RESTATE: ENABLING CROSS-QUERY KV REUSE FOR TRAINING-FREE ACCELERATION OF BERT RERANKERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Rerankers play a key role in retrieval-augmented generation (RAG) by refining initially retrieved rankings with models specifically trained to score query–document relevance, including commonly used BERT-based cross-encoders. However, rerankers may repeatedly encode unchanged documents across queries, making reuse of precomputed document representations a natural way to reduce redundant computation. Yet document states in BERT rerankers depend on both query interactions and query-dependent positions, preventing exact reuse across queries. But *can't encoders reuse keys and values?* Reranking targets relevance ordering, motivating us to rethink the need to recompute every intermediate document state. To reduce repeated encoding, we propose `ReState`, a training-free method for approximate reuse of document keys and values (K/V) in BERT rerankers. Our central idea is to separate full-document reading from document-state updating: query and score tokens remain active and attend to every document position, while only selected document states are updated under the current query. To reduce quality loss from query-dependent document positions, we propose *Taylor-based position approximation* to compensate for positional changes through precomputed correction components. To recover ranking quality with limited recomputation, we design *Score-anchored restoration* to select update locations using offline score-token attention, without query-time model computation. Across six benchmarks with a frozen MiniLM reranker and prepared caches, `ReState` uses default routing and 20% document updates to achieve a $6.857\times$ mean online pipeline speedup while retaining 96.0% of Full's mean nDCG@10.

1 INTRODUCTION

Retrieval-augmented generation (RAG) (Lewis et al., 2020) improves answer quality and reliability by retrieving documents similar to the query and supplying them as evidence for generation, supporting applications such as open-domain question answering (Lewis et al., 2020). Within this pipeline, rerankers improve the precision and relevance of initially retrieved results (Yu et al., 2024) by scoring query–document relevance with specifically trained models, such as commonly used BERT-based cross-encoders (Nogueira & Cho, 2019). However, different queries can repeatedly retrieve the same documents, especially in workloads over persistent collections, such as internal corporate repositories (Raina & Gales, 2024). In this setting, reranking incurs redundant computation by repeatedly encoding unchanged documents.

Existing methods reduce reranking cost, but repeated document processing remains an efficiency problem for frozen BERT rerankers. For example, distillation (Wang et al., 2020) and candidate pruning (Busolin et al., 2025; Zhou et al., 2026) accelerate reranking within each request, but retained documents must still be encoded again for subsequent queries. Alternatively, document precomputation reuses intermediate states with upper layers still computed online (Saad-Falcon et al., 2023; MacAvaney et al., 2020), or keeps document representations fixed during query interaction (Vast et al., 2026). To avoid repeating document projections and state updates across layers, document K/V can be cached across requests. Causal decoder rerankers support this reuse because causal attention prevents cached document-prefix states from reading subsequent query tokens (An et al., 2025). In contrast, document tokens in query-first BERT attend to the query, and query length

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.

changes their absolute positions. Both effects alter the document states produced by joint encoding and prevent exact prefix-style reuse (Figure 1). This raises a question: *can't encoders reuse K/V?*

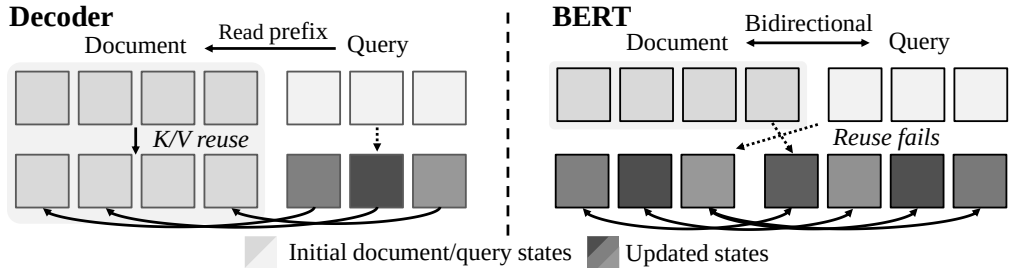

Figure 1: Why exact document K/V reuse fails in BERT. Decoders leave cached prefix states unchanged, whereas standard BERT updates all states through bidirectional query interactions.

Approximate document-state reuse provides an effective direction for accelerating BERT reranking. To preserve full-document context while reducing repeated updates, we propose `ReState` for training-free approximate document K/V reuse. Because preserving document order does not necessarily require identical scores or states, we revisit the need to recompute every intermediate document state. Our central insight is that access to the full document context need not require updating every document state. Accordingly, query and score tokens read all document positions, while only selected document states are updated under the current query. Supporting our central insight, `ReState` achieves a $6.857\times$ mean online pipeline speedup while retaining 96.0% of Full's mean nDCG@10 under default routing (Table 4).

Reusing cached states requires correcting positional discrepancies without sacrificing computational savings. To correct positions without document re-encoding, we propose *Taylor-based position approximation*. Query length shifts document positions, and BERT's learned position embeddings propagate these changes through normalization and encoder layers, which can increase K/V errors (Figure 4). Decoder-side rotary corrections and trained positional adapters do not directly provide training-free correction for BERT's learned absolute embeddings (Yang et al., 2025d; Liu et al., 2026). Our approximation uses the first-order response of embedding LayerNorm and a cached-state surrogate at deeper layers. Reusable correction components can be tabulated offline because BERT's supported positions form a finite set. Applying them during cache reads avoids position-specific K/V copies.

Recovering query interactions requires query-dependent updates, but selecting update locations should not consume the computation saved by reuse. To select useful updates without query-time probing, we design *Score-anchored restoration*. Online state-deviation probes (Yao et al., 2025) execute model layers before selection, adding query-time latency. The relevance head reads the score token, motivating its final-layer document attention as a selection prior. A complete offline pass provides this query-agnostic signal without online probing. This offline prior outperforms common online probes, with gains of up to 0.034 in mean nDCG@10 under shared shallow states and update budgets (Table 3). For query-adaptive selection, we propose `ReState-H`. It supplements this prior with shallow query–document attention and continues selective execution from the resulting joint states. To further reduce online scoring cost, we also filter candidates through lightweight token matching before reranking.

Together, these components form a reranking workflow that reuses document computation. `ReState` first filters candidates, then corrects positional discrepancies in document memory, and finally scores the retained documents through selective updates with full-document attention. In this way, routing reduces the number of candidates to score, while memory reuse reduces repeated computation within each retained pair.

Our contributions are as follows.

1. We identify a gap in existing document precomputation: training-free K/V reuse across all layers of a frozen, query-first BERT reranker remains unresolved. We distinguish positional changes from missing query interactions to explain why direct reuse fails.

2. To address this gap, we propose ReState, combining Taylor-based position approximation with score-anchored restoration to reuse document K/V while retaining full-document attention. ReState-H adds query-adaptive selection through shallow interaction.
3. Across six ranking benchmarks, ReState achieves a 6.857× mean online pipeline speedup while retaining 96.0% of Full's mean nDCG@10 with prepared caches and default routing. Controlled experiments distinguish positional discrepancies from missing query interactions and isolate the contributions of update selection and routing.

2 Related Work

Reusable document representations. Document precomputation leaves different interactions online. ColBERTv2 (Santhanam et al., 2022) uses compressed token vectors for late interaction; PreTTR (MacAvaney et al., 2020) caches lower-layer states for joint upper-layer computation; and MICE (Vast et al., 2026) keeps document representations fixed during query interaction rather than reusing the original encoder's query-conditioned K/V states. Other methods use modularized or learned interactions (Gao et al., 2020; Ji et al., 2024; Zhao et al., 2026) and compressed passages (Liu et al., 2024; 2025; Zhi et al., 2026; Louis et al., 2025). ReState reuses layer-wise K/V while selectively updating document states in a frozen encoder.

Efficient cross-encoder execution. Distillation reduces per-request computation through a smaller model (Wang et al., 2020). Sparse CE (Schlatt et al., 2024) uses local document attention while preserving document access to the query. LTC (Zhuang et al., 2026) pools intermediate tokens to reduce later-layer computation. SEE (Busolin et al., 2025) skips low-relevance candidates using embedding-level token similarity, without training. PRISM (Zhou et al., 2026) clusters intermediate scores and continues only uncertain candidates near the top-K boundary. ReState targets repeated document computation across requests.

KV reuse and selective recomputation. Causal rerankers reuse document-side states (An et al., 2025; Dai et al., 2026; Lu et al., 2026), while generation systems reuse document caches or intermediate activations (Lu et al., 2024; Yang et al., 2025c; Wang et al., 2026a; Bansal, 2025). To improve reuse across contexts, decoder systems adapt positional encoding (Yang et al., 2025d; Liu et al., 2026) or reorder context to increase prefix overlap (Jiang et al., 2025). To reduce cache memory, H_2O retains recent tokens and tokens with high accumulated attention (Zhang et al., 2023), while SnapKV selects head-specific cache positions using attention from an observation window (Li et al., 2024). To restore cross-chunk interactions, CacheBlend uses K/V deviation to select tokens for partial cache recomputation (Yao et al., 2025). Other approaches use query-conditioned or learned selection to allocate recomputation (Teng et al., 2026; Wang et al., 2026b; Yang et al., 2025b). For reuse in a frozen bidirectional reranker, ReState preserves query-first inputs and learned absolute embeddings without additional training.

3 Method

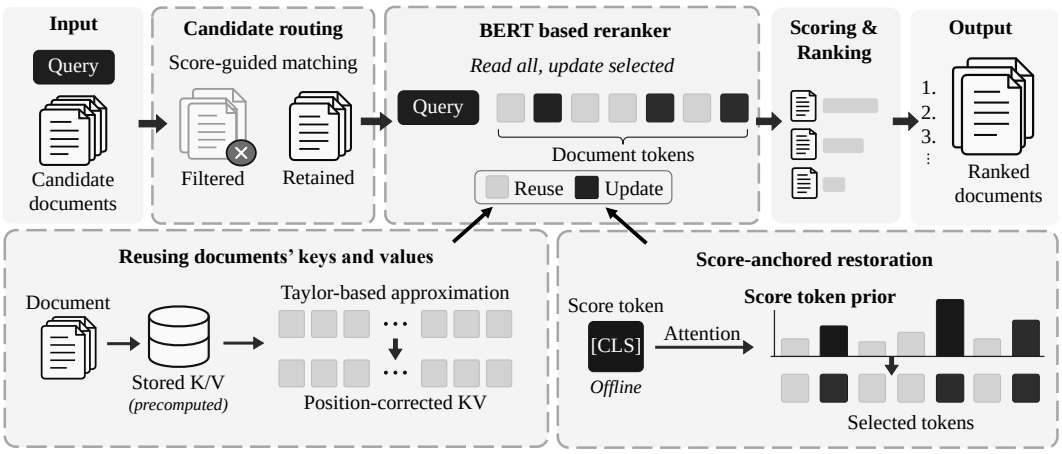

Figure 2: Overview of ReState.

3.1 OVERVIEW

To reduce repeated encoding while preserving query-dependent scoring, we propose `ReState` (Figure 2). Reuse must address changes in document positions and missing query interactions. Accordingly, we combine position correction with selective document updates while retaining full-document attention.First, we prepare layer-wise document K/V, reusable correction components, and score-token update priorities. Next, we avoid scoring low-relevance pairs by filtering candidates through token matching. For retained pairs, we correct cached K/V for their query-dependent positions. Then, to recover part of the missing query interaction, we update the document tokens selected by the stored prior, together with query tokens, the score token, and separators. Unselected document tokens supply corrected cached K/V, so every document position remains visible to attention. Finally, the frozen relevance head scores each pair. Model parameters and query-first input order remain unchanged.

3.2 TAYLOR-BASED POSITION APPROXIMATION

Changing query length shifts document positions even when the text is unchanged. Re-encoding would correct this shift but repeat the document computation. To reduce the resulting discrepancy at low online cost, we propose Taylor-based position approximation.

Memory construction. To retain each layer's document context, we cache $K^{\ell}_{d,\mathrm{off}} = H^{\ell-1}_{d,\mathrm{off}}(W^{\ell}_K)^{\top} + b^{\ell}_K$, analogous values, and source positions; H^0 is the embedding output. Online inputs remain $[\texttt{CLS}; q; \texttt{SEP}; d; \texttt{SEP}]$, with score token `CLS` and query q before document d. Valid tokens attend bidirectionally and padding is masked. Switching to document-first inputs would change the trained order and may require fine-tuning. Therefore, we adjust cached states to their target positions instead.

Discrepancy analysis. Position correction should recover positional changes separately from missing query interactions. To isolate them, let $K^{\ell}_{d,\mathrm{pos}}$ use online document positions while preserving offline tokens, segments, and visibility. Relative to the actual query–document states $K^{\ell}_{d,\mathrm{on}}$,

$$K^{\ell}_{d,\mathrm{on}} - K^{\ell}_{d,\mathrm{off}} = \big(K^{\ell}_{d,\mathrm{pos}} - K^{\ell}_{d,\mathrm{off}}\big) + \big(K^{\ell}_{d,\mathrm{on}} - K^{\ell}_{d,\mathrm{pos}}\big). \tag{1}$$

The first term is the positional discrepancy; the second contains missing online interactions, including special-token visibility changes. Values decompose analogously. Position correction targets the first term; selective recomputation addresses the second.

