QUERYLENS: LIGHTWEIGHT QUERY RECALIBRATION FOR REUSED KV CACHES

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-context language model serving increasingly relies on key–value (KV) cache reuse to avoid repeated prefill computation over recurring documents. Yet caches compiled independently and assembled under a new request cannot fully reproduce the contextual interactions of joint prefill and require correction. Existing approaches typically operate on the cache side through selective token recomputation, aiming to make reused caches more consistent with their full-context counterparts. However, cache-side repair can increase time to first token (TTFT) as the recomputation budget grows, while the native query may remain mismatched with the realized cache. To address this issue, we propose **QueryLens**, a query-correction method that introduces a new query-side perspective on KV cache reuse. It learns a lightweight, shared transformation of queries to improve how reused KV states are read. Moreover, QueryLens supports token-level and attention-head-level calibration and can be combined with existing KV cache methods to further improve performance; its statically folded head-level variant incurs negligible measured TTFT overhead. EExperiments on three tasks show that adding QueryLens to A3 at 10% recomputation raises the three-task macro-F1 on Llama-3.1-8B-Instruct from 26.5 up to a notable 69.1.

1 INTRODUCTION

Large language models are increasingly used in retrieval-augmented generation, multi-document question answering, and agentic workflows whose prompts contain long documents that recur across requests. The prefill stage processes this context before decoding can start, so its cost appears directly in time to first token (TTFT). Key–value (KV) caching is consequently a central serving optimization: a previously computed document can be reused instead of being encoded again for every request. Conventional prefix caching is effective when requests share an exact prefix, but retrieval systems routinely reorder documents, change the surrounding instruction, or insert query-specific text. These small changes invalidate an otherwise reusable prefix and leave a large gap between the available reuse opportunity and what exact-prefix matching can exploit (Kwon et al., 2023; Zheng et al., 2024; Gim et al., 2024; Liu et al., 2024; Cheng et al., 2025; Qin et al., 2025).

Position-independent cache reuse relaxes this restriction by compiling immutable document chunks independently and linking their states under a later request context. This decoupling enlarges the reuse space, but it also changes the memory distribution seen by the transformer. Specifically, reuse no longer depends on a complete context formed through the original joint prefill, but rather on states stitched together after the fact, so there is a trade-off between reuse efficiency and contextual fidelity. A document compiled in isolation has not participated in the cross-document interactions that would occur during a joint prefill. Rotary positional realignment fixes the coordinate system, but it cannot recreate the contextual interactions that were absent during compilation (Su et al., 2024). Naively concatenated states can therefore produce a useful but imperfect memory: answer-bearing content may remain, yet the model's native query may no longer access it as in joint prefill. Here, *query* means the vector computed inside an attention head to score cached keys, and the *realized cache* is the KV memory assembled for this request and used during decoding.

Most existing approaches address this problem on the cache side. CacheBlend selectively recomputes tokens at link time (Yao et al., 2025); EPIC and LegoLink target boundary artifacts with a small, often static repair budget (Hu et al., 2025); A3 allocates recomputation using query–document

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.

relevance (Zhou et al., 2025); and KVPacket wraps independently compiled blocks with learned boundary representations to avoid link-time recomputation (Chen et al., 2026). These methods make important progress: they decide which tokens to recompute, how to compose blocks, or how to represent a boundary. The native query projection used to access the resulting cache is typically left unchanged. Selective repair also adds serving-time work, and its TTFT cost grows with the amount of context repaired during serving.

This paper studies the complementary query-side opportunity. Once a reusable cache has been materialized, each attention head still uses the model's native query projection, which was optimized for jointly processed context. Its scores over independently compiled keys may differ, even when useful values remain cached. We ask whether adjusting this readout can recover the part of full-context behavior still supported by the realized cache. Query correction can reweight retained values, while information never stored in the cache is unavailable. It therefore complements cache-side repair.

Figure 1(a) shows the joint-prefill reference of one attention head and Figure 1(b) the same head on the realized cache: joint prefill defines a full-context reference, whereas the native query on the realized cache weights the same values differently. QueryLens modifies only the query, thereby steering the model's next-token output toward the full-context reference without rebuilding the memory or synthesizing absent content.

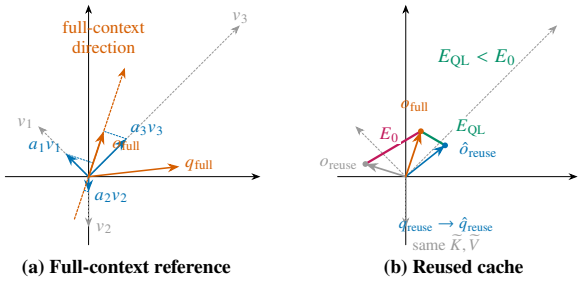


Figure 1: Illustration of query correction on one attention head. **(a)** Joint full-context reference o_{full}. **(b)** On the realized cache, the native query produces o_{reuse} with error E_0; changing only q_{reuse} to $\widehat{q}_{\text{reuse}}$ produces $\widehat{o}_{\text{reuse}}$ with $E_{\text{QL}} < E_0$, while $\widetilde{K}$ and $\widetilde{V}$ remain fixed. The highlighted segments compare both outputs with the same full-context reference.

These observations motivate **QueryLens**, a lightweight query-side repair framework for reused KV caches. QueryLens-T prepends learned soft tokens to the query prompt and corrects the query representation at token level; QueryLens-H applies a low-rank residual directly to each selected attention head. Both are trained offline by self-distillation from the full-context model, independently of the cache method used at deployment. Once trained, either variant can act alone on a reused cache or after a cache method materializes its memory, without a second KV-recomputation stage. Their positive results show that the mismatch can be repaired through two query-side interfaces. We focus our study on QueryLens-H because its per-layer/per-head intervention is more fine-grained, adds no token, and can be folded into the native query projection before serving. Unless stated otherwise, *QueryLens* below denotes QueryLens-H.

We summarize our contributions as follows: (1) to our knowledge, we are the first to identify query–cache mismatch as a distinct error source in position-independent KV reuse, where useful information remains cached but the native query is miscalibrated for the realized cache; (2) we establish that this mismatch is real and recoverable through a fixed-cache geometry and oracle that separate query-reachable from query-irreducible distortion; (3) we propose **QueryLens** with two effective query-side corrections—token-level QueryLens-T and head-level QueryLens-H—that share a full-context self-distillation objective; and (4) we systematically study the more fine-grained QueryLens-H across models, tasks, budgets, and cache methods, showing improved realized-cache performance while static folding removes additional query-correction latency at serving time.

2 Related Work

2.1 KV-cache reuse for long-context serving

Modern LLM serving separates prompt prefill from autoregressive decoding and reuses intermediate KV states to avoid repeated work (Vaswani et al., 2017; Kwon et al., 2023; Zheng et al., 2024). Prefix caching is highly effective for repeated prefixes, but exact-prefix matching is brittle when retrieval systems reorder documents or surround them with request-specific instructions.

Position-independent and context-independent cache systems instead compile document blocks independently and link them at request time. This expands the reuse space while exposing a new fidelity problem: an independently compiled state is not generally the state produced by a joint prefill under the final context of the request.

2.2 CACHE-SIDE REPAIR AND CACHE COMPOSITION

Recent work primarily improves how reusable memory is constructed or assembled. CacheBlend performs selective link-time recomputation based on representation deviations (Yao et al., 2025). EPIC/LegoLink uses a boundary-oriented repair policy motivated by attention-sink behavior (Hu et al., 2025; Xiao et al., 2024). A3 and newer query-guided systems use the request query to allocate memory-side repair (Zhou et al., 2025; Wang et al., 2026; Yan et al., 2026). KVPacket follows a recomputation-free route by wrapping independent blocks with learned boundary soft tokens (Chen et al., 2026); other systems modify the block attention interface or train auxiliary cache models (Ma et al., 2025; Yang et al., 2025c;b) for reuse.

