathrm{softmax}_{p \in \mathcal{P}(q)} \big(\langle \nu_1^q, \nu_1^p \rangle / \tau_{\text{lex}} \big) \Big). \tag{6}$$

Read diversity penalises overlap between consecutive query reads under an entropy floor,

$$\mathcal{L}_{\text{div}} = \sum_{q \in \mathcal{B}} \Big(\sum_{t=1}^{T-1} \sum_{i=1}^{L} \alpha_{t,i}\, \alpha_{t+1,i} + \sum_{t=1}^{T} \max\big(0,\; \log 8 - \mathcal{H}(\alpha_t)\big) \Big). \tag{7}$$

Usage balance keeps the soft assignment of query reads to document reads spread over the T document reads on positive pairs, with $A_{tu} = \mathrm{softmax}_u\big(\langle \nu_t^q, \nu_u^{p^+} \rangle / \tau_{\text{ub}} \big)$ and $\bar{A}_u = \frac{1}{T} \sum_t A_{tu}$,

$$\mathcal{L}_{\text{bal}} = \sum_{q \in \mathcal{B}} \Big(\log T - \mathcal{H}(\bar{A}) \Big). \tag{8}$$

Setting $s = 0$ removes the read blocks from $\phi(x)$ and makes $S_{\max} = S_{\text{dot}} = \langle \bar{e}_q, \bar{e}_p \rangle$; the read-free baseline is trained with $\mathcal{L}_{\text{lex}}$, $\mathcal{L}_{\text{div}}$ and $\mathcal{L}_{\text{bal}}$ removed, since each is defined on the reads, and retains $\mathcal{L}_{\text{base}}$ and $\mathcal{L}_{\text{anchor}}$.

Configuration H: removing the recurrent carry. H replaces z_{t-1} by z_0 in both places it appears in equations 1–2, and adds a learned per-step offset $Q_t \in \mathbb{R}^m$ to the attention query:

$$a_{t,i}^{\text{H}} = \frac{\langle \tilde{h}_i,\; W_z z_0 + Q_t \rangle}{\sqrt{m}} + \log \mu_i, \qquad \alpha_t = \mathrm{softmax}(a_t^{\text{H}}), \tag{9}$$

$$r_t = \sum\nolimits_i \alpha_{t,i}\, \tilde{h}_i, \quad z_t = \mathrm{GRU}(r_t,\, z_0), \qquad v_t = W_o\, z_t \in \mathbb{R}^c. \tag{10}$$

Everything else is held: the same W_p, W_z, GRU cell and W_o, with W_o shared across steps as in RLI, the same learned scales s, the same objective of equation 4, the same five auxiliary terms and the same stored width. Read t therefore no longer sees what reads 1 through $t-1$ retrieved, which is the single quantity this configuration is meant to remove.

The offset Q_t is what makes this configuration a control rather than a degenerate one. A plain state reset, $q_t = W_z z_0$ for every t, would give each step the same input to the same deterministic map, so $\alpha_1 = \cdots = \alpha_T$ and $v_1 = \cdots = v_T$; with W_o shared the configuration would collapse to $T = 1$, removing the capacity to form distinct reads at the same time as the carry and confounding the two. The learned offsets Q_t allow the attention queries to differ across steps. Configuration H therefore removes the recurrent carry without forcing all reads to be identical. Q_t costs $Tm = 1{,}536$ parameters against the head's 3.28M, an increase of 0.05%, so H is parameter-matched to RLI for practical purposes.

Hyperparameters. Table 4 lists the main training hyperparameters. The ablations vary the presence and construction of the reads, base-block protection, the scoring objective, output-projection sharing and recurrent carry, as specified in Table 5 and Appendix A. Figure 4(b) separately varies the read count T.

Table 4: **Hyperparameters.**

Backbone	Qwen3-4B, MS MARCO fine-tuned, causal, last-token pooling, $d = 2560$
Adapters	LoRA $r = 32$, $\alpha = 64$, on q/k/v/o and gate/up/down projections
Read head	$m = 512$, $c = 256$, $T = 3$, s_t initialised at 0.30, W_z initialised to I
Head parameters	3.28M (tied W_o); anchor head g: 21.0M, discarded at inference
Objective weights	$\lambda_b = 1$, $\lambda_a = 1$, $\lambda_x = 0.5$, $\lambda_d = 0.1$, $\lambda_u = 0.05$
Temperatures	$\tau = 0.02$ (contrastive); 0.05 (lexical teacher and usage balance)
Entropy floor	$\log 8$ per read
Optimisation	AdamW, lr 10^{-4}, warm-up 10%, one epoch, 4,521 steps, bf16, DeepSpeed stage 1
Batch	per-device 8 queries (LeetCode 16), $G = 16$ (LeetCode 2), negatives shared across 8 devices
Sequence length	512 tokens for queries, rewrites and passages at training; 8192 at evaluation
Instruction format	`Instruct: {task}\nQuery: {query}` on the query side
Hardware	2 nodes $\times$ 4 H100 80GB; about 13 hours per run
Seeds	42, 43, 44 (data order and head initialisation)

B Full baseline and ablation tables

Table 6 is the passage-level comparison of Table 1 with the six additional published rows restored and the rows grouped by system type, and Table 7 does the same for the long-document setting of Table 3. Table 8 lists the pipeline systems discussed in Appendix C. Table 5 gives the configurations of Figure 5 as numbers.