Taylor correction. Adding a projected position-embedding difference alone omits normalization. To account for this response, we take a first-order Taylor expansion of embedding LayerNorm along $\delta_i = e^{\mathrm{pos}}(t_i) - e^{\mathrm{pos}}(s_i)$, from source position s_i to target t_i. For source embedding $x_i = e^{\mathrm{tok}}(w_i) + e^{\mathrm{seg}}(1) + e^{\mathrm{pos}}(s_i)$, define $c_i = x_i - \bar{x}_i\mathbf{1}$, $v_i = \delta_i - \bar{\delta}_i\mathbf{1}$, and $\sigma_i^2 = \|c_i\|_2^2/D + \epsilon$. Bars denote coordinate means, $\mathbf{1}$ is the all-ones vector, D is the embedding width, and ϵ is the LayerNorm constant. With gain γ, the directional derivative is

$$\mathcal{J}_i[\delta_i] = \frac{\gamma}{\sigma_i} \odot \big(v_i - \rho_i c_i\big), \qquad \rho_i = \frac{\langle c_i, v_i\rangle}{D\sigma_i^2}. \tag{2}$$

The coefficient ρ_i gives the relative first-order change in σ_i. Propagating this perturbation through every encoder block would repeat document computation. Instead, we reuse its scale and shift with each layer's cached state as the anchor:

$$\widetilde{Z}^{\ell}_i = Z^{\ell}_{i,\mathrm{off}} + \frac{p^{\ell}_{Zi}}{\sigma_i} - \rho_i\big(Z^{\ell}_{i,\mathrm{off}} - o^{\ell}_Z\big). \tag{3}$$

Here $Z \in \{K, V\}$, $p^{\ell}_{Zi} = (\gamma \odot v_i)(W^{\ell}_Z)^{\top}$, and $o^{\ell}_Z = \beta(W^{\ell}_Z)^{\top} + b^{\ell}_Z$, with embedding LayerNorm bias β. At layer one, this approximates projected embedding LayerNorm. At deeper layers, it uses a cache-anchored surrogate rather than differentiating the intervening encoder blocks. Appendix A.7 separates the surrogate error from the Taylor remainder.

The reusable correction components can be tabulated offline because BERT's supported positions form a finite set. Online, projected position differences and normalization statistics are obtained

by lookup and fused into attention reads. This keeps one K/V copy per document without online differentiation. `ReState` applies correction at every layer; `ReState-H` applies it after shallow joint computation.

3.3 Score-Anchored Restoration

Position correction cannot supply missing query interactions. Recomputing selected document tokens can recover part of them, but probing the model to choose these tokens adds online computation. To avoid this selection cost, we design Score-anchored restoration using attention collected offline.

Selecting document updates. The relevance head reads the score token's final state. Accordingly, we use its final-layer document attention as a query-agnostic selection prior. The offline pass records $A_{0i}^{L,h}$ from score token 0 to document token i, normalized over valid keys. For H heads, L layers, and n_d document tokens,

$$a_i = \frac{1}{H}\sum_{h=1}^{H} A_{0i}^{L,h}, \qquad k_d = \min(n_d, \lceil r n_d \rceil), \qquad S_d = \mathrm{TopK}(a, k_d). \tag{4}$$

Here r controls the updated fraction, and prior indices match cached document indices. `ReState` stores S_d and begins selective execution at layer one; `ReState-H` also stores a for refinement. The selected locations are fixed offline, but their online states and final scores depend on the query.

Selective interaction. Skipping updates must not remove document tokens from the attention context. Therefore, active rows read current K/V from updated tokens and corrected cached K/V from the rest. Let readers $\mathcal{R}$ contain valid non-document rows, D_d contain document indices, and $U_d = D_d \setminus S_d$ contain unselected rows. Active rows $\mathcal{O} = \mathcal{R} \cup S_d$ partition valid indices as $\mathcal{V} = \mathcal{O} \dot{\cup} U_d$. Original projections produce active Q/K/V for

$$C_{\mathcal{O}}^{\ell} = \mathrm{Attn}\Big(Q_{\mathcal{O}}^{\ell}, [K_{\mathcal{O}}^{\ell}; \widetilde{K}_{U_d}^{\ell}], [V_{\mathcal{O}}^{\ell}; \widetilde{V}_{U_d}^{\ell}]\Big). \tag{5}$$

Only active rows undergo output projection, residuals, LayerNorm, and feed-forward updates. Each token supplies exactly one current or cached K/V source; the key mask follows the concatenation order. Each layer reads its own cache, yielding $|\mathcal{O}| \times N$ attention entries per head for N pair tokens.

3.4 Reader-Guided Refinement

A fixed offline prior cannot adapt update locations to each query. To provide this flexibility, we propose `ReState-H`, which supplements the prior with shallow query–document attention. The joint states also initialize selective continuation, so the added computation serves both selection and scoring.

After c joint layers, we average attention over readers and heads at the next layer to obtain query-conditioned mass m_i, excluding padding. To combine the signals on a common scale, we min–max normalize a_i and m_i to $[0, 1]$ (constant vectors map to zero) and select

$$u_i = (1 - w)\widehat{a}_i + w\widehat{m}_i, \qquad S_d = \mathrm{TopK}(u, k_d), \qquad k_d = \min(n_d, \lceil r n_d \rceil). \tag{6}$$

Here w weights reader attention. Above layer c, active rows continue from the joint states; unselected rows supply corrected offline K/V. Their shallow query-conditioned states are not propagated further, although their earlier influence remains in active states.

3.5 Candidate Routing

Even with selective updates, every scored candidate requires query-side computation and document-cache reads. To avoid these costs for low-relevance pairs, routing filters candidates through inexpensive token matching. Routing selects documents to score; restoration selects tokens to update within them.

Routing score. To avoid another encoder pass, we reconstruct embeddings z_j^q, z_i^d from cached token-plus-segment sums, target positions, and the original LayerNorm. Matching is restricted to a high-priority document subset $\mathcal{T}_d$ to limit cost. Each query token contributes its strongest nonnegative match:

$$g(q,d) = \sum_{j \in q} \max\left(0, \max_{i \in \mathcal{T}_d} \cos(z_j^q, z_i^d)\right). \tag{7}$$

Candidates with the highest $g(q,d)$ are retained for scoring. This reduces candidate evaluations while preserving full-document attention within each retained pair. Retention and small-pool safeguards are specified in the experimental settings.

4 Experiments

4.1 Experimental Setup

Datasets and tasks. Six ranking datasets test efficiency and quality across domains. DL19 (Craswell et al., 2020b) and DL20 (Craswell et al., 2020a) test web passage ranking; FiQA (Thakur et al., 2021) covers financial questions; NFCorpus (Thakur et al., 2021) tests non-technical medical queries; TREC-COVID (Thakur et al., 2021) tests biomedical evidence retrieval; and SCIDOCS (Thakur et al., 2021) tests scientific document matching. Table 1 reports query counts and mean unpadded Full pair lengths before filtering.

Table 1: Ranking datasets.

Dataset	# Queries	Avg. tokens	Description
`DL19`	43	95	Web passage ranking
`DL20`	54	96	Web passage ranking
`FiQA`	648	329	Financial question retrieval
`NFCorpus`	323	353	Biomedical retrieval
`TREC-COVID`	50	348	Biomedical retrieval
`SCIDOCS`	1,000	267	Scientific document matching

Metrics. We report nDCG@10 (Burges et al., 2005) for top-ranked relevance and speedup over each dataset's Full runtime, averaging datasets equally. Ranking disagreement, score MAE, and K/V RMSE measure changes in ordering, scores, and states, respectively.

Baselines. Seven baselines and `Full` assess efficiency and quality against four acceleration strategies. `SEE` (Busolin et al., 2025) and `PRISM` (Zhou et al., 2026) represent training-free candidate reduction. `Sparse CE` (Schlatt et al., 2024) and `LTC` (Zhuang et al., 2026) represent encoder computation reduction with training. `PreTTR` (MacAvaney et al., 2020) and `MICE` (Vast et al., 2026) represent document precomputation, while `ColBERTv2` (Santhanam et al., 2022) represents late interaction. Released checkpoints or local adaptations provide quality references; `Full` is the dense MiniLM control.

Setup. We use a 32-GB RTX 5090. `Full` and `ReState` variants share MiniLM-L12-v2 and a 512-token pair limit; other rerankers retain their evaluated checkpoints and limits. Ranking, position, selector, and pipeline experiments use all 2,118 queries with Top-100 pools. Unless stated otherwise, defaults are Taylor correction, 20% restoration, and routing with 50% candidate retention; `ReState-H` uses two joint layers and $w = .50$. Pipeline timing includes online preparation, routing, transfer, correction, and scoring with eager `Full`, packed host caches, and double buffering (An et al., 2025); retrieval, cache construction, and backend reads are excluded. GPU-resident timing uses 293 queries and each method's corresponding `Full` reference. Further details appear in Appendix A.8.

4.2 Efficiency

Measuring acceleration. We compare GPU-resident execution across rerankers to measure the computation saved by reuse (Figure 3). Panels report speedups separately for the six datasets. Reusing document states should reduce repeated encoding, especially for longer inputs, while shallow joint computation should make `ReState-H` more expensive than `ReState`.

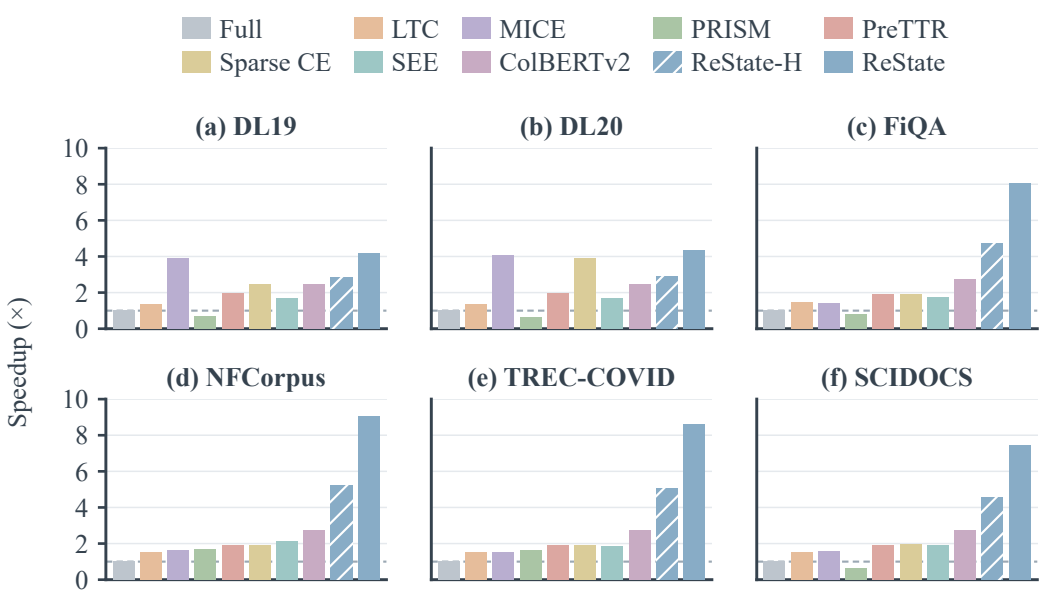

Figure 3: GPU-resident speedup.

Analysis. `ReState`/`ReState-H` achieves mean speedups of 6.955×/4.218× over the corresponding `Full` references; on NFCorpus, these reach 9.082×/5.210×. Cached K/V remove document projections and feed-forward updates for unselected rows. Routing further reduces the number of pairs that undergo scoring. Longer document inputs offer more computation to reuse, explaining the larger gains on NFCorpus. The two joint layers in `ReState-H` retain additional document updates and therefore reduce its speedup.

4.3 Effectiveness

Preserving ranking quality. We compare nDCG@10 to test quality retention under reduced computation (Table 2). Full-document context and selective updates should preserve much of dense ranking quality, while query-aware refinement should improve update locations.

Table 2: Ranking quality.

	Web		Finance	Biomedical		Scientific	All datasets
Method	DL19	DL20	FiQA	NFCorpus	TREC-COVID	SCIDOCS	Avg.
`Full`	0.731	0.673	0.335	0.313	0.709	0.164	0.488
Training-free							
`SEE`	0.679	0.585	0.259	**0.310**	0.678	0.153	0.444
`PRISM`	0.647	0.631	0.291	0.188	0.512	0.144	0.402
`ReState`	0.695	**0.632**	0.299	0.300	**0.724**	0.157	0.468
`ReState-H`	**0.699**	0.627	**0.301**	0.308	0.722	**0.158**	**0.469**
Training-required							
`LTC`	**0.712**	**0.667**	**0.322**	0.330	**0.687**	**0.160**	**0.479**
`PreTTR`	0.693	0.644	0.270	0.324	0.582	0.129	0.440
`ColBERTv2`	0.696	0.666	0.293	0.313	0.678	0.156	0.467
`Sparse CE`	0.653	0.616	0.321	**0.339**	0.647	0.154	0.455
`MICE`	0.655	0.599	0.277	0.315	0.675	0.150	0.445

Analysis. `ReState` and `ReState-H` outperform the evaluated training-free baselines in mean nDCG and remain competitive with training-required baselines. Cached K/V preserve access to all tokens in each scored document. Selective updates then recover query-dependent interaction at chosen positions. Query-aware refinement adjusts these positions to the current query. Together, these mechanisms retain relevance evidence without updating every document state.

