KVSHAREARENA: KV-CACHE REUSE ACROSS CONTEXTS AND MODEL CHECKPOINTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reusing key–value (KV) caches speeds up LLM inference by avoiding repeated computation on shared text. The standard method, prefix caching, reuses a KV cache only when the LLM is the same and all text before the reused content is identical. However, real workloads often break both conditions. The preceding text changes when a RAG system places different documents before the same document, or when agents with different system prompts read the same file or tool output. The LLM changes when multi-agent workflows use specialized LLMs on shared material, or when an updated model reads documents cached by its previous version. Because KV caches depend on both the preceding text and the model weights, direct reuse in these settings can reduce answer quality. Many methods repair the reused cache or compress it, each with its own balance between quality and cost, so choosing among them requires a comparison under the same conditions. However, each method paper uses its own tasks, models, and cost measures, so the reported results cannot be compared directly. Existing benchmarks do not provide such a comparison either, because they mainly test long-context processing or reuse of an unchanged prefix. To fill this gap, we introduce **KVShareArena**, a benchmark and open evaluation framework that compares these methods under the same conditions. KVShareArena has (1) reuse tests on 2,150 questions from three QA datasets, in which the preceding text, the LLM that wrote the cache, or both change while the answering LLM and the input text stay fixed; (2) five dense and mixture-of-experts LLMs from 4B to 30B parameters and six LLM pairs in which one version of an LLM reads caches written by another, for 33 model–dataset settings in total; (3) 11 repair and compression methods from six method classes; (4) four evaluation perspectives: answer quality, prefill computation, KV-cache memory, and latency; and (5) a common interface for adding new methods and an interactive leaderboard that lets users set their own balance between quality and speed. Experiments on KVShareArena yield two findings. First, both the quality loss from reuse and which repairs help depend on the LLM, even between two 8B models. Second, most repairs keep their quality when another version of the LLM wrote the cache, but a trained repair adapter loses quality in 12 of the 18 pair–dataset tests.

1 INTRODUCTION

LLM applications repeatedly process shared text, including instructions, documents, and conversation histories. In retrieval-augmented generation (RAG), for example, many queries retrieve the same documents. Before answering, the model encodes the text into key–value (KV) caches during *prefill*. Repeating prefill on the same text adds computation and delays the first output token. Reusing KV caches computed earlier avoids this repeated work (Hu et al., 2025; Yao et al., 2025).

Figure 2, adapted from Yao et al. (2025), shows four ways to process a RAG prompt. The standard method, prefix caching, avoids recomputing an unchanged prefix (Figure 2b). It is used in vLLM and SGLang (Zheng et al., 2024) and offered by commercial LLM APIs. Cached prefixes include system prompts, tool descriptions, and documents. However, reuse needs an exact match from the start of the prompt through the cached text (Figure 1a).

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.

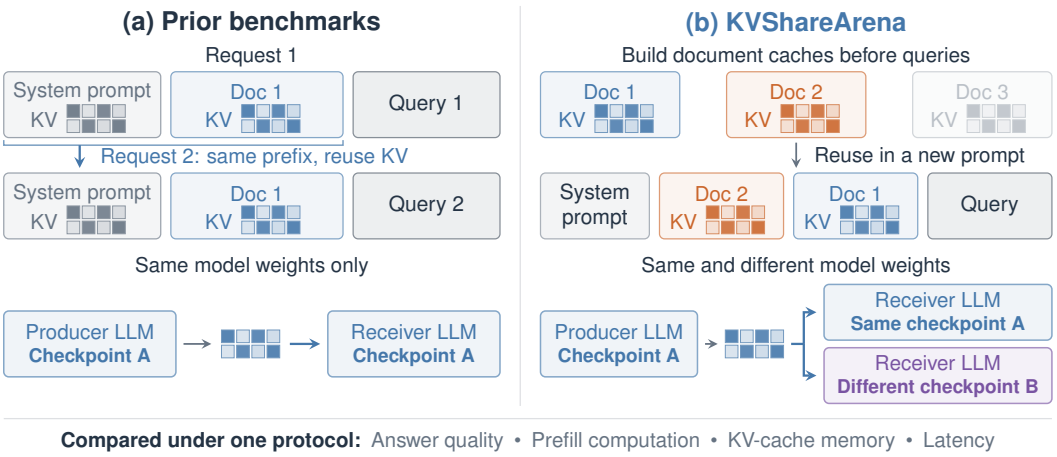


Figure 1: **KVShareArena evaluates reuse after context or model weights change.** **(a)** SCBench (Li et al., 2025) tests reuse of an unchanged prefix with the same model weights. **(b)** KVShareArena combines separately built document caches in new prompts and reads them with the same checkpoint A or a different checkpoint B of the same architecture and tokenizer. Small grids represent KV caches.