These methods address two cache-side questions: which entries should be recomputed, and how independently compiled entries should be composed. Their shared trade-off lies in the tension among cache fidelity, recomputation cost, and TTFT. None of them reconsiders the native query projection used to access the resulting cache.

2.3 QUERY-AWARE ADAPTATION AND THE POSITIONING OF QUERYLENS

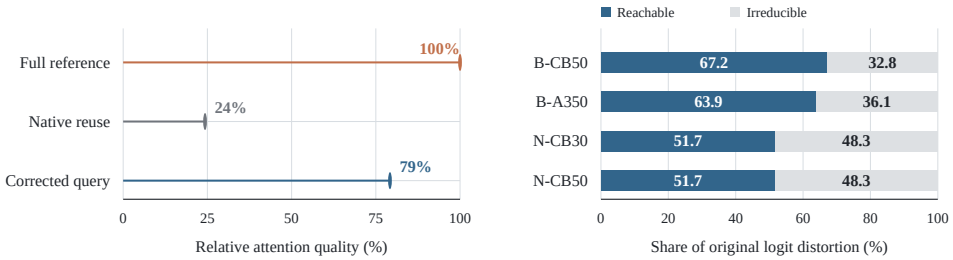
Query-aware cache fusion methods use the request query to estimate relevance and decide where memory-side work should be spent: their query is a signal for *which memory to repair*, which remains a cache-side decision. QueryLens instead corrects how the native query accesses an already realized cache, making it a query-side correction rather than a policy for allocating cache-side computation. Parameter-efficient adaptation provides useful design ingredients: prefix tuning and prompt tuning keep the backbone frozen while learning small continuous parameters (Li & Liang, 2021; Lester et al., 2021), and low-rank residuals (Hu et al., 2022) offer a compact way to alter a projection. These methods, however, generally adapt a model for a task rather than define an amortized query correction for a position-realigned reusable cache. QueryLens-T repurposes a token interface specifically for this mismatch, while QueryLens-H uses a more fine-grained head interface that preserves sequence length. Both leave document-cache contents unchanged; the primary QueryLens-H can therefore be attached after CacheBlend, A3, EPIC, or KVPacket.

3 MOTIVATION: QUERY–CACHE MISMATCH IN KV REUSE

Cache reuse changes query access. With the realized cache held fixed, a different attention query can change which cached states receive weight. This lets us test how much of the reuse error can be reduced through the readout of existing memory. For one head, joint prefill and reuse produce $z_{\text{full}} = q_{\text{full}} K_{\text{full}}^{\top}/\sqrt{d_h}$ and $z_{\text{reuse}} = q_{\text{reuse}} \widetilde{K}^{\top}/\sqrt{d_h}$. Position alignment corrects token coordinates, but it cannot restore the cross-document interactions absent when blocks were compiled independently. Consequently, reuse changes both the available keys and how a native query scores them; earlier changes can also shift queries in deeper layers. The relevant object is therefore a query calibrated to the *realized* cache, not a reconstruction of q_{full}.

Figure 2(a) gives a concrete fixed-cache test on Llama-3.1-8B-Instruct with A3–10, NIAH, and 8K context. To illustrate how reuse changes the attention pattern, we align the captured question-token attention rows, normalize each row over context keys, and average the attention mass assigned to the reference distribution's top-8 context positions across the selected layers and heads. On these positions, native reuse assigns only 0.243 times the reference attention mass. With the keys fixed, a target-aware per-instance query fit raises this ratio to 0.792, with a Jensen–Shannon divergence of 0.110 from the reference distribution. This shows that the existing cache can support an attention pattern closer to the reference when read differently. See Appendix C.2 for detailed analysis.

Fixed-cache geometry separates query-reachable distortion. Figure 2(b) provides a logit-space diagnostic under a fixed cache, measuring how much native-reuse error can be removed by changing



(a) Attention-space mismatch

(b) Fixed-cache error decomposition

Figure 2: Evidence for query–cache mismatch. **(a)** Native reuse versus a target-aware, per-instance query fit on fixed A3–10 keys. **(b)** Squared logit distortion decomposes into query-reachable and fixed-key-irreducible parts across Biography (B), NIAH (N), CacheBlend (CB), and A3 settings.

the query while keeping the realized keys fixed. To draw this panel, we project the aligned full-context target logits onto the column space of each realized key matrix; the blue segment shows the error removed by the projection, and the gray segment shows the residual.

Let $\widetilde{K}$ denote the realized keys, q the native reuse query, and $z^\star$ the full-context target logits in the same token coordinates. Define $\bar{K} = \widetilde{K}/\sqrt{d_h}$ and let $P_{\bar{K}} = \bar{K}\bar{K}^+$ be the orthogonal projection onto the column space of $\bar{K}$. Since changing the query with $\widetilde{K}$ fixed can move the logits only within this column space, the fraction of squared logit error removable by the query is

$$R_Q = 1 - \frac{\|(I - P_{\bar{K}})\, z^\star\|_2^2}{D_0}, \qquad D_0 = \|\bar{K}q - z^\star\|_2^2, \tag{1}$$

where D_0 is the squared error under native reuse. Across the Biography/NIAH and CacheBlend/A3 settings, the fixed-cache oracle removes 51.7–67.2% of this distortion. Among the 64 queries, even the least recoverable query permits 41.1% of the distortion to be removed. Meanwhile, 32.8–48.3% lies outside the realized-key column space and cannot be removed by changing the query alone. This decomposition shows that reuse error contains both a query-reachable component and a component that requires changing the cache itself. We analyze this fixed-cache decomposition in Appendix B, including the per-query recoverability distributions in Figure 5.

Together, Figures 2(a) and 2(b) show that reuse changes how retained content is accessed and that a substantial part of the resulting error remains query-reachable. The independent output-space intervention in Figure 6 reaches the same conclusion at the next-token distribution: free per-layer edits remove 99.6% of teacher KL and the learned QueryLens-H removes 85.0%. Taken together, this evidence suggests that a shared query correction may improve access to a realized cache.

4 QUERYLENS

The motivation experiments show that a realized cache retains a query-reachable degree of freedom, and that correcting only the query can realign it with the original distribution. This leads to the following general formulation:

With a frozen backbone, can query correction recover reachable full-context behavior from a fixed KV cache without rebuilding it?

Problem formulation. For a request x, a cache method first materializes $\widetilde{\mathcal{C}}(x) = (\widetilde{K}, \widetilde{V})$. The frozen backbone then produces its native query q for the realized cache. QueryLens keeps the realized cache fixed and applies a shared query map $\widehat{q} = \Phi(q; \phi)$. The map is learned offline and applied after cache build; the next two paragraphs give its token-level and head-level realizations.

To state the common objective, consider one head and one query position. Let $z^\star$ be the full-context target logits expressed in the realized-cache token coordinates. When the cache contains a token

subset or aggregate, a fixed alignment operator supplies these coordinates. Writing $\bar{K} = \widetilde{K}/\sqrt{d_h}$, we seek a corrected query whose logits on the fixed keys approach this target:

$$\min_{\widehat{q}} \frac{1}{2}\|\bar{K}\widehat{q} - z^\star\|_2^2. \tag{2}$$

Proposition 1 (Fixed-cache reachability) Let $\bar{K} = \widetilde{K}/\sqrt{d_h}$ and let $P_{\bar{K}} = \bar{K}\bar{K}^+$, where $\bar{K}^+$ is the Moore–Penrose pseudoinverse, project orthogonally onto the column space of $\bar{K}$. For any minimizer q^*, the fitted logits and minimum residual satisfy

$$\bar{K}q^* = P_{\bar{K}}z^\star, \qquad \min_u \|\bar{K}u - z^\star\|_2^2 = \|(I - P_{\bar{K}})z^\star\|_2^2. \tag{3}$$

Thus changing only the query can recover exactly the component of the target logits in the realized-key column space; the orthogonal component cannot be reduced while the cache remains fixed.