4.4 Position Alignment

Isolating position recovery. Document-only (D), equal-length empty-query (E), real-query (Q), and Taylor-corrected (T) caches separate position from query interaction (Figure 4). Normal special tokens are retained; E masks query slots. With routing and restoration disabled ($r = 0$), `ReState-H` retains its dense prefix. Effective correction should reduce T–E versus D–E; E–Q measures interaction at fixed positions.

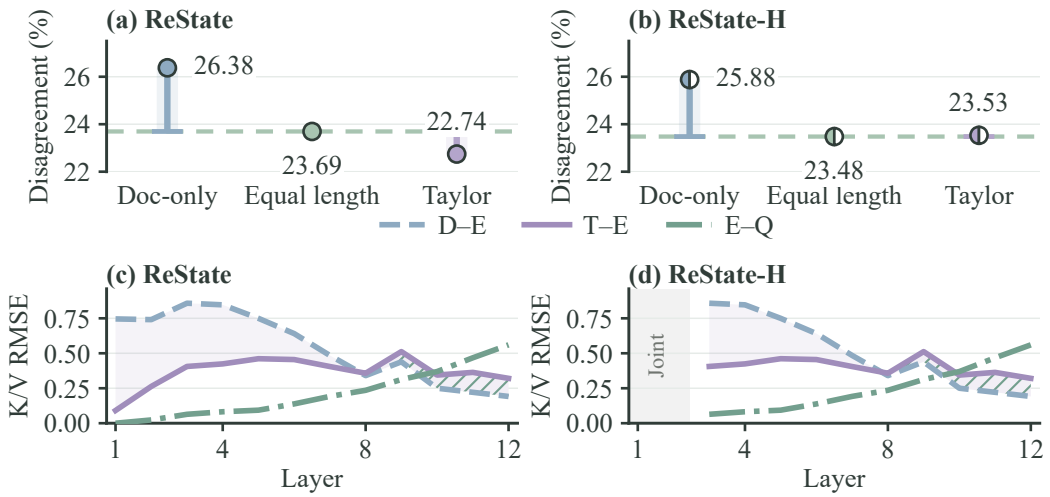

Figure 4: Position alignment.

Analysis. Relative to uncorrected document-only reuse, Taylor reduces mean positional K/V RMSE by 35.700%; for `ReState`, ranking disagreement falls from 26.380% to 22.740%. The growing E–Q residual indicates that semantic discrepancy accumulates through deeper query–document interactions. In contrast, D–E is already large in shallow layers, showing that changing document positions perturbs states before substantial semantic discrepancy develops. The smaller T–E residual through layer seven shows that Taylor removes part of this early positional discrepancy without selective recomputation. The late-layer reversal relative to D–E limits this recovery, supporting complementary roles for position correction and query-dependent restoration.

4.5 Restoration Selection

Testing update priorities. We compare selectors with shared shallow states and update counts, disabling routing (Table 3). Mass uses shallow reader attention following `H2O` and `SnapKV` (Zhang et al., 2023; Li et al., 2024); Drift follows `CacheBlend`'s K/V-deviation criterion (Yao et al., 2025). Mass+score should improve mean ranking quality over either signal alone.

Table 3: Restoration selection.

Selector	DL19	DL20	FiQA	NFCorpus	TREC-COVID	SCIDOCS	Avg.
Mass+score	0.715	**0.653**	**0.320**	0.302	0.709	**0.161**	**0.477**
Score	0.701	0.651	0.313	0.288	0.732	0.160	0.474
Self+score	0.701	0.649	0.306	0.280	**0.734**	**0.161**	0.472
Mass	**0.716**	0.650	0.309	**0.303**	0.696	0.156	0.472
Drift+score	0.713	0.642	0.288	0.302	0.718	0.157	0.470
Self	0.703	0.643	0.296	0.270	0.732	0.156	0.467
Drift	0.704	0.613	0.238	0.283	0.667	0.137	0.440
Random	0.674	0.610	0.228	0.247	0.706	0.145	0.435

Analysis. Score provides the strongest standalone selection quality, exceeding both online Mass and Drift in mean nDCG. Although computed offline, its final-layer attention comes from a complete document-encoding pass. This gives it access to deeper document context than the shallow online signals, consistent with their weaker standalone results. Its advantage supports score-token attention as an empirical priority for allocating recomputation in reranking. Mass+score further improves the

mean, indicating that shallow query information complements this offline prior rather than replacing it.

4.6 Routing & Pipelined Serving

Separating routing and reuse. We compare six execution paths to isolate the quality and speed contributions of routing and cache reuse (Table 4). Routing (R) is toggled for each scorer, and routed paths share the same retained candidates. Reuse should accelerate scoring even without pruning, while routing should provide an additional gain by reducing candidate evaluations.

Table 4: Routing and reuse.

Metric	`Full`	`Full`+R	`ReState` (+R)	`ReState-H` (+R)	`ReState` (without R)	`ReState-H` (without R)
Avg. nDCG@10	0.488	0.478	0.468	0.469	0.474	0.477
Avg. speedup	1.000×	1.808×	6.857×	4.189×	4.041×	2.337×

Analysis. With default routing, `ReState` achieves a 6.857× mean pipeline speedup while retaining 96.0% of Full's mean nDCG@10. Reuse saves document computation within each scored pair, whereas routing reduces the number of pairs. Without routing, `ReState` still achieves a 4.041× speedup, demonstrating savings from reuse alone. Routing provides additional acceleration but can discard relevant candidates, exposing a quality–efficiency tradeoff.

4.7 Downstream Question Answering

Preserving answer quality. We test answer quality on 1,000 queries each from NQ-Open (Kwiatkowski et al., 2019) and TriviaQA (Joshi et al., 2017) (Figure 5). Each reranker supplies ten passages to a fixed `Qwen3-8B` (Yang et al., 2025a) reader; routing is disabled for `ReState` variants. Preserving useful evidence should limit Exact Match (EM) and token F1 losses (Rajpurkar et al., 2016).

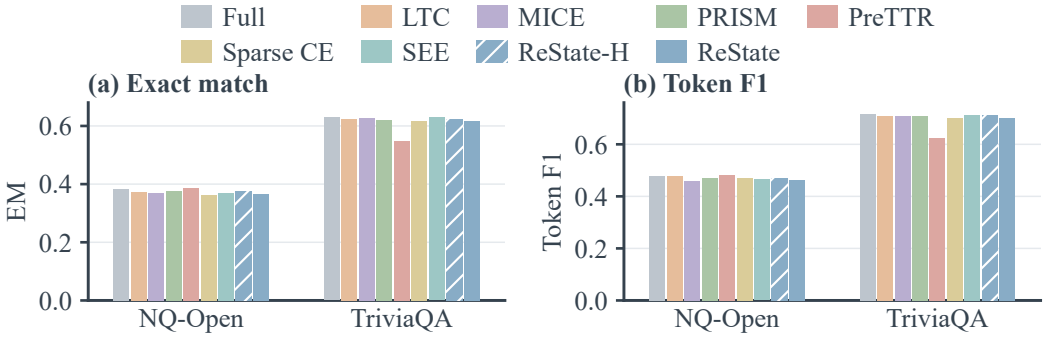

Figure 5: Downstream QA accuracy.

Analysis. Maximum EM/F1 losses relative to Full remain below 1.5/1.4 percentage points for `ReState` and 0.8/0.9 for `ReState-H`. With the reader fixed, differences arise from passage selection and ordering. Disabling routing isolates reuse from candidate removal. These bounded losses are consistent with approximate scores preserving useful evidence. The smaller losses under `ReState-H` are consistent with its query-conditioned updates.

5 Conclusion

To accelerate frozen BERT rerankers without training, we introduced `ReState`, separating full-document reading from offline-guided selective updates. Across six benchmarks, position-corrected K/V reuse accelerates execution with prepared caches and default routing while retaining most of Full's ranking quality. These findings suggest that effective reranking need not update every document state.

AI USE STATEMENT

Generative AI tools, including OpenAI Codex, assisted with method refinement, mathematical exposition, experimental design, and parameter selection. They also assisted with implementation, debugging, experiment execution, data reformatting, and result interpretation. We used generative AI for literature search and summarization, manuscript drafting and language editing, and reference formatting. AI tools also helped create and revise scientific figures and editable graphics. Verification included model and input checks, implementation consistency checks, and comparisons between reported numbers and saved experiment outputs. The authors take responsibility for the final text, methods, figures, code, and reported results.

REPRODUCIBILITY STATEMENT

Our code and experimental records are available at https://anonymous.4open.science/r/ReState-ICLR2027-EF27.

ETHICS STATEMENT

Our experiments use existing ranking and question-answering benchmarks and pretrained models, without recruiting human participants or collecting new personal data. `ReState` can inherit biases from these models and datasets, and approximate reuse can change document rankings. Therefore, deployment should assess ranking quality in the intended application and apply the source documents' access controls to cached representations.

REFERENCES

Yuwei An, Yihua Cheng, Seo Jin Park, and Junchen Jiang. HyperRAG: Enhancing quality-efficiency tradeoffs in retrieval-augmented generation with reranker KV-cache reuse. *arXiv preprint arXiv:2504.02921*, 2025. doi: 10.48550/arXiv.2504.02921.

Harsh Vardhan Bansal. LLMCache: Layer-wise caching strategies for accelerated reuse in transformer inference. In *2025 13th International Conference on Intelligent Systems and Embedded Design (ISED)*, pp. 924–929. IEEE, 2025. doi: 10.1109/ISED67359.2025.11405274.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 89–96, 2005. doi: 10.1145/1102351.1102363.

Francesco Busolin, Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Salvatore Trani, and Alberto Veneri. Efficient re-ranking with cross-encoders via early exit. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2534–2544, 2025. doi: 10.1145/3726302.3729962.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the TREC 2020 deep learning track. In *Proceedings of the Twenty-Ninth Text REtrieval Conference, TREC 2020*, volume 1266 of *NIST Special Publication*. National Institute of Standards and Technology, 2020a. doi: 10.6028/NIST.SP.1266.deep-overview.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. Overview of the TREC 2019 deep learning track. In *Proceedings of the Twenty-Eighth Text REtrieval Conference, TREC 2019*, volume 1250 of *NIST Special Publication*. National Institute of Standards and Technology, 2020b.

Beiya Dai, Yifan Wei, Guang Yang, Xing Shi, Xinbing Wang, and Zhouhan Lin. DoPR: Reusable Compressed Document Prefixes for Efficient LLM Reranking. *arXiv preprint arXiv:2609.03311*, 2026. doi: 10.48550/arXiv.2609.03311.

Luyu Gao, Zhuyun Dai, and Jamie Callan. Modularized transformer-based ranking framework. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 4180–4190, 2020. doi: 10.18653/v1/2020.emnlp-main.342.

Ziwei Ji, Himanshu Jain, Andreas Veit, Sashank J. Reddi, Sadeep Jayasumana, Ankit Singh Rawat, Aditya Krishna Menon, Felix Yu, and Sanjiv Kumar. Efficient document ranking with learnable late interactions. *arXiv preprint arXiv:2406.17968*, 2024. doi: 10.48550/arXiv.2406.17968.

Yinsicheng Jiang, Yeqi Huang, Liang Cheng, Cheng Deng, Xuan Sun, and Luo Mai. RAGBoost: Efficient retrieval-augmented generation with accuracy-preserving context reuse. *arXiv preprint arXiv:2511.03475v1*, 2025. doi: 10.48550/arXiv.2511.03475.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019. doi: 10.1162/tacl_a_00276.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, 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. Curran Associates, Inc., 2020.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Qi Liu, Bo Wang, Nan Wang, and Jiaxin Mao. Leveraging Passage Embeddings for Efficient Listwise Reranking with Large Language Models. *arXiv preprint arXiv:2406.14848*, 2024. doi: 10.48550/arXiv.2406.14848.

Qi Liu, Yanzhao Zhang, Mingxin Li, Dingkun Long, Pengjun Xie, and Jiaxin Mao. E2Rank: Unifying Text Embedding and Listwise Reranking for Effective and Efficient Search. *arXiv preprint arXiv:2510.22733*, 2025. doi: 10.48550/arXiv.2510.22733.