Cross-context reuse. Many LLM applications build each prompt from reusable pieces of text, such as documents, few-shot examples, and tool descriptions (Gim et al., 2024; Hu et al., 2025; Ordonez & Parnell, 2026). We call each such piece a *block*. The same block often recurs with different text before it (Hu et al., 2025). In RAG, the same document is retrieved together with different documents for different queries (Yao et al., 2025; Agarwal et al., 2025); in multi-agent systems, the same file or tool output follows a different system prompt in each agent (Ye et al., 2025; Ordonez & Parnell, 2026). Because it needs an exact match, prefix caching cannot reuse such a block once its preceding text changes. In practice, exact prefix caching could serve only 8% of requests in one production RAG service, and 75% of the blocks retrieved for each query had to be processed again (Agarwal et al., 2025). To reuse such blocks, recent methods encode each block separately and reuse its cache after different preceding text (Hu et al., 2025; Liu et al., 2026); we call this *cross-context reuse* (Figure 1b). A separately encoded block, however, has not seen its new preceding text. In Figure 2c, Doc 2's KV was computed without Doc 1, so direct reuse, which uses the cache as is, can lower answer quality.

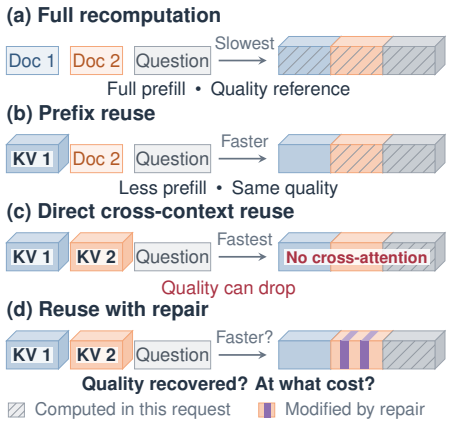


Figure 2: **Reuse saves prefill but can lose quality.** Repair tries to recover it at extra cost.

Cross-checkpoint reuse. Shared text also passes between different LLMs. For example, multi-agent workflows use models fine-tuned for different tasks. In DroidSpeak, a coding agent and a testing agent run different fine-tuned LLMs, and the testing agent must prefill the task instructions and code that the coding agent has already processed (Liu et al., 2025). Likewise, an updated model prefills again the documents its previous version has already cached. Prefix caching does not help here: it cannot reuse a cache written by another LLM, even for the same text. We call the LLM that writes a cache the *producer* and the LLM that reads it the *receiver*. We call reuse between a producer and a receiver with different weights (different checkpoints) *cross-checkpoint reuse* (Figure 1b). Different weights, however, produce different KV values for the same text, even with the same architecture and tokenizer. Direct reuse can therefore lower answer quality (Liu et al., 2025).

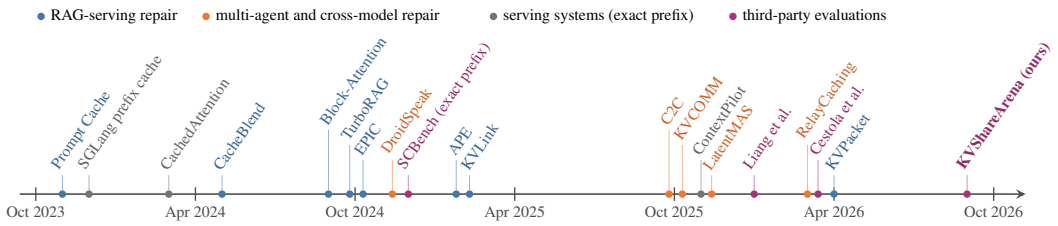


Figure 3: Repair methods for reused KV caches have appeared at a steady pace since late 2023, in RAG serving and in multi-agent systems, while third-party evaluation lags behind: one benchmark that reuses caches only under an exact prefix and two studies of cross-context reuse in 2026. Dates are arXiv first versions.