C Pipeline systems on BRIGHT, and why they are not our baselines

Table 8 reports selected published systems that use query-time generation, reranking or multiple searches. These systems follow a different inference protocol from the retriever-only comparison in Table 1. We include RLI in Table 8 as a reference. MRE-T1 generates a reasoning chain before embedding the query, so its 39.6 score is reported here.

D Seed-level statistics

Statistics. Welch's t-test on the seed-level averages gives, for RLI against the dense baseline, $t = 4.55$ with 3.9 degrees of freedom, $p = 0.011$, and a 95% confidence interval of $[+1.38, +5.86]$

Table 5: **Effect of the training objective**, the numbers behind Figure 5. "Base" is that run's base embedding scored alone; "reads add" is how much the packed vector improves on it. Rows A and B are three-seed means (Appendix D); every other row is a single run with the seed of the best RLI run, so gaps to F can overstate the seed-mean gap by up to 0.99 (F's mean is 42.42 ± 0.88).

	Configuration	Base	Reads add	nDCG@10
A	no reads (dense baseline)	38.80	–	38.80
B	aligned loss, base unprotected	34.46	+6.16	40.62
C	aligned loss, base protected	38.26	+1.16	39.42
D	fixed pooling (max/mean/attention)	36.94	+0.87	37.81
E	maximum objective, untied W_o	36.91	+1.47	38.37
G	aligned loss, tied W_o	38.38	+3.13	41.51
H	maximum objective, tied W_o, reads not recurrent	38.16	+3.31	41.46
F	RLI, maximum objective, tied W_o	38.94	**+4.47**	**43.41**

Table 6: **BRIGHT, per split, retriever-only on original queries**, all baselines, grouped by system type. The main-text Table 1 omits the six additional published rows for space; every other entry, and every number, is identical. Contriever, OpenAI, Voyage, Google and GritLM-7B are as reported by Shao et al. (2025).

	StackExchange							Coding		Theorem-based			
Model	**Bio.**	**Earth.**	**Econ.**	**Psy.**	**Rob.**	**Stack.**	**Sus.**	**Leet.**	**Pony**	**AoPS**	**TheoQ.**	**TheoT.**	**Avg.**
Lexical													
BM25	18.9	27.2	14.9	12.5	13.6	18.4	15.0	24.4	7.9	6.2	10.4	4.9	14.5
General-purpose embedding models													
Contriever	9.2	13.6	10.5	12.1	9.5	9.6	8.9	24.5	14.7	7.2	10.4	3.2	11.1
OpenAI	23.7	26.3	20.0	27.5	12.9	12.5	20.3	23.6	2.5	8.5	23.8	12.3	17.8
Voyage	23.6	25.1	19.8	24.8	11.2	15.0	15.6	30.6	1.5	7.4	26.1	11.1	17.7
Google	23.0	34.4	19.5	27.9	16.0	17.9	17.3	29.6	3.6	9.3	21.5	14.3	19.5
GritLM-7B	25.0	32.8	19.0	19.9	17.3	11.6	18.0	29.8	22.0	8.8	25.1	21.1	20.9
Qwen3-Embedding-4B	17.8	34.7	16.9	23.3	12.5	16.2	16.8	35.7	1.4	9.8	35.5	41.5	21.8
Qwen3-Embedding-8B	21.0	33.0	18.4	26.1	15.7	19.4	17.3	33.8	1.2	9.4	39.2	39.3	22.8
llama-nemotron-embed-3b-v2	31.1	36.7	22.3	28.4	18.0	18.4	20.3	32.1	6.8	12.1	25.1	16.5	22.3
Late interaction													
Reason-ModernColBERT (150M)	33.3	41.0	24.9	30.7	21.1	20.6	20.3	31.1	8.5	9.2	19.5	11.2	22.6
Retrievers trained for reasoning													
ReasonIR-8B	26.2	31.4	23.3	30.0	18.0	23.9	20.5	35.0	10.5	14.7	31.9	27.2	24.4
RaDeR-gte-Qwen2-7B	34.6	38.9	22.1	33.0	14.8	22.5	23.7	37.3	5.0	10.2	28.4	35.1	25.5
Seed-1.5-Embedding	34.8	46.9	23.4	31.6	19.1	25.4	21.0	**43.2**	4.9	12.2	33.3	30.5	27.2
DIVER-Retriever	41.8	43.7	21.7	35.3	21.0	21.2	25.1	37.6	13.2	10.7	38.4	37.3	28.9
later revision	–	–	–	–	–	–	–	–	–	–	–	–	31.9
ReasonEmbed-Llama-3.1-8B	55.4	56.2	35.2	48.5	32.1	37.3	41.1	28.8	16.8	9.1	37.9	36.6	36.2
BGE-Reasoner-Embed-8B	–	–	–	–	–	–	–	–	–	–	–	–	37.1
ReasonEmbed-Qwen3-4B	55.4	54.5	34.9	46.9	34.0	36.1	37.4	34.5	13.6	11.3	**41.4**	45.1	37.1
ReasonEmbed-Qwen3-8B	55.5	56.6	36.2	47.4	**35.3**	36.6	39.1	33.6	16.4	12.5	**41.4**	47.2	38.1
llama-nv-embed-reasoning-3b	63.4	60.2	**39.5**	45.5	32.6	34.0	**43.3**	37.5	15.0	10.5	39.5	38.5	38.3
Ours													
RLI	**70.30**	**64.26**	38.51	**49.03**	33.93	**41.55**	41.18	29.34	**45.47**	**19.04**	39.07	**49.25**	**43.41**
RLI, 3 seeds	68.54	63.06	36.92	47.43	34.27	40.92	40.18	27.78	43.51	19.22	38.56	48.61	42.42 ± 0.88
our recipe, no reads (one run)	63.02	59.33	35.15	40.55	28.03	36.64	37.97	15.65	47.74	13.65	40.24	47.23	38.77