Yang Liu, Yunfei Gu, Liqiang Zhang, Chentao Wu, Guangtao Xue, Jie Li, Minyi Guo, Junhao Hu, and Jie Meng. CacheSlide: Unlocking cross position-aware KV cache reuse for accelerating LLM serving. In *24th USENIX Conference on File and Storage Technologies*, pp. 83–99. USENIX Association, 2026.

Maxime Louis, Thibault Formal, Hervé Déjean, and Stéphane Clinchant. OSCAR: Online Soft Compression And Reranking. *arXiv preprint arXiv:2504.07109*, 2025. doi: 10.48550/arXiv.2504.07109.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating retrieval-augmented generation with precomputed KV caches for chunked text. *arXiv preprint arXiv:2410.07590*, 2024. doi: 10.48550/arXiv.2410.07590.

Xuan Lu, Haohang Huang, Yingqi Fan, Junlong Tong, Yuxuan Zhang, Ping Nie, Rui Meng, and Xiaoyu Shen. CompRank: Efficient LLM Reranking via Token-Level Compression and Decoding-Free Scoring. *arXiv preprint arXiv:2606.11700*, 2026. doi: 10.48550/arXiv.2606.11700.

Sean MacAvaney, Franco Maria Nardini, Raffaele Perego, Nicola Tonellotto, Nazli Goharian, and Ophir Frieder. Efficient document re-ranking for transformers by precomputing term representations. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 49–58, 2020. doi: 10.1145/3397271.3401093.

Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*, 2019.

Vatsal Raina and Mark Gales. Question-based retrieval using atomic units for enterprise RAG. In *Proceedings of the Seventh Fact Extraction and VERification Workshop (FEVER)*, pp. 219–233. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.fever-1.25.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1264.

Jon Saad-Falcon, Amanpreet Singh, Luca Soldaini, Mike D'Arcy, Arman Cohan, and Doug Downey. Embedding recycling for language models. In *Findings of the Association for Computational Linguistics: EACL 2023*, pp. 1933–1953. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.findings-eacl.145.

Jon Saad-Falcon, Daniel Y. Fu, Simran Arora, Neel Guha, and Christopher Ré. Benchmarking and building long-context retrieval models with LoCo and M2-BERT. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42918–42946. PMLR, 2024.

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, 2022. doi: 10.18653/v1/2022.naacl-main.272.

Ferdinand Schlatt, Maik Fröbe, and Matthias Hagen. Investigating the effects of sparse attention on cross-encoders. In *Advances in Information Retrieval*, volume 14609 of *Lecture Notes in Computer Science*, pp. 173–190. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-56027-9_11.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021. doi: 10.48550/arXiv.2104.09864.

Xin Teng, Canyu Zhang, Shaoyi Zheng, Danyang Zhuo, Tianyi Zhou, and Shenji Wan. InfoFlow KV: Information-flow-aware KV recomputation for long context. *arXiv preprint arXiv:2603.05353*, 2026. doi: 10.48550/arXiv.2603.05353.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Mathias Vast, Victor Morand, Basile van Cooten, Laure Soulier, Josiane Mothe, and Benjamin Piwowarski. MICE: Minimal interaction cross-encoders for efficient re-ranking. *arXiv preprint arXiv:2602.16299*, 2026. doi: 10.48550/arXiv.2602.16299.

Jiahao Wang, Weiyu Xie, Mingxing Zhang, Boxin Zhang, Jianwei Dong, Yuening Zhu, Chen Lin, Jingqi Tang, Yaochen Han, Zhiyuan Ai, Xianglin Chen, Yongwei Wu, and Congfeng Jiang. From prefix cache to fusion RAG cache: Accelerating LLM inference in retrieval-augmented generation. *Proceedings of the ACM on Management of Data*, 4(1):1–28, 2026a. doi: 10.1145/3786655.

Shihao Wang, Jiahao Chen, Yanqi Pan, Hao Huang, Yichen Hao, Xiangyu Zou, Wen Xia, Wentao Zhang, Chongyang Qiu, and Pengfei Wang. ProphetKV: User-query-driven selective recomputation for efficient KV cache reuse in retrieval-augmented generation. *arXiv preprint arXiv:2602.02579*, 2026b. doi: 10.48550/arXiv.2602.02579.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5776–5788, 2020.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a. doi: 10.48550/arXiv.2505.09388.

Bin Yang, Qiuyu Leng, Jun Zeng, and Zhenhua Wu. CacheClip: Accelerating RAG with effective KV cache reuse. *arXiv preprint arXiv:2510.10129*, 2025b. doi: 10.48550/arXiv.2510.10129.

Huan Yang, Renji Zhang, Mingzhe Huang, Weijun Wang, Yin Tang, Yuanchun Li, Yunxin Liu, and Deyu Zhang. KVShare: An LLM service system with efficient and effective multi-tenant KV cache reuse. *arXiv preprint arXiv:2503.16525*, 2025c. doi: 10.48550/arXiv.2503.16525.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. In *Advances in Neural Information Processing Systems*, volume 38, 2025d. doi: 10.52202/085713-4461.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109. ACM, 2025. doi: 10.1145/3689031.3696098.

Yue Yu, Wei Ping, Zihan Liu, Boxin Wang, Jiaxuan You, Chao Zhang, Mohammad Shoeybi, and Bryan Catanzaro. RankRAG: Unifying context ranking with retrieval-augmented generation in LLMs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 121156–121184. Curran Associates, Inc., 2024. doi: 10.52202/079017-3850.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Xinping Zhao, Jiaxin Xu, Ziqi Dai, Xin Zhang, Shouzheng Huang, Danyu Tang, Xinshuo Hu, Meishan Zhang, Baotian Hu, and Min Zhang. KaLM-Reranker-V1: Fast but not late interaction for compressed document reranking. *arXiv preprint arXiv:2606.22807*, 2026. doi: 10.48550/arXiv.2606.22807.

Zhewei Zhi, Yingyi Zhang, Yizhen Jing, Xianneng Li, Jianing Liu, Huajie Liu, and Yongliang Ding. Compress-then-Rank: Faster and Better Listwise Reranking with Large Language Models via Ranking-Aware Passage Compression. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(41):35059–35067, 2026. doi: 10.1609/aaai.v40i41.40811.

Jiahao Zhou, Chengliang Lin, Dingji Li, Mingkai Dong, and Haibo Chen. On-device semantic selection made low latency and memory efficient with monolithic forwarding. In *Proceedings of the 21st European Conference on Computer Systems*, pp. 126–143. ACM, 2026. doi: 10.1145/3767295.3803572.

Shengyao Zhuang, Zhichao Xu, and Ivano Lauriola. Layer-wise token compression for efficient document reranking. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4426–4432. ACM, 2026. doi: 10.1145/3805712.3809871.

A Appendix

A.1 GPU-Resident Execution

Checking acceleration across datasets. Table 5 lists the numerical values in Figure 3 for comparison across input lengths. This is the GPU-resident protocol from Section 4.1; transfer is not measured. Longer documents should offer more encoder computation to avoid.

Table 5: GPU-resident speedups by dataset.

	Web		Finance	Biomedical		Scientific	
Method	DL19	DL20	FiQA	NFCorpus	TREC-COVID	SCIDOCS	Avg.
Full	1.000×	1.000×	1.000×	1.000×	1.000×	1.000×	1.000×
LTC	1.378×	1.369×	1.494×	1.521×	1.497×	1.501×	1.460×
SEE	1.679×	1.715×	1.752×	2.112×	1.837×	1.929×	1.837×
PRISM	0.668×	0.628×	0.824×	1.673×	1.656×	0.644×	1.015×
PreTTR	1.948×	1.955×	1.932×	1.924×	1.932×	1.929×	1.937×
MICE	3.920×	4.050×	1.392×	1.646×	1.510×	1.591×	2.352×
ColBERTv2	2.481×	2.464×	2.766×	2.763×	2.767×	2.768×	2.668×
Sparse CE	2.457×	3.915×	1.930×	1.906×	1.925×	1.961×	2.349×
ReState	**4.189×**	**4.371×**	**8.060×**	**9.082×**	**8.602×**	**7.427×**	**6.955×**
ReState-H	2.831×	2.917×	4.738×	5.210×	5.049×	4.563×	4.218×

Analysis. ReState averages 6.955× speedup, reaching 9.082× on NFCorpus versus 4.189× on DL19. Longer inputs contain more document projections and feed-forward updates that reuse can skip. Query-side computation and selection overhead take a larger fraction of short-input execution. The dense prefix in ReState-H restores interaction but reduces the fraction of skipped work. Therefore, these resident results characterize execution savings; they do not measure storage-backend or transfer performance.

A.2 Transfer across BERT Checkpoints

Testing reuse beyond MiniLM. We evaluate MiniLM, MonoBERT (Nogueira & Cho, 2019), and nBoost BERT-base to test transfer across frozen absolute-position rerankers. The two BERT-base models have 12 layers and width 768; this study uses a shared document-only cache layout, six complete datasets, a 512-token limit, and no routing. Each plotted segment starts at the corresponding Full reference. Speed and quality are paired within each checkpoint, so consistent acceleration would support the execution strategy beyond the main model.

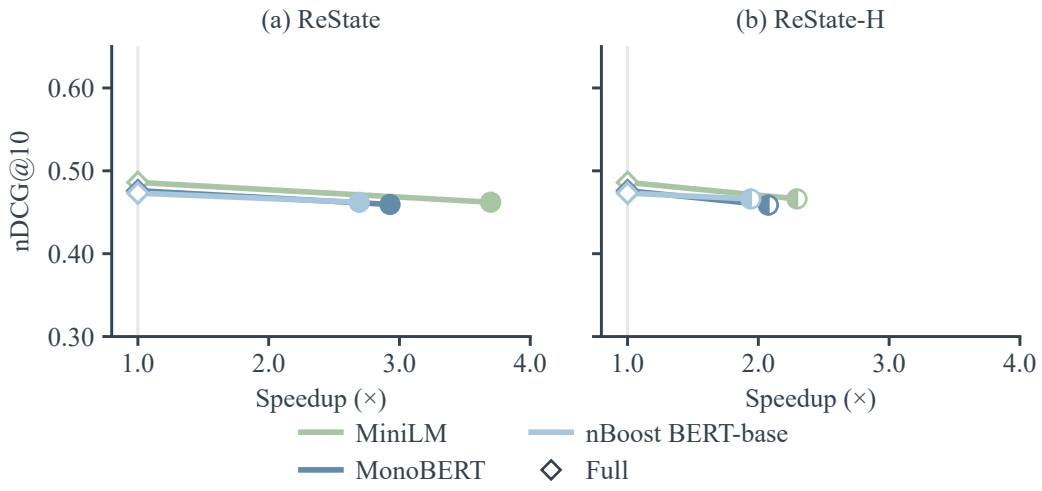

Figure 6: Transfer across BERT checkpoints.

Analysis. Across the two BERT-base checkpoints, `ReState` averages 2.692–2.928× speedup and `ReState-H` averages 1.942–2.076×. Skipping document updates reduces execution cost in both larger encoders. Checkpoint width affects both the arithmetic saved and the K/V volume moved, so the speedup need not equal MiniLM's. The joint prefix in `ReState-H` spends additional computation to expose query information before selection. Refinement improves mean quality for MiniLM and nBoost but not MonoBERT, showing that this information does not guarantee a better update set.

A.3 Transfer to Rotary-Position Encoders

Testing a different positional interface. We evaluate BCE ModernBERT across six complete datasets to test reuse with rotary positions (Su et al., 2021). The model has 22 layers, local and global attention, and a mean-pooling relevance head. It uses 256-token inputs without routing; MiniLM is a 512-token reference, so the comparison tests architectural transfer rather than isolating positional encoding.

Mean pooling provides no dedicated score token. Therefore, `ReState` selects updates by offline final-layer document attention mass, whereas `ReState-H` uses reader mass after two joint layers, without BERT's 50:50 fusion. Both retain rotary key alignment, 20% document updates, and selected-row residual completion for pooling. We also test rank-8 adapters fitted separately for each variant on 32 training and eight validation queries with no normalized-text overlap with the six test sets. The adapters were frozen before evaluation; folding them into online attention projections preserves matrix sizes. Triangles mark adapted results, and short dashed segments connect them to training-free reuse. Their fitting protocol is specified once in Section A.8.