The proposition separates the component of query–cache mismatch that query correction can address from the residual that requires changing the cache contents. It also explains why the objective above is a useful diagnostic but not a deployment algorithm: solving it separately for every request would require a cache-dependent optimization at serving time. QueryLens instead amortizes this correction into Φ across representative requests. We analyze this result in Appendix B, where we give the proof, discuss the centered-logit variant, and examine per-query recoverability.

QueryLens-T: token-level query correction. Let $A_\phi = [a_1, \ldots, a_m] \in \mathbb{R}^{m\times d}$ be m learned soft query tokens and $X_q = [x_1, \ldots, x_n]$ the original query-prompt embeddings. After cache construction, QueryLens-T appends learned soft query tokens to the query-prompt embeddings:

$$X_q^{\mathrm{T}} = [A_\phi; X_q]. \tag{4}$$

Processed only on the query side, this prefix lets later tokens attend to its learned states and recalibrates their hidden representations and per-layer queries toward $(\widetilde{K}, \widetilde{V})$ without changing document K/V. It neither wraps document tokens nor enters the reusable document cache, and it does not trigger cache reconstruction. Only A_ϕ is trained under the shared objective below. We use $m = 8$; these added query tokens cannot be folded into W_Q. QueryLens-T is a flexible token-level query correction, with additional query-token and prompt-computation costs.

QueryLens-H: head-level correction over fixed KV memory. Figure 3 summarizes this primary implementation: QueryLens-H is trained by self-distillation, applied after reusable KV memory is materialized, and changes only the native query projection while keys, values, positions, and sequence layout remain fixed. Solving Eq. 2 independently for every query would require a cache-dependent least-squares solve at serving time. QueryLens-H instead learns a shared query correction from representative requests. The fixed-cache oracle motivates this construction but is not its optimization target. The correction adds a residual to the query already produced by the frozen model. For adapted layer l and head h,

$$\widehat{q}_{l,h} = q_{l,h} + \gamma_{l,h}(q_{l,h}D_{l,h})U_{l,h}, \qquad \gamma_{l,h} = \tanh(g_{l,h}), \tag{5}$$

where $D_{l,h} \in \mathbb{R}^{d_h\times r}$ and $U_{l,h} \in \mathbb{R}^{r\times d_h}$ with $r \ll d_h$. The two factors provide a low-dimensional update subspace, while the scalar gate controls the strength of the residual. The parameters are attached only to selected layers and heads, so the storage cost is approximately $2L_sHrd_h + L_sH$. The parameterization is the smallest intervention that changes how the same memory is addressed without changing what it contains. The update is additive and low rank, so the correction is confined to a shared r-dimensional subspace per head rather than an unconstrained reparameterization of the projection. The per-head gate gives the learned correction layer- and head-specific strength. This is finer-grained than QueryLens-T: the update can be localized to selected layers and heads, while QueryLens-T exposes a shared token interface to the subsequent query path.

The residual corrects the query mapping rather than re-encoding or modifying the cache. Its parameters are learned from the distribution of native queries and realized caches encountered during training, but the deployed map itself does not modify $\widetilde{K}$, $\widetilde{V}$, positions, or token count. The same QueryLens parameters therefore compose with different cache methods through a shared materialized-KV/query interface, without method-specific retraining. Unlike the shared token interface of QueryLens-T, this per-layer/per-head parameterization directly controls where and how

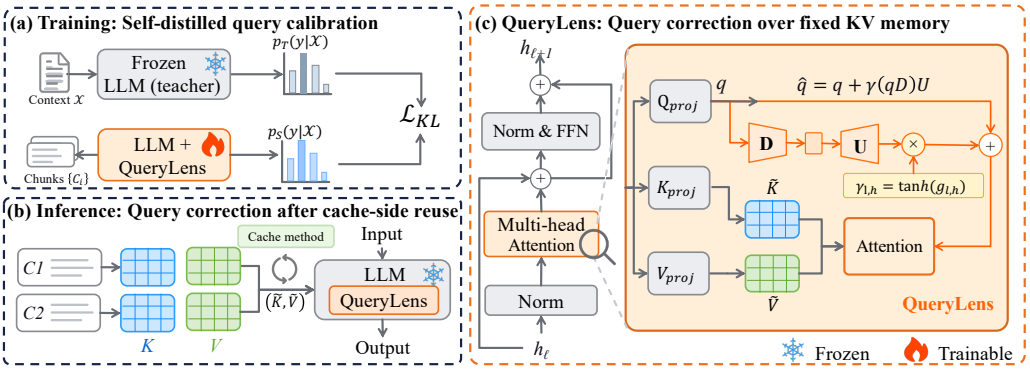


Figure 3: Overview of the primary QueryLens-H implementation. **(a)** Training: self-distilled query calibration, in which a frozen full-context teacher provides the target distribution for the realized-cache student. **(b)** Inference: query correction after cache-side reuse, where the reuse method first materializes $(\widetilde{K}, \widetilde{V})$ and QueryLens-H then corrects only the native query. **(c)** QueryLens-H: query correction over fixed KV memory, implemented as a low-rank residual at the query projection while the keys, values, positions, and sequence layout remain unchanged.

strongly the query is corrected, adds no token, and can be folded into the query projection. Consequently, H avoids the extra query tokens of T and its static path has no measurable additional query-side TTFT in the matched measurements of Appendix F.1. Both variants provide useful query-side repairs, but H is the primary method for the analyses and deployment experiments that follow.

Training: self-distilled query calibration. For each training request, we run the frozen backbone once with the full context to obtain teacher next-token distributions p_T. A second pass independently compiles the original document chunks and assembles their reusable KV states, without invoking a downstream cache-repair method; this student setup is independent of the deployment cache method and supplies $(\widetilde{K}, \widetilde{V})$. The backbone, document states, keys, and values are frozen. QueryLens-H updates $\phi_{\mathrm{H}} = \{D, U, g\}$, while QueryLens-T updates only $\phi_{\mathrm{T}} = \{A_\phi\}$, using the same teacher–student distillation:

$$\mathcal{L}_{\mathrm{KL}}(\phi) = \frac{1}{|T_q|} \sum_{t \in T_q} D_{\mathrm{KL}}(p_T(\cdot|x_{\le t}) \,\|\, p_S(\cdot|x_{\le t}; \phi)) \,. \tag{6}$$

The full-context run supplies the target behavior, while the independently compiled cache exposes the mismatch to be corrected. Distillation requires no task labels and is averaged over query positions T_q. Cache methods and budgets are deployment choices, not training conditions: for a given model/task profile, one QueryLens checkpoint is trained once on the original training requests and reused unchanged across compatible cache methods and budgets.

Inference: query correction after cache-side reuse. At inference, a cache method first materializes $(\widetilde{K}, \widetilde{V})$ according to its own procedure. QueryLens-H then corrects the query inside the model's native order: the query projection produces the native query, the residual correction is added to it, query-dependent normalization and positional rotation follow unchanged, and attention consumes the corrected query together with the materialized keys and values, after which decoding proceeds as usual. For an adapted layer l and head h, the attention output separates the two contributions:

$$o_{l,h} = \mathrm{softmax}\left(\frac{\widehat{q}_{l,h}\widetilde{K}_{l,h}^{\mathsf{T}}}{\sqrt{d_h}}\right)\widetilde{V}_{l,h}, \tag{7}$$

where QueryLens supplies the corrected query, while the realized cache supplies the keys and values. Nothing on the key or value path is touched, so the content, order, positions, and length of the memory stay exactly as the cache method produced them.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models, datasets, and cache methods. We evaluate Llama-3.1-8B-Instruct (Dubey et al., 2024) and Qwen3-4B-Instruct-2507 (Yang et al., 2025a) on four datasets: Synthetic Biography, NIAH/RULER retrieval (Hsieh et al., 2024), HotpotQA (Yang et al., 2018), and MultiFieldQA-en from LongBench (Bai et al., 2024). CacheBlend and A3 use matched recomputation-ratio sweeps; EPIC uses its native boundary-token budget (or explicitly marked matched-token runs); KVPacket uses its native recomputation-free setting.