on the difference of +3.62; for RLI against the aligned-objective, base-unprotected configuration (40.62 ± 0.91), $t = 2.46$, $\mathrm{df} = 4.0$, $p = 0.070$, CI $[-0.23, +3.83]$ on +1.80.

E THE INFERENCE SCORER: WHY THE DOT PRODUCT WINS, AND SERVING FORMS

This appendix expands the scorer comparison of §4.5.

Why the maximum loses. The inference scorer is the dot product between the packed query and document vectors. It combines the base similarity with the weighted aligned read similarities, with the normalisation described in equation 3. The cross-read maximum instead lets each query read match its best document read. Figure 6 illustrates cases in which the dot product ranks a gold passage above the distractor preferred by the cross-read maximum. Its read-level diagnostics

Table 7: **BRIGHT long documents** (Recall@1), selected single-model entries from the benchmark leaderboard (Su et al., 2025). †Reasons at query time.

Model	Avg.	Bio.	Earth	Econ.	Psy.	Rob.	Stack	Sus.	Pony
BM25	11.4	10.7	15.4	10.7	8.4	7.4	22.2	10.7	5.4
text-embedding-3-large	21.3	32.1	31.4	23.8	34.2	11.9	10.7	26.3	0.0
voyage-large-2-instruct	24.5	34.4	35.4	26.7	41.6	12.9	12.8	31.1	1.3
e5-mistral-7b-instruct	25.5	29.9	36.3	26.2	46.7	17.3	14.5	32.2	1.1
GritLM-7B	26.0	37.5	40.3	25.7	34.4	17.8	20.1	32.4	0.0
SFR-Embedding-Mistral	26.0	30.3	37.0	24.3	47.7	17.3	14.5	35.0	2.0
gte-Qwen1.5-7B-instruct	27.8	39.2	36.1	25.7	42.3	21.3	**23.5**	33.1	1.3
inf-retriever-v1-pro	30.5	44.1	42.2	31.4	43.1	20.8	21.4	41.0	0.4
Gecko text-embedding-004	33.2	–	–	–	–	–	–	–	–
MRE-T1†	35.1	46.5	**46.0**	34.5	52.7	27.7	22.2	45.2	6.3
RLI (ours)	**37.1**	**50.9**	40.4	**39.6**	**55.9**	**28.2**	19.7	**51.0**	**10.7**

Table 8: Pipeline results on BRIGHT, for context only. "Retriever-only" is the setting of Table 1. Figures are as reported by each system on the benchmark leaderboard (Su et al., 2025).

System	Components beyond a retriever	BRIGHT avg
RLI (ours)	none (retriever only)	43.41
BGE-Reasoner-0928	retriever + LLM query expansion + reranker	46.4
DIVER	retriever + LLM query expansion + reranker	46.8
NeMo	llama-nv-embed-reasoning-3b + LLM agent + fusion	50.9
RakanEmbed4B	LLM query rewriter + 4B embedder	52.4
INF-X-Retriever	LLM query aligner + 7B retriever	63.4
MRE-T1	query-time reasoning chain, then one embedder	39.6
Mira	RL-trained query rewriter + MRE-T1	66.9