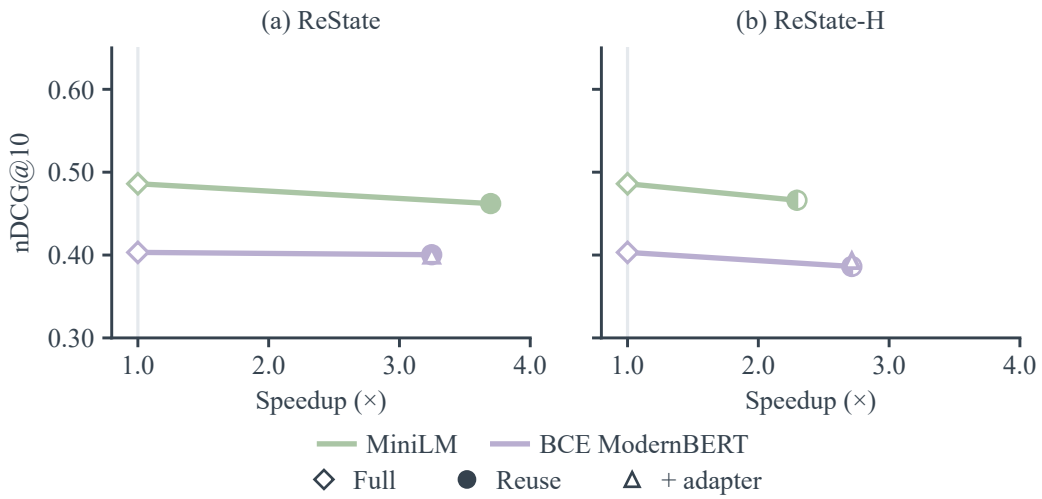

Figure 7: Transfer to a rotary-position encoder.

Analysis. On BCE ModernBERT, `ReState` averages 0.400 nDCG versus Full's 0.403 at 3.246× speedup. Rotary alignment makes cached keys position-compatible, but cannot restore query-conditioned document states. Refinement lowers mean quality to 0.386, showing that reader mass does not necessarily select better updates than the offline prior. Adaptation raises `ReState-H` to 0.393, but provides no mean improvement for `ReState`. Because this pilot uses one rotary encoder with mean pooling and no dedicated score token, the result is architecture-specific transfer evidence rather than a guarantee of quality preservation across rotary rerankers. Folding adapters into existing projections preserves matrix sizes, keeping mean speedups within 0.1% of their training-free counterparts.

A.4 Cache Storage and Reuse

Accounting for persistent memory. We quantify the storage exchanged for reduced document computation (Table 6). For L layers, n document tokens, hidden width d, and b bytes per scalar, the payload is

$$B_{\text{doc}} = 2Lndb + 4n + B_{\text{metadata}}.$$

The terms represent K/V, FP32 score priors, and token/source-position indices plus selection and routing metadata. Position tables are shared once per model. The table isolates MiniLM K/V ($L = 12$, $d = 384$); serving uses FP16, while FP32 supports diagnostics. Collection-scale estimates assume 256 cached tokens per entry.

Table 6: Document K/V storage.

Cache dtype	Per token (KiB)	Per entry (MiB)	Million entries (TiB)
FP16	18.000	4.500	4.292
FP32	36.000	9.000	8.583

Analysis. An FP16 256-token entry requires 4.500 MiB of K/V, or 4.292 TiB (4.719 TB) for one million entries, plus 1 KiB of score priors per entry and metadata. Capacity grows with layer count, hidden width, and cached length. Reducing the restoration fraction does not shrink K/V because unselected tokens remain attention sources. Shared position tables avoid storing another cache for each query length. Thus, selective execution reduces online work while retaining a substantial persistent-memory cost.

Measuring evaluation-pool capacity. Table 7 counts distinct document-prefix entries in the saved evaluation pools. Capacity sums their actual cached token counts in FP16, excluding padding, metadata, shared tables, and workspace. Prefix reuse should reduce repeated storage, while different encoded boundaries may require separate entries.

Table 7: Evaluation-pool cache capacity.

Dataset	Documents	Prefixes	Tokens (M)	K/V (GiB)
DL19	4,297	4,297	0.366	6.280
DL20	5,396	5,396	0.465	7.978
FiQA	22,632	22,632	5.305	91.065
NFCorpus	3,537	3,537	1.205	20.685
TREC-COVID	4,370	4,370	1.400	24.027
SCIDOCS	22,634	25,684	6.733	115.580

Analysis. The six pools contain 265.615 GiB of logical FP16 K/V; SCIDOCS accounts for 115.580 GiB. Persistent cost depends on the distinct cached prefixes and their lengths, rather than the number of query appearances alone. Different prefix boundaries change bidirectional states and can require separate entries. Position correction allows each such entry to serve different query lengths without position-specific copies. These logical capacities are neither simultaneous GPU allocation nor a measurement of full-corpus serving.

Separating storage from working memory. The corpus cache need not reside entirely in host or GPU memory. Following HyperRAG's storage hierarchy (An et al., 2025), persistent entries can be partitioned across shared NVMe or remote backends, host memory can retain a bounded working set, and GPU buffers can stage active batches. Consequently, corpus capacity and resident memory are separate budgets. For example, two buffers of 100 entries with 256 tokens each require 900 MiB of FP16 K/V, before padding, metadata, model weights, and workspace. Our double buffering overlaps next-batch host-to-GPU transfer with current-batch scoring. This can hide the transfer within the scoring interval, without requiring the corpus cache to reside in RAM. The main-text pipeline measures this host-to-GPU leg; cache construction and reads from the persistent backend remain outside its timing boundary.

Amortizing construction. Cache construction encodes each distinct document prefix and writes its states once per version. Final-layer score attention can be collected during the same pass when

K/V and the prior use the same initialization; a different prior initialization requires another pass. For build cost C_{build} and u subsequent accesses, reuse saves computation and access time when

$$u\big(C_{\text{Full}} - C_{\text{reuse}}\big) > C_{\text{build}},$$

where C_{reuse} includes cache fetching, transfer, and online execution under the chosen backend. Thus, access frequency and backend performance determine amortization; prepared-host timings alone do not determine a corpus-level break-even point.

Maintaining evolving collections. To prevent stale reuse, a deployment should version entries by document content, prefix boundaries, tokenizer, encoder checkpoint, and offline initialization. Additions require new entries; edits invalidate affected K/V and priors; deletions remove their entries. Unchanged documents remain reusable. Changing encoder weights or tokenization requires rebuilding dependent entries. Until a valid cache is available, requests can use full encoding. This incremental policy avoids rebuilding the entire corpus for an isolated document edit; corpus updates and backend eviction are outside the present timing measurements.

A.5 Execution Correctness

Recovering dense execution. We first check whether selective execution recovers the original reranker when every document row is updated (Table 8). The control increases restoration to 100% on five diagnostic queries and 100 pairs. Agreement up to floating-point error would separate implementation errors from the approximation introduced by skipped updates.

Table 8: Full-recompute equivalence.

Model	Mean $\lvert\Delta s\rvert$	Max $\lvert\Delta s\rvert$
`MiniLM-L12`	5.940×10^{-7}	2.380×10^{-6}

Analysis. Full restoration gives 1.000 ranking agreement and a maximum score error of 2.380×10^{-6}. Each token contributes one current K/V source when all rows are active. This recovers the dense attention context and the original state updates. Therefore, the test supports the correctness of the segmented execution on these inputs. It does not establish that a partial-restoration cache reproduces query-dependent document states.

Separating kernel effects. We next time 100% restoration to isolate kernel overhead from reuse (Table 9). This FP32 control uses fixed 50-query subsets, five warmups, five measurements, and an independently tuned SDPA Full reference. Without skipped updates, any speed difference should come from the execution implementation.

Table 9: Full-recompute kernel control.

Dataset	Pairs	`Full` (s)	100% recompute (s)	Speedup	nDCG@10
`DL20`	4,928	0.701	0.697	1.006×	0.317
`TREC-COVID`	5,000	2.861	3.515	0.814×	0.710

Analysis. At full restoration, speedups are $1.006\times$ on DL20 and $0.814\times$ on TREC-COVID, with 1.000 ranking agreement. The segmented path performs the same document arithmetic as dense attention. Indexing and kernel launches can add overhead when no rows are skipped. Reuse creates its saving by removing document projections and state updates. Thus, this control does not attribute the main speedups to an intrinsically faster dense kernel.

A.6 Position Correction Controls

Comparing correction quality and cost. We compare position rules to test whether reduced cache error improves ranking without losing the execution saving. This control uses the 293-query subsets with routing disabled; source caches, selected rows, and checkpoints are fixed within each variant at 20% restoration. A useful correction should improve the quality–cost tradeoff, rather than minimize state RMSE alone.

None reads the source cache D unchanged; Raw adds the projected position-embedding difference. LN-first uses the exact finite embedding-LayerNorm difference only in layer one; LN-all projects it into every layer. Taylor uses the proposed cache-anchored approximation. E is re-encoded at target positions with query slots masked, while online scoring uses the real query. LN-first and None coincide for `ReState-H`, which starts cached execution at layer three.

The correction control uses FP32, 32-pair batches, five warmups, and seven interleaved repetitions. Batch times are summed per query before taking the median, then averaged within and across datasets. Timing includes checkpoint execution, selection, correction, and scoring; LN-all includes its projections and transient cache writes. All rules retain one persistent K/V copy. RMSE pools scalar errors against E within each dataset before equal-dataset averaging; E itself is untimed.

Table 10: Position-correction comparison.

		nDCG@10, 20%		ms/query, 20%	
Rule	RMSE	`ReState`	`ReState-H`	`ReState`	`ReState-H`
None	0.600	0.457	0.475	25.058	41.915
Raw	0.559	0.459	0.475	25.074	41.922
LN-first	0.560	0.458	0.475	25.350	41.933
LN-all	0.358	0.467	0.477	39.488	54.441
Taylor	0.384	0.470	0.475	25.344	42.168
Target E	0.000	0.470	0.479	—	—

Analysis. Taylor reaches 0.470 `ReState` nDCG in 25.344 ms/query, compared with 0.467 in 39.488 ms for LN-all. Uniform K/V RMSE does not weight coordinates by their influence on the relevance score. Consequently, LN-all's lower RMSE does not imply better ranking. Its token-dependent projections and transient cache writes also add online work. Taylor uses shared preprojected terms during cache reads, keeping its measured cost within 0.270 ms of Raw.

Checking consistency across datasets. Table 11 expands the 20% restoration comparison to test whether the aggregate result holds across domains. The correction-only controls above are unchanged. Consistent gains would support the approximation across these distributions, whereas reversals would identify its limits.

Table 11: Position correction by dataset.

		`ReState`			`ReState-H`		
Dataset	`Full`	Raw	LN-all	Taylor	Raw	LN-all	Taylor
DL19	0.731	0.692	0.703	0.704	0.710	0.716	0.715
DL20	0.667	0.642	0.645	0.647	0.651	0.651	0.647
FiQA	0.347	0.289	0.303	0.308	0.324	0.313	0.314
NFCorpus	0.332	0.280	0.288	0.297	0.317	0.317	0.314
TREC-COVID	0.709	0.700	0.709	0.711	0.688	0.696	0.700
SCIDOCS	0.167	0.150	0.155	0.154	0.161	0.168	0.160

Analysis. Taylor improves `ReState` over Raw on all six datasets, with mean nDCG rising from 0.459 to 0.470. A position correction changes the context read by active rows even when their update locations stay fixed. Therefore, the shared selection separates this gain from choosing a different token set. For `ReState-H`, two joint layers already change the states from which selective execution continues. Its mixed outcomes across correction rules show that the best approximation depends on that execution context.

Combining correction and restoration. We repeat the equal-length position bridge at 20% restoration to test whether query-dependent updates complement correction (Figure 8). Routing stays disabled and the cache conditions share selected rows and checkpoint states. Restoration should reduce disagreement with Full; the cached layer residuals do not change, so they are not plotted again.

Disagreement uses unrounded scores. Dashed lines mark E's disagreement with Full; vertical segments compare these rates and are not direct ranking distances to E.

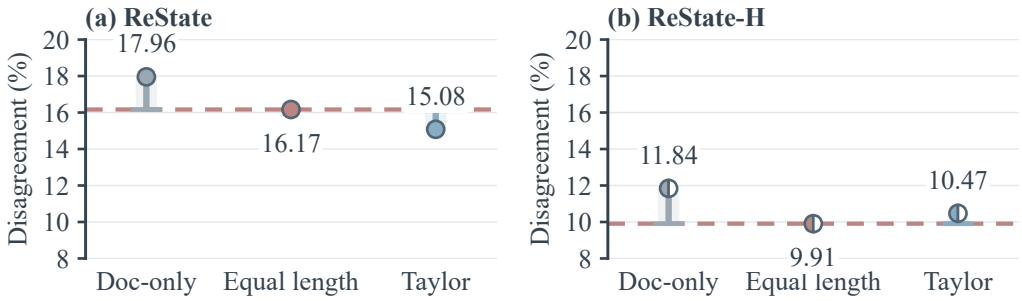

Figure 8: Position alignment with restoration.

Analysis. At 20% restoration, Taylor lowers `ReState`'s disagreement from D's 17.957% to 15.080%, a 2.877-point reduction. Position correction changes the reused context, while restoration updates states using the current query. Both variants are closer to Full than at zero restoration, consistent with these complementary roles. For `ReState-H`, target-position E still gives lower disagreement than Taylor, leaving a positional residual after restoration. Therefore, selective updates improve scoring fidelity without making the position approximation exact.