Hardware, software, and evaluation. Experiments run on NVIDIA A100-SXM4 80GB GPUs with PyTorch 2.8.0+cu128 (CUDA 12.8). Biography, HotpotQA, and MultiFieldQA-en are evaluated with token-level F1; NIAH uses exact-match task accuracy. The three-task macro-F1 reported throughout is the unweighted mean of Biography F1, NIAH exact-match accuracy, and HotpotQA F1; MultiFieldQA-en is reported separately. We report paired quality deltas, harmful-correction rate, median/p95 TTFT, cache-side recomputation FLOPs, and QueryLens arithmetic. Every pair uses the same cache method and native-query baseline; full prompt, split, decoding, and timing protocols are in Appendix A.

Here, the *cache-only* baseline disables all query-side parameters and uses $\Phi_{\mathrm{Id}}(q) = q$: the native query accesses the realized cache produced by the same cache method. It is therefore a cache-only/native-query baseline, not a full-context reference.

5.2 QUERY-SIDE REPAIR AT TWO GRANULARITIES

We first test whether the fixed-cache mismatch is repairable through distinct query interfaces. Table 1(c) shows gains from both; QueryLens-H is stronger at low budgets and nearly matches QueryLens-T at NIAH/CacheBlend–50, while providing per-layer/per-head control without extra tokens. Thus the table provides a controlled validation that both granularities are useful, rather than treating T as an architectural alternative without evidence. We therefore use H for the analyses.

5.3 MAIN RESULTS

5.3.1 QUERY CORRECTION COMPLEMENTS CACHE-SIDE REPAIR

Figure 4 compares the native-query baseline and QueryLens-H across Llama-3.1-8B-Instruct and Qwen3-4B-Instruct-2507. Columns show Biography, HotpotQA, NIAH, and MultiFieldQA-en from left to right; Llama-3.1-8B-Instruct occupies the first row and Qwen3-4B-Instruct-2507 the second. The horizontal positions use each cache method's native budget labels; KVPacket is shown separately because it has no recomputation axis.

The largest gains appear under aggressive reuse. With A3 at 10% recomputation, the three-task macro-F1 over Biography, NIAH, and HotpotQA rises from 26.5 to 69.1 on Llama and from 18.4 to 54.9 on Qwen; Llama Biography alone rises from 6.5 to 98.8. MultiFieldQA-en is reported separately: at CacheBlend–10, its F1 rises from 43.16 to 48.16 on Llama and from 34.59 to 43.74 on Qwen. The same checkpoint for each model/task profile is reused across cache methods, so these gains require no method-specific retraining.

5.3.2 BUDGET TRENDS

The full paired matrix in Appendix D shows that gain size varies non-monotonically with budget. As a cache method recomputes more tokens, the same query correction acts on a different cache.

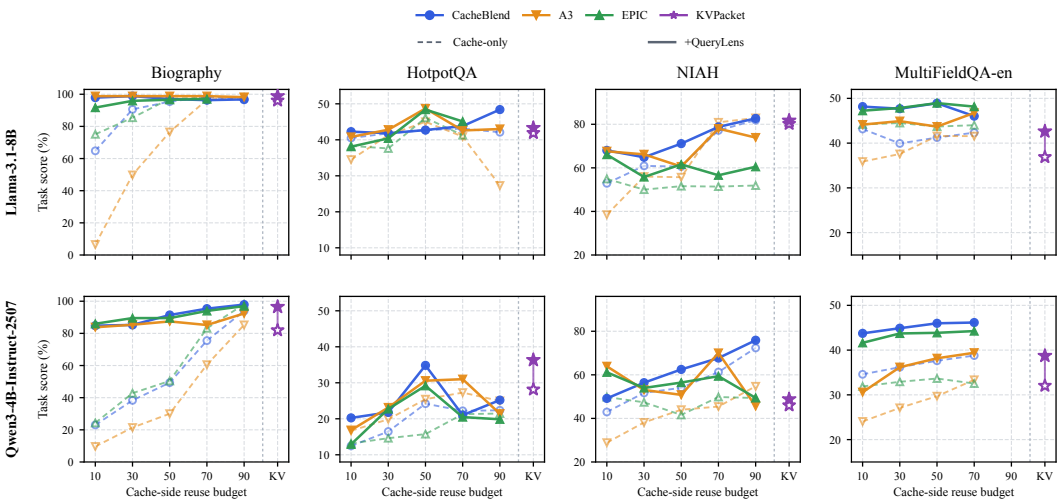


Figure 4: QueryLens-H across models, tasks, and cache methods. Dashed open markers are cache-only baselines and solid filled markers the corresponding QueryLens-H combination.

Table 1: Controlled QueryLens ablations (score, %). *cache-only* disables the query correction and uses the native query on the same realized cache. **(a)** Head-level residual rank; **(b)** residual scale; **(c)** token- versus head-level query correction. Panel (c) retains the main-table cache-only baseline and QueryLens-H results.

(a) Residual rank

Setting	*cache-only*	$r = 1$	$r = 2$	$r = 4$	$r = 8$	Best Δ
Qwen / A3–10	28.85	**57.64**	56.18	56.53	54.65	+28.79
Llama / CB–10	52.92	65.49	63.51	**66.43**	61.39	+13.51

(b) Fixed scalar residual scale

Setting	*cache-only*	$\alpha = 0.25$	$\alpha = 0.50$	$\alpha = 0.75$	$\alpha = 1.00$	Best Δ
Qwen / A3–10	28.85	53.15	55.56	**58.32**	56.53	+29.47
Llama / CB–10	52.92	53.73	56.57	**68.01**	66.43	+15.09

(c) Query-side repair granularity

Setting	*cache-only*	QueryLens-T	QueryLens-H
Biography / CB–10	64.80	79.46	**97.91**
Biography / CB–50	95.40	96.55	**97.01**
NIAH / CB–10	52.90	56.37	**68.02**
NIAH / CB–50	60.60	70.93	**71.12**

5.4 Ablations and Analysis

5.4.1 QueryLens-H ablations: capacity, strength, and causal controls

Controlled setup. Table 1 changes one axis while fixing the cache method, full-context teacher, query split, decoding path, and cache-only baseline. Panels (a–b) use low-repair NIAH settings; panel (c) uses the same Llama/CacheBlend task–budget settings for both query-side interfaces.

Rank $r = 4$ balances capacity and stability. The rank sweep is non-monotonic: $r = 1$ is strongest on Qwen and $r = 4$ on Llama, while $r = 8$ gives no further gain. We therefore retain $r = 4$ as the main setting: it is large enough to express the dominant head-level correction on the larger model, but remains a compact per-head subspace that constrains unstable directions rather than freely reparameterizing W_Q.

Correction magnitude is an independent control. Panel (b) fixes the rank-4 direction and varies only $\widehat{q} = q + \alpha\Delta q$. The shared optimum $\alpha = 0.75$ on both models shows that the learned direction alone is not the method: bounded strength is a separate safeguard against over-correction.

Query-only control and two query-side interfaces. Panel (c) shows QueryLens-T gains in all four settings, so query-side repair is useful beyond the head parameterization. H is stronger at low budgets, comparable at CacheBlend–50, and additionally supports layer/head control and folding. Table 6 in Appendix E further isolates the query-only effect on independently compiled caches: the same query correction improves both models and both tasks without any cache-side recomputation. This establishes necessity of the query-side intervention while also respecting its information boundary: QueryLens can change how existing keys and values are addressed, but cannot synthesize content absent from the KV cache.