weight the three reads at 0.45, 0.10 and 0.20 rather than by the deployed $(2s_t)^2 = 0.29$, 0.29, 0.30, and the gold and distractor of each pair are the ones that weighting ranks highest; Table 9 has the dot product ahead of the maximum at every κ. The reads retain partial dependence on slot alignment: deliberately mis-aligning the reads costs 0.5 to 1.5 points, and on each query's top-100 a query read fires on its aligned document read 65% of the time (chance 33%), so the reads are partly interchangeable aspects rather than three indexed slots. The maximum loses because it lets a passage repair a weak aligned match by borrowing a different document read. For 17% of queries, the passage ranked highest by the maximum is stronger than the gold on one read pair but weaker across the reads (Figure 6; the opposite pattern occurs for 6%). On 142 queries, that passage beats the gold under the maximum but loses under the dot product, compared with 46 queries showing the opposite pattern. In a typical query of this kind, the gold passage is moderately close on all three reads while the distractor is very close on one and far on another, and the maximum repairs the far read by borrowing a different document read (Figure 7a). The dot product is also the more stable operator: under seven edits of 60 queries the gold stays in the top-10 95% of the time against 89% (Figure 7b), under Gaussian noise on the query vectors the top-10 overlaps the clean ranking more (0.84 against 0.81 at cosine 0.90), and the maximum's read-to-read matching itself changes in 14% of the edits. One split prefers the maximum, TheoremQA-Questions ($+3.8$), whose gold pool is 409 passages of one source style. Training through the maximum shapes the reads; serving through the dot product scores them. A direct test of what the dot product uses is to permute a document's read slots at search time, which the rest of this appendix reports in full. Appendix G shows, on real BRIGHT queries, what the reads change relative to the base vector alone.

Slot identity at search time. The read contribution to the inference score combines the weighted aligned similarities, so it uses *which* slot a read sits in; the cross-read maximum takes the best document read for each query read and is invariant to the order of a document's slots by construction.

Table 9: **Inference scorer and operating point** (nDCG@10, one encoding pass per run; κ scales the read blocks of query and document; the configuration we serve, selected on held-out ReasonIR queries, is in bold). The same numbers are plotted in the right panel of Figure 1; the number of reads T is swept in Figure 4(b).

Aggregator	$\kappa = 1$	$\kappa = 2$	$\kappa = 4$
Base block only (no reads)		38.94	
Concatenated dot product	40.89	**43.41**	39.40
Cross-read maximum	39.93	40.52	38.32
ReasonIR dev, dot product (1k queries)	20.51	23.60	23.10

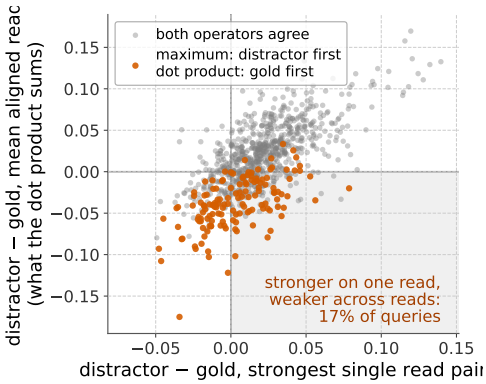


Figure 6: Every BRIGHT query: the gold passage against the passage the maximum ranks highest for it. Horizontal axis, the strongest single read pair, a diagnostic for the maximum, whose scorer sums each query read's best document read; vertical axis, the mean aligned read similarity, which the dot product sums.

For every BRIGHT query we take the gold passage ranked highest by the dot product together with its strongest distractor, compute the full 3×3 matrix of read similarities $\langle q_t, d_{t'} \rangle$, and permute that document's three read slots in all five non-identity orders, at search time and for that document only, re-scoring it against an unchanged corpus. Table 10 reports the effect over the benchmark and Table 11 per split. Permuting a document's slots costs it about 0.032 in score, the size of a typical gold–distractor margin, so the gold leaves the top ten in roughly a quarter of the cases where it was there and loses rank one in more than half of the cases where it held it. Permuting the *query's* slots barely moves anything, because it changes the comparison against every document in the corpus, where permuting one document's slots changes only that document's score (its mean gold rank even drifts slightly down, 172 against 177, while nDCG@10 falls by 0.5 to 1.5, the two measures weighting the tail differently): what the dot product uses is the relative alignment between a query and each document, not absolute slot labels. The maximum ranks the gold slightly worse than the dot product before any manipulation (mean rank 182 against 177) and is blind to it after. The effect is largest where the benchmark is hardest, AoPS, Stack Overflow, LeetCode and Robotics, and smallest where the gold is already unambiguous, Pony and Biology. The reads are therefore only *partly* slot-specific: the mildest permutation, the swap of reads 2 and 3, costs little on every split, and permuting the query's slots benchmark-wide costs only 0.5 to 1.5 points, so strictly indexed slots would overstate what these numbers show.

The read matrices behind six of those queries. Table 12 gives the full 3×3 matrices and Table 13 what the permutation then does to the gold's rank. The diagonal is the largest entry of its row in fifteen of these eighteen rows, and in the other three it trails the largest by at most 0.008: query read t is typically most similar to document read t, because both come from corresponding recurrent steps of a head whose parameters are shared between the two sides. The dot product sums that diagonal and the shuffle destroys it, while the maximum takes the row maxima, which the shuffle

Table 10: **Permuting a document's read slots at search time** (RLI-4B, packed dot product). Each statistic is computed per split over BRIGHT's 1,384 queries and then averaged over the twelve splits. Percentages over the five non-identity orders unless stated.