Stressing positional displacement. We vary target positions to test whether correction degrades at larger shifts (Table 12). The first two documents from each of the QASPER-title and QMSum subsets of LoCo (Saad-Falcon et al., 2024) supply four fixed prefixes at each of 64 and 128 tokens. We insert 8–376 masked query slots, retain normal special tokens, and compare against exact target-position re-encoding in FP32, without restoration or routing. All inputs remain within MiniLM's 512-position table. K/V RMSE is averaged over four documents against the equal-position empty-query reference. Effective correction should reduce K/V RMSE relative to the uncorrected cache; gap closure is $1 - \mathrm{RMSE}_{\mathrm{Taylor}}/\mathrm{RMSE}_{\mathrm{None}}$.

Table 12: Sensitivity to position shifts.

Tokens	Shift	None	Raw	Taylor	LN-all	Taylor gap closed
64	8	0.640	0.593	0.393	0.365	38.516%
64	32	0.876	0.816	0.524	0.478	40.136%
64	128	0.926	0.870	0.615	0.576	33.648%
64	256	0.983	0.923	0.623	0.588	36.619%
64	376	0.911	0.854	0.574	0.537	36.950%
128	8	0.641	0.594	0.404	0.372	37.061%
128	32	0.815	0.760	0.491	0.450	39.727%
128	128	0.953	0.895	0.636	0.588	33.319%
128	256	1.046	0.981	0.645	0.592	38.348%
128	376	0.946	0.889	0.613	0.566	35.208%

Analysis. Taylor closes 33.319%–40.136% of the positional gap across these ten conditions; at a 376-position shift with 128 document tokens, RMSE decreases from 0.946 to 0.613. This reduction persists without query semantics or selective updates, isolating a contribution from position correction. The nonmonotonic residual is consistent with learned absolute embeddings: a larger index shift need not produce a proportionally larger embedding perturbation. LN-all retains lower RMSE, leaving both finite-shift approximation error and deeper propagation error relevant to the comparison. These four-document controls test displacement within the supported table, rather than accuracy beyond the model's native context limit.

A.7 Taylor Approximation Derivation

The derivation separates three quantities: the exact embedding-LayerNorm differential, its finite-shift Taylor approximation, and the deeper cache-anchored surrogate. This distinction identifies which error is controlled by the expansion and which comes from modeling deep position propagation.

LayerNorm differential. To account for the normalization response to a position shift, we linearize embedding LayerNorm at the cached input. Let $x_i \in \mathbb{R}^{1\times D}$ be the token-plus-segment-plus-position embedding at source s_i, and $\delta_i = e^{\text{pos}}(t_i) - e^{\text{pos}}(s_i)$ the shift to target t_i. Define the centering projector and source statistics by

$$\Pi = I - \frac{\mathbf{1}\mathbf{1}^\top}{D}, \qquad c_i = x_i\Pi, \qquad v_i = \delta_i\Pi, \qquad \sigma_i^2 = \frac{\|c_i\|_2^2}{D} + \epsilon. \tag{8}$$

Writing $\Gamma = \text{diag}(\gamma)$, embedding LayerNorm is $\text{LN}_{\text{emb}}(x_i) = c_i\Gamma/\sigma_i + \beta$. Its differential along the position change admits a centered scaling term and a rank-one variance correction:

$$\begin{aligned}\mathcal{J}_i[\delta_i] &:= \frac{\mathrm{d}}{\mathrm{d}a}\,\text{LN}_{\text{emb}}(x_i + a\delta_i)\Big|_{a=0} \\ &= \delta_i\left(\frac{\Pi}{\sigma_i} - \frac{c_i^\top c_i}{D\sigma_i^3}\right)\Gamma = \frac{\gamma}{\sigma_i}\odot(v_i - \rho_i c_i), \qquad \rho_i = \frac{\langle c_i, v_i\rangle}{D\sigma_i^2}.\end{aligned} \tag{9}$$

The projector removes a uniform embedding shift; the rank-one term accounts for the change in variance along the centered source state. Thus, two tokens with the same position shift can receive different corrections. For either key or value projection, the first-layer transport is

$$\widetilde{Z}_i^1 = Z_{i,\text{off}}^1 + \mathcal{J}_i[\delta_i](W_Z^1)^\top, \qquad Z \in \{K, V\}. \tag{10}$$

This uses the exact LayerNorm directional derivative, with a first-order approximation to the finite shift. The finite-shift remainder is given below.

Transporting layer-wise memory. To retain this inexpensive correction at depth, we anchor a position-response surrogate at each layer's actual cached K/V. Let $p_{Zi}^\ell = (\gamma \odot v_i)(W_Z^\ell)^\top$ and $o_Z^\ell = \beta(W_Z^\ell)^\top + b_Z^\ell$. Along perturbation magnitude a, define

$$\begin{aligned}\sigma_i(a) &= \sqrt{\|c_i + av_i\|_2^2/D + \epsilon}, \\ \widehat{Z}_i^\ell(a) &= o_Z^\ell + \frac{\sigma_i\big(Z_{i,\text{off}}^\ell - o_Z^\ell\big) + ap_{Zi}^\ell}{\sigma_i(a)}, \qquad \widehat{Z}_i^\ell(0) = Z_{i,\text{off}}^\ell.\end{aligned} \tag{11}$$

At layer one this is the actual projected embedding; at depth, it transfers the normalization-induced scale and shift to the cached state. The position change is evaluated through the tangent at the offline anchor:

$$\frac{\mathrm{d}\widehat{Z}_i^\ell(a)}{\mathrm{d}a}\bigg|_{a=0} = \frac{p_{Zi}^\ell}{\sigma_i} - \rho_i\big(Z_{i,\text{off}}^\ell - o_Z^\ell\big). \tag{12}$$

Evaluating the first-order Taylor polynomial $\widehat{Z}_i^\ell(0) + \widehat{Z}_i^{\ell\prime}(0)$ at unit displacement gives Equation (3). At depth, the expanded function is the defined cache surrogate, not the full encoder. The residual analysis below separates surrogate discrepancy from Taylor truncation.

Exact first-layer position change. Let x_i be the cached token's embedding sum before embedding LayerNorm. With token and segment embeddings fixed, $\Delta e_i = \delta_i$ changes only its absolute position. At inference, the bracketed LayerNorm difference is the exact finite embedding change. Projecting it with the respective key and value matrices gives the exact first-layer K/V changes:

$$\begin{aligned}\Delta K_i^1 &= \big[\text{LN}_{\text{emb}}(x_i + \Delta e_i) - \text{LN}_{\text{emb}}(x_i)\big](W_K^1)^\top, \\ \Delta V_i^1 &= \big[\text{LN}_{\text{emb}}(x_i + \Delta e_i) - \text{LN}_{\text{emb}}(x_i)\big](W_V^1)^\top.\end{aligned} \tag{13}$$

Projection biases cancel; the two distinct matrices map the embedding difference into key and value coordinates. Replacing the bracketed difference with $\mathcal{J}_i[\delta_i]$ gives the first-order approximation, not the exact finite change. At layer one, $Z_{i,\text{off}}^1 - o_Z^1 = (\gamma \odot c_i)(W_Z^1)^\top/\sigma_i$. Substituting the differential in Equation (9) into Equation (10) gives the closed form in Equation (3). The surrogate in Equation (11) is exact at this layer because $\text{LN}_{\text{emb}}(x_i + a\delta_i) = (c_i + av_i)\Gamma/\sigma_i(a) + \beta$.

Finite-shift remainder. Set $F_Z(x) = \mathrm{LN}_{\mathrm{emb}}(x)(W_Z^1)^\top + b_Z^1$ and $f_i^Z(a) = F_Z(x_i + a\delta_i)$. Taylor's formula with integral remainder yields

$$\begin{aligned} Z_{i,\mathrm{pos}}^1 - \widetilde{Z}_i^1 &= \int_0^1 (1-a)\,\frac{\mathrm{d}^2 f_i^Z(a)}{\mathrm{d}a^2}\,\mathrm{d}a, \\ \left\| Z_{i,\mathrm{pos}}^1 - \widetilde{Z}_i^1 \right\| &\le \frac{1}{2} \sup_{a\in[0,1]} \left\| \mathrm{D}^2 F_Z(x_i + a\delta_i) \right\|_{\mathrm{op}} \|\delta_i\|^2. \end{aligned} \tag{14}$$

The operator norm is that of the bilinear second derivative. Since $\epsilon > 0$, this derivative is continuous along the interpolation segment. The bound is local in the position-embedding displacement; learned embeddings at discrete positions need not be close, so it does not imply uniformly small residuals across query lengths.

Deep-state residual. At deeper layers, Equation (11) matches the offline cache at $a = 0$ but need not equal the exact target-position cache at $a = 1$. Its derivative at the anchor gives the implemented correction in Equation (3). Consequently,

$$\begin{aligned} Z_{i,\mathrm{pos}}^\ell - \widetilde{Z}_i^\ell = \underbrace{Z_{i,\mathrm{pos}}^\ell - \widehat{Z}_i^\ell(1)}_{\text{surrogate error}} \\ + \underbrace{\widehat{Z}_i^\ell(1) - \widehat{Z}_i^\ell(0) - \widehat{Z}_i^{\ell\prime}(0)}_{\text{Taylor remainder}}. \end{aligned} \tag{15}$$

The surrogate error vanishes at layer one but can dominate at depth. A higher-order expansion directly improves only the Taylor remainder. Applying the exact embedding-LN difference to every deeper layer is a different surrogate and need not improve total error. Neither position rule supplies the semantic-query term in Equation (1).

A.8 Implementation and Evaluation Protocols

Preserving the original attention sources. Algorithm 1 specifies the offline memory and online execution used by the BERT variants. Query tokens, CLS, both separators, and selected document rows form the active set. Current and cached K/V occupy disjoint original pair indices; the validity mask follows the same ordering, and padded columns receive $-\infty$ before softmax. Inference disables dropout and preserves the original relevance head.

The assignments denote one K/V source per valid token. TaylorRead applies Equation (3) inside attention, without materializing a second corrected cache.

Shared tables and fused evaluation. For fixed model weights, precompute $T_Z^\ell(p) = (\gamma \odot [e^{\mathrm{pos}}(p) - \mathrm{mean}(e^{\mathrm{pos}}(p))])(W_Z^\ell)^\top$ and offsets o_Z^ℓ. Then $p_{Zi}^\ell = T_Z^\ell(t_i) - T_Z^\ell(s_i)$. For document segment ID 1, let $\nu(w,p)$ be the variance plus ϵ of the token-plus-segment-plus-position embedding. Define the centered position difference $\delta_c(s,t) = [e^{\mathrm{pos}}(t) - e^{\mathrm{pos}}(s)]\Pi$ and its mean squared magnitude $\eta(s,t) = \|\delta_c(s,t)\|_2^2/D$. The two required scalars follow from table lookups:

$$\sigma_i^{-1} = \nu(w_i, s_i)^{-1/2}, \qquad \rho_i = \frac{\nu(w_i,t_i) - \nu(w_i,s_i) - \eta(s_i,t_i)}{2\nu(w_i,s_i)}. \tag{16}$$

The identity follows by expanding $\|c_i + v_i\|^2$; the two variance stabilizers cancel in the numerator. The implementation constructs variance tables in FP64 and stores them in FP32. These tables are shared across all documents. They add model-level storage, not additional document K/V copies. Only document K/V, token IDs, source positions, and selection/router metadata persist per document.

Online execution computes the two scalars once per document row. A Triton attention kernel loads each unselected cached K/V tile, applies Equation (3), and consumes it immediately. Selected rows contribute their current K/V only. The implementation uses 32-row query tiles, stops each document loop at its actual length, and prepares concatenated Q/K/V projection weights once. No dense Jacobian matrix, online automatic differentiation, or separate corrected-cache write is required. The same correction applies through the final layer; `ReState-H` begins cached execution at layer three after retaining its two dense layers.

Algorithm 1 Offline caching and online scoring.