5.4.2 CLEAN-CONTEXT Q-LORA CONTROL

To separate cache-mismatch repair from generic query-projection adaptation, we train a clean-context Q-LoRA control under the same rank, per-head parameter count, initialization, optimizer, training queries, and full-context teacher as QueryLens-H. The only change is the student view: Q-LoRA reads the jointly prefilled context KV and never observes independently compiled reused-cache states during training. In the two evaluated settings, this control is nearly unchanged on Llama (52.40 versus 52.92) and yields a modest gain on Qwen (32.33 versus 28.85), whereas QueryLens-H improves the same reused-cache settings to 66.43 and 56.53. The full comparison is reported in Appendix E.2, Table 7. The gap rules out task-level query adaptation as the main explanation: the useful signal is the query correction learned from the cache states it must actually consume.

5.5 QUERYLENS-H EFFICIENCY: NEAR-ZERO LATENCY VIA STATIC QUERY-PROJECTION FOLDING

QueryLens-T adds eight query-side soft tokens in our comparison. QueryLens-H adds no token, and its low-rank residual need not remain online. For a fixed profile, define and fold its linear map into the native projection:

$$T_{l,h} = I + \gamma_{l,h} D_{l,h} U_{l,h}, \quad W'_{Q,l,h} = T_{l,h}^{\mathsf{T}} W_{Q,l,h}, \quad b'_{Q,l,h} = T_{l,h}^{\mathsf{T}} b_{Q,l,h}. \tag{8}$$

This is algebraically identical up to rounding and executes through the native linear layer, with no hook, cache token, or recomputation. All QueryLens-H results use this folded version under matched serving conditions.

Across 4K–32K, folded QueryLens-H changes median TTFT by -0.27% to $+0.19\%$; the per-setting measurements are in Appendix F.4, Table 8. A paired run with newly trained profiles gives -0.027% to $+0.244\%$ (at most 1.002 ms), while the unfused hook adds approximately 11–12 ms.

6 LIMITATIONS AND CONCLUSION

Limitations. QueryLens cannot synthesize values absent from a reused cache or recover logits outside the key-induced subspace. Gains depend on cache geometry, and larger cache-side updates can change the paired effect across budgets. The implementation assumes standard KV tensors and an intact query path; altered interfaces require recalibration, and dynamic profile selection must be timed. These limitations define QueryLens as a complement to cache-side reconstruction: the cache supplies the content, while query correction changes how that content is addressed.

Conclusion. We identify query–cache mismatch as a distinct source of degradation in KV reuse and establish its query-reachable component under a fixed cache. QueryLens-T and QueryLens-H show that this mismatch can be repaired through both token- and head-level query interfaces. We focus on the more fine-grained QueryLens-H, whose per-layer/per-head correction raises the reported ceilings of CacheBlend, A3, EPIC, and KVPacket across two model families. Its rank and magnitude controls provide useful stability constraints, while the query-only control confirms that the gain is not caused solely by cache-side recomputation. Finally, static folding retains the correction without measurable context-dependent TTFT, establishing fine-grained query correction as a practical complement to cache-side repair in reusable-context serving.

AI USE STATEMENT

We used generative AI tools solely for language editing and readability improvement of selected passages. No generative AI tools were used for research ideation, experimental design, data generation or analysis, or drafting of technical content. All AI-assisted edits were reviewed and verified by the authors. We take full responsibility for the accuracy, integrity, and content of this manuscript.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Chuangtao Chen, Grace Li Zhang, Xunzhao Yin, Cheng Zhuo, Bing Li, and Ulf Schlichtmann. Kv packet: Recomputation-free context-independent kv caching for llms. *arXiv preprint arXiv:2604.13226*, 2026.

Yihua Cheng, Yuhan Liu, Jiayi Yao, et al. Lmcache: An efficient kv cache layer for enterprise-scale llm inference. *arXiv preprint arXiv:2510.09665*, 2025.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 325–338, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? In *Proceedings of the First Conference on Language Modeling*, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. Epic: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025. Also available as arXiv:2410.15332.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059, 2021.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, pp. 4582–4597, 2021.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yujian Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56, 2024.

Dongyang Ma, Yan Wang, and Lan Tian. Block-attention for efficient prefilling. *arXiv preprint arXiv:2409.15355*, 2025.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation—a kvcache-centric architecture for serving llm chatbot. In *Proceedings of the 23rd USENIX Conference on File and Storage Technologies*, pp. 155–170, 2025.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

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*, 2026.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2024.

Jianxin Yan, Zeheng Qian, Wangze Ni, Zhitao Shen, Zhiping Wang, Haoyang Li, Jia Zhu, Lei Chen, and Kui Ren. Qcfuse: Query-centric cache fusion for efficient rag inference. *arXiv preprint arXiv:2604.08585*, 2026.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Bin Yang, Qiuyu Leng, Jun Zeng, and Zhenhua Wu. Cacheclip: Accelerating rag with effective kv cache reuse. *arXiv preprint arXiv:2510.10129*, 2025b.

Jingbo Yang, Bairu Hou, Wei Wei, et al. Kvlink: Accelerating large language models via efficient kv cache reuse. *arXiv preprint arXiv:2502.16002*, 2025c.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

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 retrieval-augmented generation with cached knowledge fusion. In *Proceedings of the 20th European Conference on Computer Systems*, 2025. doi: 10.1145/3689031.3696098.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Yuechi Zhou, Yi Su, Jianxin Zhang, et al. A^3: Attention-aware accurate kv cache fusion for fast large language model serving. *arXiv preprint arXiv:2511.17560*, 2025.

A Experimental Protocol and Reproducibility

This appendix records the complete evaluation protocol needed to reproduce the reported comparisons. It separates model and data choices from cache-method details and timing boundaries; deployment mechanics and implementation cost are collected later in Appendix F.

A.1 MODELS, DATA, AND PROMPTS

The primary model releases are Llama-3.1-8B-Instruct and Qwen3-4B-Instruct-2507. We use Synthetic Biography, NIAH/RULER-style retrieval, HotpotQA, and MultiFieldQA-en from LongBench. Biography uses the short-answer prompt and a maximum of 32 generated tokens; HotpotQA uses the long-answer prompt and a maximum of 512 tokens; NIAH follows the evaluator's exact-match protocol. MultiFieldQA-en is evaluated with token-level F1 and reported separately from the Biography/NIAH/HotpotQA three-task macro-F1. Qwen3-4B-Instruct-2507 uses its native chat template and end-of-turn marker, while Llama-3.1-8B-Instruct uses the Llama chat template and its corresponding marker. We never compare checkpoints trained for one model family against the other family.

Unless stated otherwise, the evaluation uses 100 samples, seed 42, deterministic decoding, and the same sample IDs for the cache-only and QueryLens rows. A learned query is trained and selected using query-level splits: all budget variants of a query remain in one split. A random split is reported only as a diagnostic because it can overestimate amortization by placing the same query under another budget in the training set.

For both QueryLens-H and QueryLens-T, each model/task profile is trained once from the original training split, independently of CacheBlend, A3, EPIC, or KVPacket. Its checkpoint is then reused unchanged for all held-out queries, cache methods, and budgets reported for that profile. Cache methods are neither jointly optimized with QueryLens nor used to select per-method parameters.

A.2 CACHE METHODS AND TIMING BOUNDARY

CacheBlend and A3 are evaluated at 10%, 30%, 50%, 70%, and 90% recomputation. EPIC is evaluated at its native boundary-token budgets; if a matched-ratio variant is used, the actual number of recomputed tokens is stated explicitly. KVPacket is evaluated with its native no-recomputation wrapper. For each pair, cache-side reuse, token selection, cache transfer, positional realignment, prompt construction, and decoding are held fixed. TTFT is reported under two timing boundaries:

1. *end-to-end*: cache loading/assembly, cache-side recomputation, query adaptation, prompt forward, and first-token generation;
2. *compiled query*: a static QueryLens transform is merged into W_Q before the timed request, so only the native request path is timed.