Statistic (gold document, dot product)	**Value**
Gold at rank 1, slots in order	40.7%
Gold at rank 1 after permuting its slots	16.7%
Gold leaves the top ten after permuting its slots	23.4%
worst order	33.1%
mildest order (swap of reads 2 and 3)	8.2%
Mean rank of the gold, slots in order	177
document slots permuted	267
query slots permuted	172
under the cross-read maximum	182
Mean score drop of the gold when its slots are permuted	0.032

Table 11: **Permuting a document's read slots, per split.** "Gold in top 10" counts the queries whose gold passage is in the top ten with slots in order, which is the population the next three columns are taken over.

	gold in	% leaving the top ten			mean gold rank		
split	top 10	mean	worst	mildest	in order	permuted	maximum
Biology	95	8%	11%	2%	10	21	14
Earth Science	109	15%	20%	8%	7	14	6
Economics	64	30%	45%	16%	84	164	99
Psychology	82	27%	40%	11%	46	80	60
Robotics	65	32%	48%	8%	70	159	80
Stack Overflow	81	35%	46%	11%	101	246	126
Sustainable Living	73	17%	22%	7%	62	100	57
LeetCode	80	34%	45%	11%	112	228	188
Pony	111	6%	12%	2%	2	5	5
AoPS	58	43%	60%	12%	158	373	213
TheoremQA-Q	98	11%	14%	4%	1429	1736	1269
TheoremQA-T	59	23%	34%	7%	42	74	66

leaves untouched. In these cases the distractor is the passage that matches strongly on a single view, with a best read of 0.85 to 0.87 against an aligned read as low as 0.60.

Training and inference use different aggregators. During training, $S_{\max}$ allows each query read to match any document read. At inference, S_{dot} combines the aligned similarities. The trained models achieve higher retrieval scores with S_{dot} across the four tested backbones (Table 16). This observation is consistent with maximum-based supervision producing reads that are useful under aligned aggregation, but it does not establish why the two aggregators differ.

Candidate recall of the any-match index. Table 14 gives the candidate-stage numbers per split. Indexing the base and the three reads as four separate entries and taking the union of their top-100 lists raises candidate recall on eleven of the twelve splits, by 3.47 on average, and the gain is concentrated where the reads carry the lexical channel: LeetCode +8.69 and AoPS +7.38. Pony is the one split where the single vector's top-100 is better. Two qualifications matter. The comparison is at equal K, not at equal cost: the union returns 286 candidates on average against the single vector's 100. Deepening both shows the trade: the single vector's top-500 reaches 89.4 from 500 candidates, any-match 91.6 from 1,358, so 2.2 further points of recall cost 2.7 times the candidates. And the extra recall does not become accuracy: ranking within the any-match candidates scores 43.42 against the packed vector's 43.41. At this operating point, the any-match index increases candidate recall and approximately preserves the packed-vector nDCG@10 after rescoring. This comparison evaluates candidate generation; it does not isolate the cause of the remaining retrieval errors.

Table 12: **Read matrices of six gold passages.** Rows are query reads, columns document reads. The diagonal, in bold, is what the dot product sums; the row maxima are what the training objective sums.

	Snakes and dizziness (Biology)				**Power sums of roots** (AoPS)		
	d_1	d_2	d_3		d_1	d_2	d_3
q_1	**0.792**	0.735	0.741	q_1	**0.749**	0.723	0.739
q_2	0.615	**0.750**	0.747	q_2	0.644	**0.871**	0.862
q_3	0.756	0.806	**0.831**	q_3	0.643	0.797	**0.800**
	Infant memory (Biology)				**Collinear points** (LeetCode)		
	d_1	d_2	d_3		d_1	d_2	d_3
q_1	**0.829**	0.776	0.835	q_1	**0.872**	0.808	0.817
q_2	0.641	**0.818**	0.743	q_2	0.740	**0.899**	0.799
q_3	0.773	0.822	**0.828**	q_3	0.777	0.854	**0.853**
	Economic nature of water (Economics)				**XOR of a sequence** (Pony)		
	d_1	d_2	d_3		d_1	d_2	d_3
q_1	**0.899**	0.870	0.894	q_1	**0.720**	0.679	0.728
q_2	0.802	**0.897**	0.835	q_2	0.680	**0.753**	0.705
q_3	0.820	0.857	**0.863**	q_3	0.809	0.817	**0.833**

Table 13: **Shuffling a gold passage's read slots at search time** (RLI-4B, packed dot product). Columns: the diagonal $\langle q_t, d_t \rangle$ of the gold's read matrix; its rank under the dot product and under the cross-read maximum with slots in order; and its rank under the dot product after each permutation of its three slots.

	gold, aligned reads			gold rank, slots in order		gold rank, dot product, slots permuted				
query	r_1	r_2	r_3	dot	max	1 3 2	2 1 3	2 3 1	3 1 2	3 2 1
snakes (Biology)	.79	.75	.83	2	2	2	7	3	8	2
water (Economics)	.90	.90	.86	1	1	1	2	2	2	1
infant memory (Biology)	.83	.82	.83	1	2	2	28	17	15	1
roots (AoPS)	.75	.87	.80	1	29	2	85	64	76	62
collinear points (LeetCode)	.87	.90	.85	1	4	3	27	39	25	10
XOR (Pony)	.72	.75	.83	1	17	6	7	7	6	3