Require: Frozen reranker, documents $\mathcal{D}$, query q, candidates $\mathcal{C}$, routing flag, variant, r, c, w

Offline: once per cache configuration

Prepare shared projected-position and normalization tables
for $d \in \mathcal{D}$ **do**
  Encode d offline; store document K/V, token IDs, and source positions as M_d
  Store prior a_d and routing support $\mathcal{T}_d$; $k_d \leftarrow \min(n_d, \lceil r n_d \rceil)$
  Store $S_d^{\texttt{ReState}} \leftarrow \mathrm{TopK}(a_d, k_d)$
end for

Online: for each query

$\mathcal{C}' \leftarrow \mathrm{Route}(q, \mathcal{C})$ ▷ $\mathcal{C}' = \mathcal{C}$ if routing is disabled
for $d \in \mathcal{C}'$ **do**
  $H^0 \leftarrow \mathrm{Embed}(q, d)$; $(t, S_d) \leftarrow (0, S_d^{\texttt{ReState}})$ ▷ Query-first input
  $(\sigma^{-1}, \rho) \leftarrow \mathrm{PositionStatistics}(M_d, q)$
  if variant is `ReState-H` **then**
    $H^c \leftarrow \mathrm{DenseLayers}_{1:c}(H^0)$; $t \leftarrow c$; $m \leftarrow \mathrm{ReaderAttention}_{c+1}(H^c)$
    $S_d \leftarrow \mathrm{TopK}((1-w)\widehat{a}_d + w\widehat{m}, k_d)$
  end if
  $U_d \leftarrow D_d \setminus S_d$; $\mathcal{O} \leftarrow \mathcal{R} \cup S_d$; gather $H_{\mathcal{O}}^t$
  for $\ell = t+1, \ldots, L$ **do**
    $(Q_{\mathcal{O}}, K_{\mathcal{O}}, V_{\mathcal{O}}) \leftarrow \mathrm{Project}_\ell(H_{\mathcal{O}}^{\ell-1})$
    $(K, V)[\mathcal{O}] \leftarrow (K_{\mathcal{O}}, V_{\mathcal{O}})$
    $(K, V)[U_d] \leftarrow \mathrm{TaylorRead}_\ell(M_d, U_d, \sigma^{-1}, \rho)$
    $C \leftarrow \mathrm{softmax}(Q_{\mathcal{O}} K^\top / \sqrt{d_h} + M)V$ ▷ $M_{ij} = 0$ for valid keys, $-\infty$ otherwise
    $H_{\mathcal{O}}^\ell \leftarrow \mathrm{OutputBlock}_\ell(H_{\mathcal{O}}^{\ell-1}, C)$ ▷ Original residual, LN, FFN
  end for
  $s_d \leftarrow \mathrm{RelevanceHead}(H_{\texttt{CLS}}^L)$
end for
return candidates in $\mathcal{C}'$ ordered by s_d

Precision and routing safeguards. Routing matches queries against the highest-priority half of document tokens. Ranking and serving use FP16 K/V with FP32 computation and priors; position and selector controls use FP32 caches.

Routing retains at least ten candidates and bypasses pools of at most twenty.

Pipeline execution. The main checkpoint is `cross-encoder/ms-marco-MiniLM-L12-v2`. The implementation uses Python 3.12.3, PyTorch 2.8.0+cu128, Transformers 5.15.0, and Triton 3.4.0 on RTX 5090. Contiguous pinned FP16 K/V and FP32 priors feed two device buffers; a separate copy stream overlaps transfer and scoring. Full/Full+R and shallow joint layers use eager attention. Blocks contain at most ten queries split into batches of at most five. Three warmups precede seven interleaved repetitions; block medians are weighted by query count. The timer includes online input/layout preparation, routing, candidate-ID resolution, transfer, checkpoints, selection, correction, and scoring. It starts from tokenized inputs and packed host caches, excluding retrieval, construction, backend reads, host packing/allocation, and pipeline fill/drain. Optimized execution preserves bitwise scores against its FP16 reference on all 2,118 queries.

When the kth and $(k+1)$th selection priorities differ by at most $32\epsilon_{\mathrm{FP32}}$, a single-pair shallow replay at the true sequence length resolves selection. Scoring still uses the original batched checkpoint. The boundary check, synchronization, and any replays are included in timing.

GPU-resident timing. The GPU-resident comparison uses the first 50 canonical queries per dataset, or all 43 for DL19 (293 total). The GPU-resident timer includes request preparation and online execution, starting with cached states already on the GPU. `ReState` variants retain the main defaults and use eager attention, three warmups, and seven interleaved repetitions. ColBERTv2 encodes queries per pair.

Metrics and selector conventions. nDCG@10 uses linear gain $\max(\mathrm{rel}, 0)$ under the TREC convention (Craswell et al., 2020b) and all saved qrels for IDCG; unjudged documents have zero gain, and queries with zero IDCG contribute zero. Means weight queries equally within datasets, then datasets equally. Ranking disagreement is $d_q = (N_{\mathrm{rev}} + \frac{1}{2}N_{\mathrm{tie}})/N_{\mathrm{eligible}}$: among candidate pairs with unequal Full scores, reversed orders count one and method ties count one half.

Selection uses the same cached prefix as K/V. Self averages offline final-layer attention over valid document readers and heads; Mass averages online readers at layer $c + 1$. Mixtures use equal weights and min–max normalization, except Self+score, which adds offline document attention with L1 normalization. Min–max normalization maps constant vectors to zero, and ties favor earlier positions. Random averages three seeds. The joint checkpoint is reused for continuation, including ratio and weight endpoint controls.

BERT transfer execution. The checkpoints are `cross-encoder/ms-marco-MiniLM-L12-v2`, `veneres/monobert-msmarco`, and `nboost/pt-bert-base-uncased-msmarco`. They share a 512-token pair limit with no separate document cap, normal special tokens and document segment IDs in source caches, FP32 computation, and FP16 K/V. Four-query blocks use two buffers; a final singleton joins the preceding block. Two warmups precede five repetitions with randomized method order, and block medians are query-weighted. Runs span two RTX 5090 instances, with all methods paired on one GPU within each block. Three-query DL19 and TREC-COVID probes check native full-restoration equivalence and an independent attention implementation before each checkpoint runs.

RoPE execution and adaptation. The checkpoint is `tomaarsen/reranker-msmarco-v1.1-ModernBERT-base-bce`. It uses the BERT transfer block protocol, FP32 computation, FP16 K/V, and FP32 final-state caches and priors. Full uses native eager attention, while reuse uses SDPA; this is an implementation comparison, not a matched-kernel ablation. Timings are paired within each block on an RTX 5090. The native and independent-attention checks pass before continuation.

To align rotary keys without re-encoding, the RoPE executor caches keys before rotation and applies the target-position rotation. With row-vector keys and layer-specific rotation $R_\ell(p)$,

$$K_i^\ell(t) = K_{i,\mathrm{pre}}^\ell R_\ell(t)^\top = K_i^\ell(s) R_\ell(s) R_\ell(t)^\top.$$

Values are not rotated. This changes the explicit positional phase; it does not restore query-conditioned hidden states or remove the effect of local attention masks.

Rank-8 adapters modify online QKV and attention-output projections, adding 811,008 parameters for `ReState` or 737,280 for `ReState-H`. Encoder weights, relevance heads, and source caches remain frozen. Fitting uses 32 training and eight validation queries from a custom MS MARCO dev-small split, with at most eight candidates per query. Four AdamW epochs use learning rate 3×10^{-4}, weight decay 0.01, and gradient clipping at 1. The loss combines teacher-distribution KL, 0.1-weighted score MSE, active-state MSE, and pooled-state MSE. Validation chooses a checkpoint from candidates including zero initialization. Calibration queries have no normalized-text overlap with the six test sets or shared-ID overlap with DL19/DL20.

Pooling completion control. A separate five-query control on each of DL19 and DL20 holds selected rows and final states fixed. Replacing updated rows directly instead of applying residual completion lowers BCE `ReState` nDCG from 0.480 to 0.241. This isolates a pooling error beyond key alignment and token selection. A readout-contribution prior measures score changes when a token is removed from pooling at fixed encoder states; it is not the same intervention as query-dependent recomputation. These pilot diagnostics are excluded from the six-dataset means.

QA reader execution. QA starts from Top-100 pools; mean unpadded Full pair lengths are 150 and 160 tokens for NQ-Open and TriviaQA. EM measures exact answers, while token F1 measures partial overlap. Qwen3-8B uses BF16, batch size two, greedy decoding, and no thinking, with input/output limits of 4,096/32 tokens. The prompt requests the shortest supported answer span. Included passages are ordered from lowest rank to rank one, placing the highest-ranked evidence nearest the question under left truncation. Identical prompts share deterministic generation. Full and `ReState` variants were regenerated together; the other baselines use saved reader outputs.

A.9 Baseline Controls and Adaptations

Comparing pipeline execution. We pair the MiniLM checkpoint and Top-100 inputs to compare reuse with SEE and PRISM while disabling routing in `ReState` (Table 13). The control uses the first 50 canonical queries per dataset, or all 43 for DL19, with FP32 eager computation and FP16 document K/V. Seven interleaved measurements follow three warmups; two buffers overlap

host-to-GPU transfer with scoring, and the timer includes gathering, selection or clustering, and execution. Cache construction, host packing, backend reads, and pipeline fill/drain remain outside this prepared-cache boundary. Reuse should reduce scoring cost while retaining all candidates, independently of candidate pruning.

Table 13: Matched pipeline comparison.

	Full	SEE	PRISM	ReState (without R)	ReState-H (without R)
Avg. nDCG@10	0.492	0.447	0.408	0.481	0.483
Avg. speedup	1.000×	1.759×	1.017×	3.903×	2.249×

Analysis. ReState reaches 0.481 nDCG at 3.903× speedup, compared with SEE's 0.447 at 1.759× and PRISM's 0.408 at 1.017×. Reuse saves document-state updates within every scored pair while retaining all candidates. In contrast, candidate pruning saves further computation by restricting which pairs continue through the model. Including selection, clustering, and host-to-GPU transfer shows that the reuse benefit survives these online costs in the tested pipeline. This comparison isolates the shared-checkpoint operating points, without establishing dominance over every pruning threshold or implementation.

The following settings specify the released checkpoints and local adaptations behind the reference comparisons.

SEE and PRISM. SEE uses the released embedding-level amax signal, per-query normalization, and a fixed proximity-static threshold of 0.200, retaining at least ten candidates. PRISM applies progressive boundary clustering to scalar MiniLM scores with a fixed threshold of 0.500, layers 0–11, and one or two clusters. Earlier accepted clusters rank first; candidates within each cluster use their current layer score. The thresholds remain fixed across datasets and across quality and timing evaluation.

LTC checkpoint. Our LTC implementation starts from ms-marco-MiniLM-L12-v2 and mean-pools contiguous document states to 25% at layer 6. We fine-tune it on 500 MS MARCO groups, each containing one positive and three negatives. Training uses two epochs, listwise loss, learning rate 2×10^{-5}, and seed 42. Scoring retains the trained 32-query/256-document limit.

Sparse CE checkpoint. The released webis/sparse-cross-encoder-4-512 checkpoint provides the sparse-attention comparison on the complete Top-100 pools in the main-text ranking table.

PreTTR checkpoint. The released split-3 PreTTR checkpoint uses the release's fixed query slot and precomputes document states through layer 3. Online query encoding is followed by three joint layers of the six-layer DistilBERT backbone, using resident document states.

A.10 Quality–Cost Tradeoffs

Varying document updates. We pair ranking quality with pipeline latency while varying restoration over 5%, 10%, 20%, and 40% (Figure 9). The sweep uses the 293-query subsets and compares routing on and off. Along each curve, restoration increases from bottom to top. Larger markers mark 20%, half-filled markers denote ReState-H, and dashed curves disable routing. More updates should recover query-dependent states at higher computation cost; pipeline overlap can make latency non-monotonic.

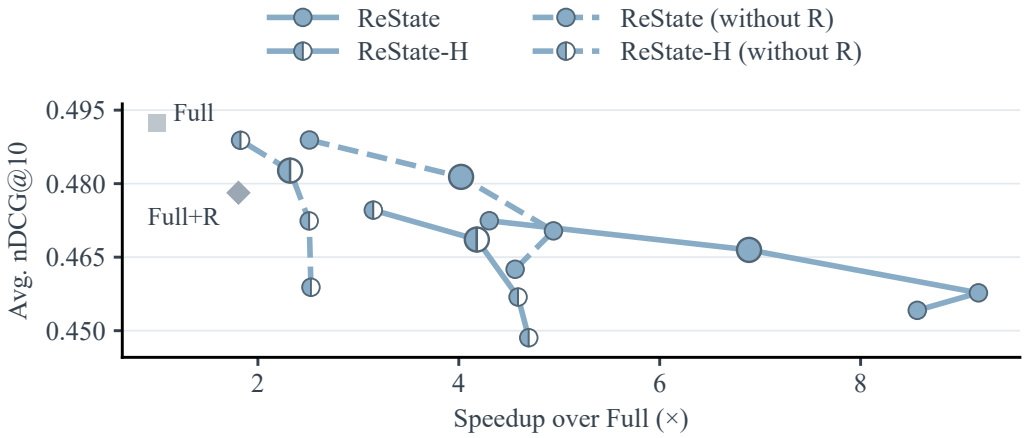

Figure 9: Quality–speed tradeoffs.

Analysis. Without routing, 40% restoration brings both variants to 0.489 nDCG, close to Full's 0.492. Additional updates allow more document states to incorporate the current query. Routing removes candidate computations, producing a distinct efficiency–quality tradeoff. Transfer overlap and batching can make latency non-monotonic; the faster 10% `ReState` point does not imply less arithmetic than 5%. At the smallest update fraction, `ReState-H`'s extra joint computation does not yield a better tradeoff, so refinement should not be treated as uniformly dominant.

Separating ratio, weight, and depth. Table 14 varies restoration, reader-mass weight, and joint depth one at a time on all DL19 queries, using FP32 without routing. Full reaches 0.731 nDCG@10. The shared setting gives 14 distinct configurations; ratio and weight endpoints retain the checkpoint and reader probe. More restoration or deeper joint computation should recover interaction, whereas the weight sweep tests the balance between selection signals.