The development hook path is retained for debugging but is not used to support a claim of zero overhead. Every timing number should be reported as median and p95 over repeated warm runs, together with GPU type, batch size, and context length.

B FIXED-CACHE GEOMETRY AND RECOVERABILITY ANALYSIS

This section gives the formal fixed-cache argument behind the motivation. It defines the attainable query component, derives the projection decomposition, and then reports oracle measurements that quantify the opportunity without turning per-instance optimization into a training or deployment procedure.

B.1 NOTATION

For one head, let $q \in \mathbb{R}^{d_h}$ be a column query, let $\widetilde{K} \in \mathbb{R}^{N \times d_h}$ be the available key matrix, and let $z^\star \in \mathbb{R}^N$ be the scaled full-context target logits expressed in the realized-cache coordinates. If the cache is a subset or an aggregation of full-context tokens, a fixed alignment operator is understood in this definition. We write $\bar{K} = \widetilde{K}/\sqrt{d_h}$ and define the cache-only distortion $D_0 = \|\bar{K}q - z^\star\|_2^2$. The fixed-cache optimal query is

$$q^* \in \arg\min_{u \in \mathbb{R}^{d_h}} \|\bar{K}u - z^\star\|_2^2. \tag{9}$$

Table 2: Fixed-cache query recoverability R_Q (%).

Cache setting	Layer	Samples	R_Q (%)
Biography / KVPacket	0	16	40.7
Biography / A3–50	1	16	63.9
Biography / CacheBlend–50	2	16	67.2
NIAH / CacheBlend–30	2	16	51.7
NIAH / CacheBlend–50	2	16	51.7

B.2 Projection result

Let $\bar{K}^+$ denote the Moore–Penrose pseudoinverse and $P_{\bar{K}} = \bar{K}\bar{K}^+$ the orthogonal projector onto the column space of $\bar{K}$. A minimizer need not be unique when $\bar{K}$ is rank-deficient, but every minimizer satisfies the normal equations

$$\bar{K}^\mathsf{T}\bar{K}q^* = \bar{K}^\mathsf{T}z^\star$$

and therefore yields the same fitted logits:

$$\bar{K}q^* = P_{\bar{K}}z^\star, \qquad z^\star = P_{\bar{K}}z^\star + (I - P_{\bar{K}})z^\star. \tag{10}$$

The two terms in the second equality lie in orthogonal subspaces: $P_{\bar{K}}z^\star \in \mathrm{col}(\bar{K})$ and $(I-P_{\bar{K}})z^\star \in \mathrm{col}(\bar{K})^\perp$. Since every $\bar{K}u$ lies in $\mathrm{col}(\bar{K})$, the Pythagorean identity gives

$$\min_u \|\bar{K}u - z^\star\|_2^2 = \|(I - P_{\bar{K}})z^\star\|_2^2,$$

so the first component is reachable by changing only the query, while the orthogonal component is irreducible under a fixed key matrix. The attainable fraction is the R_Q defined in Eq. (1): the optimal query removes exactly the reachable component, so $1 - R_Q = \|(I - P_{\bar{K}})z^\star\|_2^2/D_0$, with $R_Q = 1$ when $D_0 = 0$, and therefore $0 \le R_Q \le 1$. In practice, we use a numerically stable least-squares or ridge solve rather than forming $\bar{K}^+$ explicitly. Since softmax is invariant to adding the same scalar to every logit, we also report a centered variant using $C = I - \mathbf{1}\mathbf{1}^\mathsf{T}/N$ and replacing $(\bar{K}, z^\star)$ by $(C\bar{K}, Cz^\star)$.

B.3 The fixed-cache oracle as a mismatch diagnostic

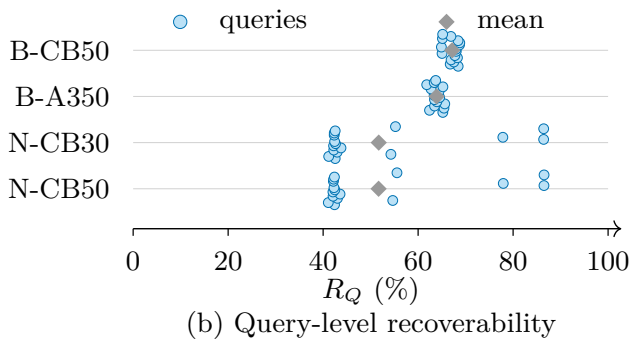
The fixed-cache oracle is an intervention on one cache snapshot, layer, head, and query position. It is used to establish that changing only the query-side query can reduce a nonzero portion of the discrepancy under the realized keys. Neither QueryLens-H nor QueryLens-T is fitted to these per-snapshot solutions: both are trained by the shared distillation objective in Eq. 6. The oracle therefore diagnoses the source of the error rather than predicting a deployed model's generation quality.

The corresponding KVPacket Biography query displacement has norm ratio about 22.2% and query cosine about 0.976, so the optimum does not need to replace the query wholesale.

B.4 Reading the recoverability figure

Figure 2 reports this diagnostic for four settings that span two tasks (Biography and NIAH) and two cache methods (CacheBlend and A3). Figure 2(b) is the decomposition predicted by the geometry above: the full bar is the original distortion of the native query, the blue segment is the query-reachable component that the optimal fixed-cache query removes, and the gray segment is the component that lies outside the column space of the realized keys and is therefore irreducible for *any* query. Figure 5 resolves the same quantity per query: each dot is one of the 16 queries of a setting, placed at its own R_Q on the shared 0–100 scale, and the gray diamond on the row centre line is that setting's mean. The rows and the scale are those of Figure 2(b), so a setting can be read across them directly, and the percentages printed there are exactly the diamond positions here.

Three observations follow. First, the query-side degree of freedom is large and appears consistently across the evaluated settings: the optimal fixed-cache query removes 51.7–67.2% of the distortion depending on the setting, and the weakest of the 64 evaluated queries still recovers 41.1%. Second,



(b) Query-level recoverability

Figure 5: Per-query fixed-cache recoverability. One dot per query at its own R_Q on the shared 0–100 scale, with the setting mean as a gray diamond; rows are those of Figure 2(b).

that freedom is partial: 32.8–48.3% of the distortion is irreducible under the fixed keys, so query-side adaptation cannot substitute for cache-side repair. Third, the recoverable share is largely insensitive to the cache state: the NIAH CacheBlend–30 and CacheBlend–50 settings select different token sets and differ in their global key and value tensors, yet their recovery profiles nearly coincide (51.71% versus 51.68% on average).

The query-level spread is itself informative. Biography is tight (61.8–68.8%, median 65.2%), so its mean does describe a typical query. NIAH is bimodal instead: in each setting 11 of the 16 queries sit near 42%, two near 55%, and three between 77% and 87%. Its mean (51.7%) is therefore pulled above its median (42.5%) by a minority of queries and should not be read as the typical NIAH query. The robust observation is the sign of the effect, not a fixed effect size: every query we evaluated has a non-trivial query-reachable component, while the amount of recoverability varies by query and cache method state.

C Additional Query–Cache Mismatch Diagnostics

The following diagnostics examine the same mismatch at complementary levels. The fixed-cache geometry in Appendix B establishes the query-reachable subspace; the output, attention, and cache-state analyses below show how that freedom appears in model outputs, attention rows, and realized cache geometry.