Table 1: **Of the evaluations listed, only KVShareArena covers reuse across both contexts and checkpoints, all four evaluation perspectives, and an open leaderboard.** SCBench compares compute and memory complexity; Liang et al. also report selection consistency and reuse rate.

	Reuse across		Evaluation perspectives				Open
Evaluation	Contexts	Checkpoints	Quality	Compute	Memory	Latency	leaderboard
SCBench	×	×	✓	✓	✓	×	×
Cestola et al. (2026)	✓	×	✓	×	×	×	×
Liang et al. (2026)	✓	×	✓	×	×	×	×
KVShareArena (ours)	✓	✓	✓	✓	✓	✓	✓

In both settings, the reused text is unchanged. Yet **the same text does not guarantee reusable KV.** When the preceding text changes, a reused cache misses that text and also sits at new positions in the prompt. The simplest fix, *position alignment*, re-encodes the cache's positions to match its new place in the prompt, without recomputing any token. It fixes the positions, but it does not add the missing context or adapt the cache to new weights. Such fixes, which we call *repair*, take extra computation when the cache is reused, or training the model or an adapter in advance. Because repair adds cost, reuse is useful only if it preserves enough answer quality and still costs less than recomputation (Figure 2d).

A missing benchmark setting. Several lines of work repair reused caches (Figure 3). RAG serving systems recompute a small share of tokens (Yao et al., 2025; Hu et al., 2025) or recalibrate attention across cached documents (Yang et al., 2025b). Other methods train the model or a small adapter to read separately cached blocks (Ma et al., 2025; Chen et al., 2026), and multi-agent systems correct caches with offsets stored from earlier requests (Ye et al., 2025). However, each study uses its own tasks, models, quality baselines, and cost measures, so a gain reported in one study may reflect a better method, an easier setting, or a different way of measuring. Existing benchmarks and studies do not compare these methods under the same conditions (Table 1). SCBench tests only reuse of an unchanged prefix with the same weights (Li et al., 2025). Two studies compare cross-context methods, but only on quality and not across checkpoints (Cestola et al., 2026; Liang et al., 2026). DroidSpeak builds a system for cross-checkpoint reuse rather than comparing methods (Liu et al., 2025). The open question is: **Which methods preserve answer quality at a lower cost than recomputation after context or checkpoint changes?**

Answering this question needs three design choices. First, the preceding text and the producer weights both change the KV, so they must be varied separately. Second, quality must be measured against full recomputation over the same text, so that a loss reflects reuse, not missing information. Third, each cost must be measured on its own, because one saving does not imply another: a method can avoid prefill computation yet keep the full cache.

We introduce **KVShareArena**, a benchmark and open evaluation framework for cross-context and cross-checkpoint reuse of document KV caches in RAG, built on these three choices. We measure quality by performance-gap recovered (PGR): 0 matches answering without the documents, and 1 matches full recomputation over the same text. Experiments on five LLMs from 4B to 30B yield

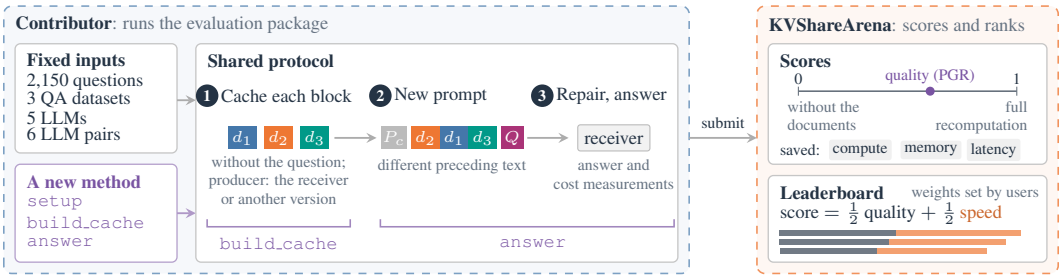


Figure 4: **KVShareArena needs three calls from a new method and supplies the inputs, protocol, and scoring.** Purple marks the method's own code.

two findings. First, the quality loss from reuse and the repairs that help depend on the model: position alignment alone reaches a multi-hop PGR of .41 on Llama-3.1-8B but .03 on Qwen3-8B, and CacheBlend, which recomputes a small share of tokens, significantly improves multi-hop QA on Qwen3-8B but not on Llama-3.1-8B (§3.5). Second, most repairs change by less than .02 F1 on average when another version of the LLM wrote the cache, while KVPacket's trained adapter loses up to .146 F1. Retraining the adapter on the new producer's caches recovers this loss, at about 20 GPU-hours per producer with Qwen3-8B as the receiver (§3.6). The experiments also show that compression, which drops or quantizes cached tokens, saves 25–72% of KV memory, but its best multi-hop PGR is .27, against .44–.52 for the leading repairs (§3.3).