The main operating point remains $r = .20$, $c = 2$, $w = .50$, with routing fractions fixed at one half. These sweeps characterize sensitivity and do not establish an independently calibrated optimum; the defaults have not been validated through a disjoint calibration-and-test protocol.

Table 14: Parameter sensitivity.

r	w	c	`ReState`	`ReState-H`
0.000	0.500	2	0.664	0.655
0.050	0.500	2	0.691	0.688
0.100	0.500	2	0.698	0.697
0.200	0.000	2	0.704	0.701
0.200	0.250	2	0.704	0.712
0.200	0.500	1	0.704	0.704
0.200	0.500	2	0.704	0.715
0.200	0.500	3	0.704	0.719
0.200	0.500	4	0.704	0.721
0.200	0.750	2	0.704	0.711
0.200	1.000	2	0.704	0.716
0.300	0.500	2	0.711	0.717
0.500	0.500	2	0.719	0.721
1.000	0.500	2	0.731	0.731

Analysis. Increasing joint depth from one to four layers raises `ReState-H` nDCG from 0.704 to 0.721. A deeper prefix exposes more query–document interaction before selective execution. The two-layer default retains part of this recovery while limiting dense work. Full restoration reaches the Full score because every document state is updated. Since this table measures quality alone, it cannot identify the depth with the best serving tradeoff.

A.11 Reader-Guided Fusion Sensitivity

Testing signal complementarity. We sweep 13 reader-mass weights from 0 to 1 to test whether combining the offline prior and online signal helps (Figure 10). Ten paired rounds sample 20 queries per dataset with FP32 and no routing; 1,200 appearances contain 658 unique queries. The curve averages dataset means and then rounds, and complementarity would favor an interior weight over both endpoints.

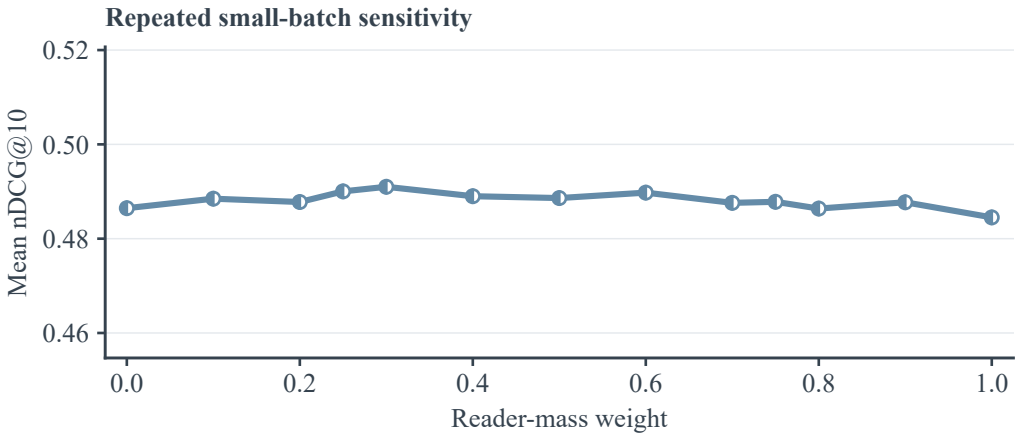

Figure 10: Reader-mass fusion sensitivity.

Analysis. The highest sampled mean is 0.491 at $w = .30$, compared with 0.489 at equal fusion, 0.486 at score-only, and 0.485 at mass-only. The offline prior and shallow reader attention reflect different computation contexts. An interior mixture can retain useful offline choices while responding to the current query. The non-monotonic response shows that increasing reader mass alone does not ensure better selection. Therefore, this curve supports complementarity while leaving the main equal-weight rule unchanged.

A.12 Speedup Aggregation

Comparing aggregation targets. We aggregate the same six-dataset pipeline measurements to distinguish a typical dataset from the evaluated request mixture. Let F_d and M_d denote mean time per query and n_d the query count. The main arithmetic mean, $D^{-1}\sum_d F_d/M_d$, weights domains equally. The harmonic mean is $D/(\sum_d M_d/F_d)$, whereas $S_{\text{work}} = (\sum_d n_d F_d)/(\sum_d n_d M_d)$ estimates the total-time ratio for the 2,118-query mixture. Different request mixtures need not yield the same speedup.

Table 15: Speedup aggregation.

	Without R		With R	
Aggregation	`ReState`	`ReState-H`	`ReState`	`ReState-H`
Arithmetic mean	4.041	2.337	6.857	4.189
Harmonic mean	3.713	2.226	6.229	3.941
Workload time ratio	4.602	2.594	7.736	4.664

Analysis. Without routing, the workload time ratios are $4.602\times$ /$2.594\times$ for `ReState`/`ReState-H`, versus arithmetic means of $4.041\times$ /$2.337\times$. Longer-input requests contribute more Full execution time and offer more document computation to reuse. Consequently, the request-weighted time ratio is higher on this particular mixture. Arithmetic and harmonic means instead summarize dataset-level ratios and should not be interpreted as total serving-time ratios. All three retain the same prepared-cache timing boundary.

A.13 Dataset-Level Routing and Serving

Separating ranking effects. Table 16 expands the main pipeline comparison to test how routing changes quality within each dataset. Full and the reuse variants use the same packed inputs; routed paths share retained candidate IDs. Routing can improve a particular ordering but may lose relevant candidates, so it need not preserve quality uniformly.

Table 16: Ranking quality by dataset.

Dataset	Full	Full+R	ReState	ReState-H	ReState (without R)	ReState-H (without R)
DL19	0.731	0.715	0.695	0.699	0.704	0.715
DL20	0.673	0.652	0.632	0.627	0.652	0.653
FiQA	0.335	0.308	0.299	0.301	0.315	0.320
NFCorpus	0.313	0.314	0.300	0.308	0.289	0.302
TREC-COVID	0.709	0.719	0.724	0.722	0.724	0.709
SCIDOCS	0.164	0.161	0.157	0.158	0.161	0.161
Avg.	0.488	0.478	0.468	0.469	0.474	0.477

Analysis. Routing lowers mean ReState nDCG from 0.474 to 0.468 and ReState-H from 0.477 to 0.469. Removing candidates changes the ordering problem before cross-encoder scoring. Relevant candidates lost at this stage cannot be recovered by better cached states. Therefore, dataset-specific gains do not establish that filtering preserves evidence. Retaining the complete pool isolates reuse when quality preservation takes priority.

Measuring the pipeline saving. Table 17 pairs these quality results with the main online timer. Each cell reports milliseconds per query followed by speedup; timing includes online request preparation and transfer. Means average each dataset's Full/method ratio, rather than dividing the mean times. Longer inputs should offer more work to skip, while routing should add a separate reduction in scored pairs.

Table 17: Pipeline efficiency by dataset.

Dataset	Full	Full+R	ReState	ReState-H	ReState (without R)	ReState-H (without R)
DL19	18.563/1.000	11.428/1.624	4.414/4.206	6.486/2.862	7.264/2.556	11.190/1.659
DL20	18.925/1.000	11.482/1.648	4.389/4.312	6.511/2.907	7.245/2.612	11.132/1.700
FiQA	84.395/1.000	44.407/1.900	10.359/8.147	17.425/4.843	17.698/4.769	31.363/2.691
NFCorpus	93.469/1.000	50.331/1.857	10.628/8.795	18.331/5.099	18.027/5.185	32.949/2.837
TREC-COVID	90.937/1.000	46.835/1.942	10.663/8.529	18.139/5.013	18.970/4.794	33.899/2.683
SCIDOCS	65.554/1.000	34.901/1.878	9.163/7.154	14.872/4.408	15.127/4.334	26.696/2.456
Avg.	61.974/1.000	33.230/1.808	8.269/6.857	13.627/4.189	14.055/4.041	24.538/2.337

Analysis. Without routing, ReState speeds up DL19 by $2.556\times$ and NFCorpus by $5.185\times$; routing raises its six-dataset mean from $4.041\times$ to $6.857\times$. Longer documents increase the projections and state updates eliminated by reuse. Short inputs leave less computation over which to amortize request handling and memory movement. Double buffering overlaps part of the transfer with scoring but cannot remove all fixed work. Routing adds to this saving by reducing the number of pairs, at the quality cost shown above.

A.14 Dataset-Level BERT Transfer

Checking checkpoint consistency. Table 18 expands Figure 6 to identify differences hidden by the model averages. Each speedup uses its own checkpoint's Full reference under the shared transfer protocol. A transferable execution saving can coexist with checkpoint-dependent quality recovery.

Table 18: BERT transfer by dataset.

		Full	ReState		ReState-H	
Model	Dataset	nDCG	nDCG	Speedup	nDCG	Speedup
MiniLM	DL19	0.731	0.704	2.607×	0.715	1.773×
	DL20	0.673	0.652	2.673×	0.653	1.828×
	FiQA	0.334	0.314	4.205×	0.316	2.574×
	NFCorpus	0.305	0.245	4.777×	0.259	2.762×
	TREC-COVID	0.710	0.701	4.226×	0.695	2.531×
	SCIDOCS	0.164	0.157	3.696×	0.159	2.303×
MonoBERT	DL19	0.691	0.681	2.572×	0.670	1.806×
	DL20	0.637	0.628	2.589×	0.623	1.837×
	FiQA	0.337	0.313	3.103×	0.315	2.221×
	NFCorpus	0.336	0.324	3.328×	0.327	2.324×
	TREC-COVID	0.693	0.651	3.035×	0.660	2.175×
	SCIDOCS	0.162	0.159	2.940×	0.158	2.094×
nBoost BERT-base	DL19	0.683	0.686	2.314×	0.695	1.659×
	DL20	0.650	0.631	2.331×	0.636	1.684×
	FiQA	0.336	0.308	2.850×	0.316	2.075×
	NFCorpus	0.333	0.314	3.110×	0.319	2.206×
	TREC-COVID	0.677	0.676	2.838×	0.672	2.059×
	SCIDOCS	0.159	0.157	2.710×	0.158	1.970×

Analysis. ReState accelerates every model–dataset pair; the range is 2.314–4.777×. The saving persists across both BERT-base checkpoints, supporting reuse beyond the compact backbone. Its variation across datasets follows the amount of document computation available to skip. Refinement changes the update set after joint computation and improves some rankings while worsening others. Therefore, a shared absolute-position interface permits execution transfer without guaranteeing identical quality loss across checkpoints.

A.15 Dataset-Level RoPE Transfer

Locating losses under offline selection. Table 19 expands the rotary-position ReState results to test whether the mean hides domain-specific losses. Adapters and training-free reuse share source caches and the update ratio. If key alignment were sufficient, large remaining quality losses would not persist after positions are aligned.

Table 19: Rotary-position transfer with ReState.

		Full	ReState		+ adapter	
Model	Dataset	nDCG	nDCG	Speedup	nDCG	Speedup
BCE ModernBERT	DL19	0.639	0.601	2.457×	0.610	2.454×
	DL20	0.543	0.543	2.492×	0.539	2.496×
	FiQA	0.256	0.241	3.534×	0.242	3.534×
	NFCorpus	0.265	0.260	3.925×	0.260	3.926×
	TREC-COVID	0.636	0.671	3.606×	0.662	3.606×
	SCIDOCS	0.081	0.087	3.463×	0.084	3.464×

Analysis. On DL19, ReState reaches 0.601 nDCG versus Full's 0.639, and adaptation raises it to 0.610. This remaining loss shows that aligned keys do not fully recover query-conditioned relevance computation. Mean pooling also reads final document states that remain partly cached. Adapters alter the online transformations without adding a full document forward pass. Their uneven gains across datasets explain why the mean does not improve over training-free ReState.

Checking recovery from joint computation. Table 20 tests whether the shallow joint path and adapters reduce those losses across domains. The two-layer checkpoint and pure reader-mass selec-

tor are the RoPE variant specified above. Recovery should be judged against the same checkpoint's Full result, not an unrelated model's score.

Table 20: Rotary-position transfer with `ReState-H`.

		`Full`	`ReState-H`		+ adapter	
Model	Dataset	nDCG	nDCG	Speedup	nDCG	Speedup
BCE ModernBERT	DL19	0.639	0.585	2.198×	0.615	2.199×
	DL20	0.543	0.511	2.213×	0.523	2.213×
	FiQA	0.256	0.229	2.909×	0.232	2.909×
	NFCorpus	0.265	0.247	3.146×	0.260	3.146×
	TREC-COVID	0.636	0.664	2.958×	0.648	2.958×
	SCIDOCS	0.081	0.081	2.866×	0.081	2.866×

Analysis. On DL19, adaptation raises `ReState-H` from 0.585 to 0.615 nDCG at 2.199× speedup, while Full reaches 0.639. The joint prefix supplies query-conditioned states before selective execution. Folded adapters can improve the subsequent transformations at the same update ratio. However, gains are not uniform across datasets, and adapted `ReState-H` still trails basic `ReState` in mean quality. These results support an adaptable execution strategy, rather than universal quality preservation under rotary positions.