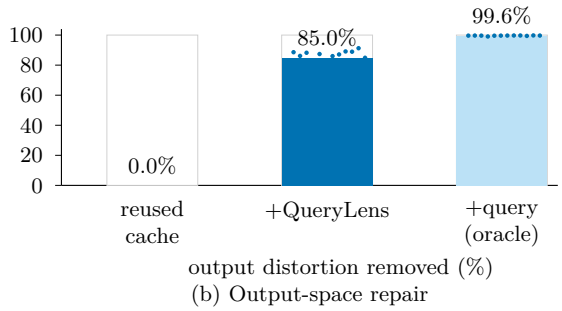
C.1 Output-space Frozen-cache Intervention

Figure 6 is an output-space diagnostic that is separate from both the logit projection in Figure 2(b) and the attention-row analysis in Section C.2. It uses 12 Llama-3.1-8B-Instruct NIAH requests at 8K context with the realized cache held fixed. For each request, we optimize an unconstrained additive edit at every layer's query-projection output for 250 Adam steps to minimize the same next-token KL to the frozen full-context teacher used during distillation. The displayed removed-distortion fraction is $1 - D_{\mathrm{KL}}(p_T \| p_{\mathrm{edit}})/D_{\mathrm{KL}}(p_T \| p_{\mathrm{reuse}})$. The learned QueryLens-H curve is evaluated over the same cache, prompts, teacher, and output-KL metric. This experiment is an intervention test for the existence of query-side mismatch; the free edits are not oracle labels, deployment targets, or generation claims.

C.2 Attention-space Query–Cache Mismatch: Separate Diagnostic

The output-space intervention above establishes mismatch at the model output. This section provides a complementary attention-row view of how cache reuse changes query access.

Reference rows and alignment. The reference $a^\star$ contains all captured question-token rows from a clean full-context prefill of the same request (up to 256 rows), rather than only the last prompt token. The reused row a_{reuse} is captured for the same question-token rows from the real A3 query



(b) Output-space repair

Figure 6: Output-space frozen-cache intervention on 12 NIAH/8K requests. Bars report the fraction of reuse-induced next-token teacher KL removed by deployed QueryLens-H and by non-deployable free per-layer query edits.

Table 3: Aligned attention diagnostic on the Appendix-D setting (Llama-3.1-8B, NIAH, A3–10, 8K). Values are request-level means with 95% bootstrap intervals; all heads and captured question-token rows are averaged within a request before resampling. Top-8 is relative to the full-context reference (1.0 by definition), and JS is in nats.

Row	Top-8 / reference	JS
Full-context reference	1.000 [1.000, 1.000]	0.000
Native reuse	0.243 [0.200, 0.290]	0.371 [0.342, 0.399]
Instance attention fit	0.792 [0.772, 0.811]	0.110 [0.104, 0.117]

at recomputation ratio 0.10, by patching `scaled_dot_product_attention` during the query's prompt prefill, so it is the row the query actually uses and not a reconstruction from a hand-built query. Both rows are evaluated over the same realized keys. The rows are normalized over the context keys only, i.e. they are the attention distribution conditioned on the context; this is applied identically to every query, including the reference. The clean prefill must contain the same prefix as the cache (system preamble plus documents). Omitting the preamble shifts the two key ranges by 30 tokens, which leaves the concentration measures largely unchanged but destroys the rank agreement artificially (Spearman ρ falls from 0.88 to 0.26 for the same data). We report the aligned numbers.

Diagnostic rows. We compare the full-context reference, the native query on the realized keys, and a target-aware instance attention fit initialized at the native query. The fit minimizes Jensen–Shannon divergence to $a^\star$ for 1500 Adam steps at learning rate 0.05, retaining the best iterate. It characterizes the query-access gap under the same realized keys.

Metrics and uncertainty. For each request we average over the five layers, heads, and captured question-token rows, then bootstrap requests (2000 replicates). We report Jensen–Shannon divergence, the attention mass on the full-context top-8 positions divided by the reference mass on those positions. The full-context reference is therefore 1.0 for the relative top-8 metric by construction.

Table 3 reports the aligned rows. Native reuse has relative top-8 quality 0.243 and JS 0.371, whereas the instance fit reaches 0.792 and 0.110. This gap directly visualizes the query–cache mismatch: with the realized keys held fixed, a large part of the full-context attention structure remains accessible through the query. The diagnostic uses 8 NIAH requests and five layers (1, 8, 16, 24, 31) of Llama-3.1-8B-Instruct at 8K context under A3–10, the setting studied in the main text.

C.3 Cache State versus Query-visible Geometry

The NIAH CacheBlend diagnostic compares two independently realized cache states for the same 16 queries. The 30% and 50% settings select different token sets (Jaccard similarity 0.604); their layer-2 key and value relative changes are 3.08% and 5.69%, respectively. Nevertheless, the relative change in the query-visible logits is only 1.40%. The additional tokens introduced by the 50%

Table 4: Llama-3.1-8B-Instruct paired budget matrix, cache-only → QueryLens. The strongest configuration (10% recomputation, A3) raises three-task macro-F1 from 26.5 to 69.1 and Biography from 6.5 to 98.8.

Cache Method	Budget	Biography	NIAH	HotpotQA	Macro	Δ macro
CacheBlend	10%	64.8→97.9	52.9→68.0	40.5→42.3	52.7→69.4	+16.7
	30%	90.6→98.8	60.9→64.9	41.8→41.8	64.4→68.5	+4.1
	50%	95.4→97.0	60.6→71.1	42.7→42.7	66.2→70.3	+4.1
	70%	98.4→96.3	77.2→78.8	43.1→43.8	72.9→73.0	+0.1
	90%	96.5→96.8	81.9→82.7	42.1→48.4	73.5→76.0	+2.5
A3	10%	6.5→98.8	38.5→67.6	34.5→40.8	26.5→69.1	+42.6
	30%	49.8→98.8	56.0→66.2	41.3→42.8	49.0→69.3	+20.3
	50%	76.5→98.8	55.7→60.6	45.0→48.7	59.1→69.4	+10.3
	70%	96.6→98.8	80.9→78.0	40.9→42.6	72.8→73.1	+0.3
	90%	98.2→98.2	82.9→73.9	27.3→43.0	69.5→71.7	+2.2

setting receive only 1.42% of the attention mass under the 30% query. The average attention entropy changes from 1.500 to 1.493 and Top-32 mass from 0.930 to 0.931.

For offline analysis we may compare two cache states using

$$D_{\mathrm{QV}}(a,b) = \frac{\|q(K_a - K_b)^{\mathsf{T}}\|_2}{\|qK_a^{\mathsf{T}}\|_2 + \varepsilon}, \tag{11}$$

but a deployment-time gate normally has only one cache. Online descriptors should therefore be functions of the current $(q, \widetilde{K})$, such as attention entropy, effective support, Top-k mass, and logit margin. The diagnostic supports the chain

$$\text{nominal budget} \rightarrow \text{realized cache} \rightarrow \text{query-visible geometry} \rightarrow \text{recoverability},$$

not a claim that one scalar descriptor universally predicts recoverability.

D Complete Paired Results

This appendix provides the complete paired matrices behind the main cross-model figure. Each table is a result ledger: it preserves cache method, budget, sample, and timing variations without introducing additional mechanism claims.

Notes for Figure 4. Every point pairs a cache-only query with its QueryLens counterpart under the same samples, cache state, seed, decoding policy, and timing boundary. KVPacket is plotted without a recomputation axis because it has no budget knob.

D.1 Llama-3.1-8B-Instruct: cache-only and QueryLens

Table 4 reports the paired Llama-3.1-8B-Instruct matrix for CacheBlend and A3. The values are F1 in percent; each cell is cache-only → QueryLens, and Δ is the paired difference in percentage points. EPIC is reported separately because its budget axis counts recomputed tokens rather than a recomputation ratio.