Serving the reads as separate vectors. A natural alternative to the packed vector is the ColBERT serving pattern at read granularity (Santhanam et al., 2022a): index the base and the three reads of each document as separate entries, generate candidates from any query–document vector match, and rescore the candidates by the sum over query vectors of the maximum over document vectors. Table 15 evaluates this pipeline on RLI-4B, using exhaustive scoring as the reference for approximate candidate generation. The candidate stage is a real gain: the union of the per-vector top-100 lists raises candidate recall to 80.1%, against 76.6% for the packed vector and 72.0% for the base alone. Within the any-match candidates reported in Table 15, the packed dot product scores 43.42, compared with 40.57 for sum-of-max, a gap of 2.85 points. Exhaustive scoring gives a gap of 2.89 points (Table 9). We then trained a variant *for* that deployment, with the base projected into the reads' 256-dimensional space so that all four vectors share one space, the objective replaced by the full 4×4 maximum, and each document stored as four 256-dimensional vectors ($1{,}024$ floats, less than a third of the packed vector). Served as intended it scores 37.71; the same four vectors under the aligned sum score 38.48; its $2{,}560$-dimensional base alone scores 38.59, and its packed concatenation 41.53. Two things follow. The sum beats the maximum even inside one shared space and for a model trained through the maximum, so the result of Table 9 is not an artefact of the base and reads living in different spaces. Throughout training, the query's projected base vector selected the document's projected base vector. Thus, projecting all four vectors into one space did not eliminate the base vector's distinct matching role.

Table 14: **Candidate recall at $K = 100$** (RLI-4B, mean fraction of a query's gold passages contained in its top-K). "Base" is the base block alone, "single" the packed vector, "any-match" the union of the per-vector top-100 lists. "Cands." is the mean size of that union.

Split	Base	Single	Any-match	Δ	Cands.
Biology	95.47	94.96	**97.31**	+2.35	262
Earth Science	88.26	89.75	**92.61**	+2.86	274
Economics	72.04	74.53	**78.89**	+4.36	263
Psychology	80.81	81.60	**84.77**	+3.18	282
Robotics	65.43	73.72	**77.79**	+4.07	263
Stack Overflow	76.81	83.17	**84.39**	+1.21	261
Sustainable Living	79.76	81.93	**84.66**	+2.73	253
LeetCode	57.49	70.85	**79.53**	+8.69	320
Pony	55.55	**69.43**	67.18	−2.25	273
AoPS	38.40	47.24	**54.63**	+7.38	332
TheoremQA-Q	63.21	62.20	**67.21**	+5.01	329
TheoremQA-T	91.23	89.58	**91.67**	+2.08	324
Mean	72.04	76.58	**80.05**	+3.47	286

Table 15: **Serving forms** (BRIGHT nDCG@10, exhaustive search). "Any-match" indexes the base and the three reads as four entries per document and takes the union of the per-vector top-100 lists as candidates. The tiny-vector model is trained for that deployment (shared 256-d space, 4×4 maximum), data and recipe otherwise unchanged.

Model	Serving form	Floats / doc	nDCG@10
RLI-4B	packed vector, dot product	3,328	**43.41**
RLI-4B	sum-of-max over the blocks, exhaustive	–	40.52
RLI-4B	any-match candidates, then sum-of-max	3,328 as 4 entries	40.57
RLI-4B	any-match candidates, then dot product	3,328 as 4 entries	43.42
tiny-vector model	4×256, any-match, sum-of-max	1,024	37.71
tiny-vector model	the same four vectors, aligned sum	1024	38.48
tiny-vector model	base alone	2,560	38.59
tiny-vector model	packed concatenation	3,328	41.53

F SCALE AND BACKBONES

Table 16 gives the runs behind Figure 4(c) as numbers, including the off-diagonal argmax rate and the sum-of-max scorer that the figure omits.

Table 16: **The recipe on four backbones, with and without the MS MARCO warm-up** (BRIGHT nDCG@10, single runs, recipe and mixture unchanged); the numbers behind Figure 4(c). Base = the head-free embedding of the same run; off-diag. = end-of-training off-diagonal argmax rate; "max at search" = the sum-of-max scorer on the same vectors. RLI-4B is the row in bold.

Backbone	Warm-up	Base	Reads add	BRIGHT	Off-diag.	Max at search
Qwen3-0.6B	none	31.06	+4.15	35.21	64%	33.22
Qwen3-0.6B	MS MARCO	32.12	+5.21	37.33	64%	33.97
Granite-4.1-3B	none	36.05	+3.85	39.90	65%	37.52
Granite-4.1-3B	MS MARCO	37.69	+4.15	41.84	65%	39.27
Llama-3.2-3B	none	34.06	+4.13	38.19	63%	35.52
Llama-3.2-3B	MS MARCO	35.31	+4.52	39.83	64%	36.49
Qwen3-4B	none	38.28	+4.01	42.28	65%	39.25
Qwen3-4B	MS MARCO	**38.94**	**+4.47**	**43.41**	54%	40.52

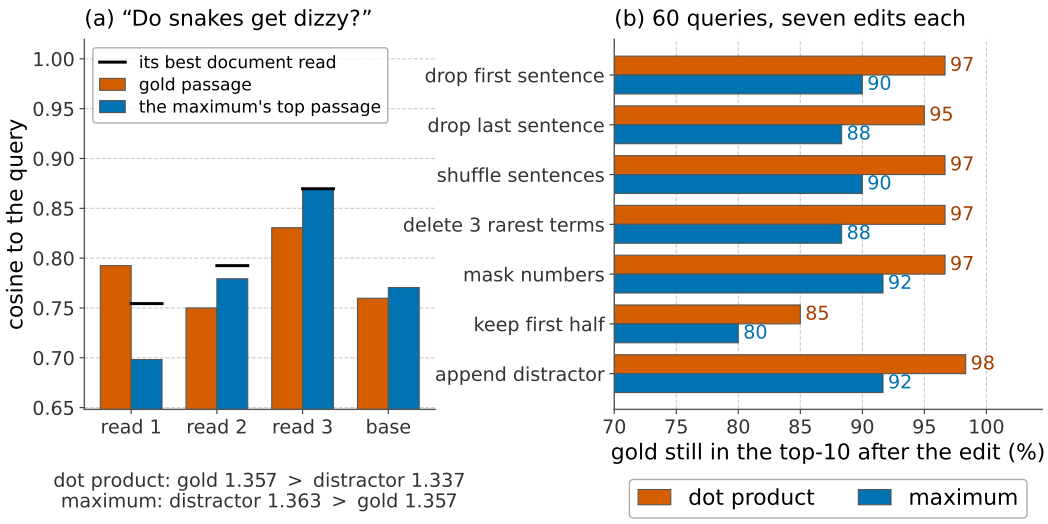


Figure 7: **(a)** One query: the three aligned read similarities and the base similarity of the gold passage and of the passage the maximum ranks highest, with a rule at the best document read the maximum finds for each query read. **(b)** Sixty queries edited seven ways and re-encoded, and how often the gold stays in the top-10 under each operator.

G Real examples from BRIGHT

Each example presents a BRIGHT query, a benchmark-labelled relevant passage (the gold), and the passage ranked first by the base block of the trained RLI-4B model. The ranks are the gold's rank under the base vector alone, under the packed dot product (base plus the three aligned reads) and under the cross-read maximum of Table 9; the similarities are the base cosine and the three aligned read cosines to the query. Everything here is scored at the deployed $\kappa = 2$, as in Table 13. The quoted passage is the gold the dot product places highest, and its dot and maximum ranks are that passage's own; where a query has several gold passages the base rank is that of the best-placed one. Over the 1,384 queries the base vector puts a gold first for 490 and the packed vector for 558, moving a gold onto rank one on 121 queries and off it on 53. Texts are quoted verbatim and truncated, with markdown formatting removed. In every example the passage the base vector prefers has the higher base similarity and the lower similarity on read 1, the read the lexical-fidelity term targets. The examples are Figures 8, 9, 10, 11 and 12.

Economics: the nature of water

Query. "What is the economic nature of water? How should one classify water in economic terms? Is it a commodity, a natural resource, can it be both? Does it depend on how it is being used (e.g., as input/raw material in some process)? . . . "

Gold passage. (rank 6 under the base vector, rank 1 under the dot product, rank 1 under the maximum; base 0.752, reads 0.90/0.90/0.86) "Natural Resources: Meaning, Types and Characteristics. Meaning of Natural Resource Economics. Natural resources are derived from the environment. Every man-made product in an economy is composed of natural resources to some degree. The long-term viability of the food supply depends upon the sustainable use of natural resources . . . "

First under the base vector. (base 0.775, reads 0.70/0.85/0.84) a reference-list entry of the *Tragedy of the commons* page: "121. O'Donnell, Erin L.; Talbot-Jones, Julia (2018). 'Creating legal rights for rivers: lessons from Australia, New Zealand, and India'. Ecology and Society. 23 (1): art7 . . . "

Reading. The base vector rewards the topical overlap (water, rivers, rights); the reads reward the passage that defines a natural resource in economic terms.

Figure 8: Economics: a reference-list entry with high topical overlap, against the passage the accepted answer cites.

Biology: when infants begin to retain memories

Query. "How old does a baby have to be before it can retain memories? For most people (my earliest memories are of perhaps when I was 2 or so) their earliest memories would be when they were over a year old. How old does the average baby have to be before it can retain memories? Are there any preconditions that need to be fulfilled?"