The table shows how query-correction effects vary across cache methods and recomputation settings. As the recomputation extent changes, each cache method produces a different realized cache, and the paired query-side effect changes accordingly; the full matrix retains these setting-level fluctuations explicitly.

D.2 Qwen3-4B-Instruct

Table 5 reports the paired Qwen3-4B-Instruct matrix for CacheBlend and A3.

Table 5: Qwen3-4B-Instruct paired budget matrix, cache-only → QueryLens. The strongest configuration (10% recomputation, A3) raises three-task macro-F1 from 18.39 to 54.85 and Biography from 9.64 to 83.80.

Cache Method	Budget	Biography	NIAH	HotpotQA	Macro	Δ macro
CacheBlend	10%	23.03→84.79	42.95→49.18	12.46→20.25	26.14→51.41	+25.26
	30%	38.33→85.25	51.71→56.40	16.46→21.78	35.50→54.47	+18.97
	50%	49.36→91.42	54.05→62.48	24.20→34.84	42.54→62.91	+20.37
	70%	75.50→95.37	61.28→67.70	22.22→21.01	53.00→61.36	+8.36
	90%	93.96→97.90	72.35→75.88	22.36→25.19	62.89→66.33	+3.44
A3	10%	9.64→83.80	28.85→63.91	16.67→16.84	18.39→54.85	+36.46
	30%	21.42→85.38	38.15→53.05	19.86→23.13	26.48→53.85	+27.37
	50%	30.03→87.44	43.99→50.81	25.45→30.58	33.16→56.28	+23.12
	70%	60.51→85.20	45.43→70.02	27.27→31.00	44.40→62.07	+17.67
	90%	85.27→92.38	54.74→45.63	24.97→21.40	55.00→53.14	−1.86

Table 6: Query-only QueryLens on independently compiled, no-recompute caches (score, %).

Model	Task	No-recompute	+QueryLens	Δ
Qwen3-4B-Instruct	NIAH	19.22	44.24	+25.02
Llama-3.1-8B-Instruct	NIAH	52.63	55.61	+2.98
Qwen3-4B-Instruct	Biography	18.12	83.11	+64.99
Llama-3.1-8B-Instruct	Biography	6.33	98.38	+92.05

The Qwen3-4B-Instruct matrix likewise reports query-correction effects across cache methods and budgets, preserving setting-level variation rather than assigning each paired fluctuation to a single scalar budget effect.

D.3 KVPacket combination

The most traceable Llama-3.1-8B-Instruct KVPacket combination uses Biography, NIAH, and HotpotQA scores of 96.13, 80.21, and 41.90 for cache-only, and 98.80, 81.70, and 43.30 with QueryLens. The corresponding macro-F1 values are 72.74 and 74.60, so the query composes with a recomputation-free cache method.

E Ablations and Controls

These controlled experiments isolate query-side effects from cache construction, parameter capacity, and correction magnitude. Each subsection states the held-fixed conditions before reporting the corresponding table.

E.1 Query-only causal isolation

Table 6 evaluates QueryLens-H on independently compiled, no-recompute caches. It uses the same cache for the cache-only and corrected queries, so the difference cannot be attributed to token recomputation or a change in cache construction. The complete results are reported here because the main text focuses its limited space on controlled component comparisons.

E.2 Clean-context Q-LoRA control

Table 7 gives the two NIAH controls discussed in the main text. Its cache-only column reuses the corresponding baselines from Table 1; the clean-context Q-LoRA scores come from the separately run matched control.

Table 7: Matched query-projection control (NIAH, score %). *cache-only* disables query correction and uses the native query on the reused cache. Q-LoRA and QueryLens-H use the same rank-4 per-head parameterization, initialization, training queries, optimizer, and full-context teacher. Q-LoRA trains on a clean joint-context KV student and therefore does not expose reused-cache mismatch.

Setting	*cache-only*	Clean-context Q-LoRA	QueryLens-H
Qwen / A3–10	28.85	32.33	**56.53**
Llama / CB–10	52.92	52.40	**66.43**

E.3 Rank, layer, and loss controls

The rank and scale controls in the main text fix the cache method, query split, teacher, decoding protocol, and cache-only baseline while changing the stated query-side component. Their matched comparisons separate capacity and correction-magnitude effects from changes in cache realization or timing.

F Efficiency and Deployment Accounting

This appendix closes with deployment details that are orthogonal to quality: amortization across queries and cache methods, online arithmetic, static folding, and the timing boundary used by the TTFT measurements.

F.1 QueryLens deployment and composition

Amortization across queries and cache methods. QueryLens learns a shared mapping rather than a per-request solution. We split data at the query level and keep every budget variant of a query in the same split; the resulting checkpoint is applied directly to held-out queries without further optimization. At deployment, the same QueryLens-H or QueryLens-T checkpoint can act alone on independently reused KV or compose after any compatible cache method. The cache method determines the realized memory, whereas QueryLens determines how the query accesses it; neither variant is retrained for the query, cache method, or budget.

Composition: compatibility with cache methods. For QueryLens-H, the separation is structural:

$$\text{cache construction: } x \mapsto (\widetilde{K}, \widetilde{V}) = \mathcal{M}(x), \qquad \text{query correction: } q \mapsto \widehat{q} = \Phi(q; \phi). \tag{12}$$

The cache method determines realized memory, while QueryLens-H changes only the query mapping. Given materialized, head-separated KV and an intact query path, it does not depend on the cache method's internal policy or change positions, tokens, values, or cache layout. Compatible cache methods therefore compose directly; altered query interfaces require recalibration.

Deployed cost: the static profile. With a fixed profile the correction leaves the runtime path. A frozen scalar gate turns the residual into $T_{l,h} = I + \gamma_{l,h} D_{l,h} U_{l,h}$ on the query, which is merged into the native projection before deployment (Eq. 8). The query path then has the cost of the unadapted model; an adaptive hook instead costs one rank-r pair of matrix products per adapted head.

F.2 Arithmetic and storage

For L_s selected layers, H heads, head dimension d_h, and rank r, QueryLens stores approximately

$$2L_s H r d_h + L_s H$$

parameters and performs $O(L_s H d_h r)$ arithmetic per corrected query token. It adds no sequence positions, no virtual tokens, and no second KV cache. The cache-side recomputation cost remains unchanged. QueryLens arithmetic is accounted for separately from the cache-construction cost.

F.3 STATIC MERGE EQUIVALENCE

For a fixed gate, let

$$T_l = \mathrm{blkdiag}_h \left(I + \gamma_{l,h} D_{l,h} U_{l,h} \right).$$

Then replacing $W_{Q,l}$ by $T_l^{\mathsf{T}} W_{Q,l}$ (and transforming a bias when present) is algebraically equivalent to the online residual up to floating point rounding. The static-folding measurements use a fixed premerged profile; merge and restore costs are outside the timed request.

F.4 STATIC FOLDING TTFT BY MODEL AND CONTEXT LENGTH

Table 8 gives the per-setting median TTFT comparisons summarized in Section 5.5.

Table 8: Median TTFT (ms) over 90 requests: cache-only / folded QueryLens (relative change), with 10% recomputation.

	Qwen3-4B		Llama-3.1-8B	
Context	A3–10	CB–10	A3–10	CB–10
4K	123.8/123.4 (−0.27%)	131.4/131.0 (−0.27%)	128.0/127.8 (−0.21%)	139.1/139.4 (0.21%)
8K	203.1/202.6 (−0.26%)	225.2/224.8 (−0.16%)	218.5/218.6 (0.03%)	248.0/247.4 (−0.24%)
16K	498.5/498.4 (−0.02%)	568.7/568.5 (−0.05%)	505.4/505.9 (0.10%)	593.4/593.3 (−0.02%)
32K	1498.8/1501.7 (0.19%)	1737.4/1740.1 (0.16%)	1461.1/1462.2 (0.07%)	1731.5/1733.9 (0.14%)