Gold passage. (rank 13 under the base vector, rank 1 under the dot product, rank 2 under the maximum; base 0.728, reads 0.83/0.82/0.83) the *Mirror test* page: "Humans. A human child exploring his reflection. The rouge test is a version of the mirror test used with human children. Using rouge makeup, an experimenter surreptitiously places a dot on the face of the child. The children are then placed in front of a mirror and their reactions are monitored; depending on the child's development . . . "

First under the base vector. (base 0.753, reads 0.72/0.76/0.84) "Infants are limited in their ability to interpret their surroundings shortly after birth. Object permanence and understanding of motion typically develop within the first six months of an infant's life, though the specific cognitive processes are not understood . . . "

Reading. Both passages are about infant development; the answer's evidence is self-recognition in the mirror test, which the accepted answer ties to the onset of autobiographical memory, and only the reads separate the two.

Figure 9: Biology: two passages on infant development, one of which the accepted answer cites.

AoPS: power sums of the roots of a cubic

Query. "Let r, s, and t be the three roots of the equation $8x^3 + 1001x + 2008 = 0$. Find $(r + s)^3 + (s + t)^3 + (t + r)^3$."

Gold passage. (rank 63 under the base vector, rank 1 under the dot product, rank 29 under the maximum; base 0.792, reads 0.75/0.87/0.80) AMC 12A 2019, Problem 17: "Let s_k denote the sum of the kth powers of the roots of the polynomial $x^3 - 5x^2 + 8x - 13$. In particular, $s_0 = 3$, $s_1 = 5$, and $s_2 = 9$. Let a, b, and c be real numbers such that $s_{k+1} = a\,s_k + b\,s_{k-1} + c\,s_{k-2}$ for $k = 2, 3, \ldots$ What is $a + b + c$?..."

First under the base vector. (base 0.827, reads 0.62/0.84/0.83) a multiple-choice drill: "The sum of the cube of three numbers is 138, while the sum of their products taken two at a time is 131. Their sum is: Answer Choices: (A) 20 (B) 21 (C) 22 (D) 24 (E) 25. Let's solve the multi-choice question step by step..."

Reading. The drill shares the surface (three numbers, cubes, sums of products); the gold shares the technique (Newton's identities for power sums of roots). In this example, adding the reads improves the gold passage's rank from 63 to 1.

Figure 10: AoPS: a distractor that shares the surface form of the query against a gold that shares its technique.

LeetCode: the maximum number of collinear points

Query. "Given an array of `points` where `points[i] = [xi, yi]` represents a point on the X-Y plane, return the maximum number of points that lie on the same straight line. Example 1: Input: `points = [[1,1],[2,2],[3,3]]` Output: 3. Example 2: Input: `points = [[1,1],[3,2],[5,3],[4,1],[2,3],[1,4]]` Output: 4..."

Gold passage. (rank 104 under the base vector, rank 1 under the dot product, rank 4 under the maximum; base 0.656, reads 0.87/0.90/0.85) LeetCode 2280: "You are given a 2D integer array `stockPrices` where `stockPrices[i] = [dayi, pricei]` indicates the price of the stock on day `dayi` is `pricei`. A line chart is created from the array by plotting the points on an XY plane with the X-axis representing the day and the Y-axis representing the price and connecting adjacent points. ... Return the minimum number of lines needed to represent the line chart...."

First under the base vector. (base 0.781, reads 0.47/0.76/0.87) a two-point helper from a GitHub repository: "`def _get_slope(self)`: Return a float of the increase in y divided by the increase in x, both from left to right. ... `rise = dy - oy; run = dx - ox; return rise / run`"

Reading. Both problems are solved by comparing exact slopes between points (normalised by their greatest common divisor) to decide which points share a line; the base vector locks onto the word *slope* in a helper that computes one. The reads carry the gold from rank 104 to rank 1; the maximum places it fourth.

Figure 11: LeetCode: a helper function matching on the word *slope*, against a problem using the same technique.

Pony: the XOR of an arithmetic sequence

Query. "I will use the programming language pony. Problem: You are given an integer n and an integer start. Define an array nums where nums[i] = start + 2 * i (0-indexed) and n == nums.length. Write a function that returns the bitwise XOR of all elements of nums. Here is the code template: `fun xorOperation(n: I32, start: I32): I32 =>...`"

Gold passage. (rank 6 under the base vector, rank 1 under the dot product, rank 17 under the maximum; base 0.726, reads 0.72/0.75/0.83) the *Precedence* page of the Pony tutorial: "In Pony, unary operators always bind stronger than any infix operators: `not a == b` will be interpreted as `(not a) == b` instead of `not (a == b)`. When using infix operators in complex expressions a key question is the precedence, i.e. which operator is evaluated first..."

First under the base vector. (base 0.748, reads 0.60/0.78/0.85) a listing from the numeric built-ins: "`fun modc(y: A): (A, Bool)` Calculate the modulo of this number after floored division by `y` and return the result and a flag indicating division by zero or overflow. The result will have the sign of the divisor. `fun op_and(y: A): A => this and y...`"

Reading. The base vector matches the operator names in the API listing; the reads prefer the page a programmer needs to write the expression correctly.

Figure 12: Pony: an API listing matching operator names, against the tutorial page for the same operation.