Our contributions are:

- **A controlled protocol.** KVShareArena holds the receiver and the input text fixed and varies the preceding text and the producer weights separately, so each source of quality loss is measured on its own (§2.2). Every method uses the same fixed questions, texts, prompts, and scoring rules.
- **Four evaluation perspectives.** KVShareArena measures quality as PGR, and prefill computation, KV-cache memory, and latency as recomputed tokens, cache bytes, and time to first token (§2.3). The leaderboard ranks all methods by quality and speed (TTFT saved) and reports memory separately.
- **An open framework.** A new method implements three calls and is evaluated under the same protocol. Researchers can submit its per-sample outputs for review and inclusion in the public leaderboard (Figure 4; Appendix L).

2 KVSHAREARENA

Figure 4 shows how KVShareArena evaluates a method. This section follows the figure: the data and models (§2.1), how caches are built and reused (§2.2), and how answers and costs are scored (§2.3).

2.1 DATA AND MODELS

The main evaluation caches document blocks separately and assembles them for each question. It uses three QA datasets in the LongBench v1 format (Bai et al., 2024): Qasper, MultiFieldQA, and HotpotQA, which span scientific-paper, single-document, and multi-hop QA. HotpotQA blocks are separate Wikipedia articles, as in multi-document retrieval. Qasper and MultiFieldQA blocks are 512-token windows of one long document, as when a RAG system splits a document into chunks. Every method reads the same documents the dataset provides for each question, so retrieval quality does not affect the comparison. We extend LongBench's Qasper and HotpotQA subsets with more questions from the original datasets, to 1,000 questions each; MultiFieldQA keeps its 150 (Appendix G). The documents of each question form 2 to 43 separately cached blocks. Combining several separately cached blocks is how RAG builds a prompt, and it is where reuse loses quality and repairs differ (Appendix D). Appendix C runs the same test on reports written by specialist agents (*Agent Reports*).

We evaluate five models. Each model is both the producer and the receiver of its own caches: Qwen3-8B, Qwen3-4B, Qwen3-14B, the mixture-of-experts Qwen3-30B-A3B-Instruct-2507, and Llama-3.1-8B-Instruct. Six pairs, each joining a receiver with another version of the same model, test a producer change (§3.6).

2.2 Context and producer changes

We vary two factors independently: whether a block is read after the same text it was encoded with, and whether the producer weights match the receiver weights. The resulting 2×2 design has four cells: the context is the same or different, and the weights are the same or different. In the same-context cells the producer prefills the whole prompt in order and the receiver computes only the question; with matching weights this is exact prefix caching, which loses nothing and serves as a control.

In the different-context cells, each block is encoded alone, from position zero, with no text before it. The caches of m blocks are then assembled into one receiver prompt $[P_c, \mathrm{kv}(d_1), \ldots, \mathrm{kv}(d_m), Q]$, where P_c is the receiver's instruction prefix and Q the question. Every block starts at position zero, so their positions overlap. For a weight change we keep the architecture, tokenizer, receiver, text, samples, and scoring fixed: matching tensor shapes are guaranteed, matching KV values are not. Direct reuse, without any repair, is tested in all four cells. The repair methods run in the two different-context cells: the main results use the cell with the same weights, and §3.6 the cell with a weight change.

2.3 Metrics

We score answers with token-level F1, LongBench's official metric for these subsets (Bai et al., 2024). Their reference answers are short, so F1 needs no judge model and scores every submission the same way. For each model and subset we fix two reference scores: the *floor*, where the receiver answers without the documents, and the *ceiling*, where it recomputes the complete prompt over the *same text*. Because the ceiling reads the same text as the method, a loss against it reflects reuse, not missing information. A method's quality is the share of the floor-to-ceiling gap it recovers,

$$\mathrm{PGR} \;=\; \frac{S_{\mathrm{method}} - S_{\mathrm{floor}}}{S_{\mathrm{ceiling}} - S_{\mathrm{floor}}}, \tag{1}$$

which is 0 at the floor, 1 at the ceiling, may exceed 1, and may be negative: a reused cache can be worse than no context. The floor and ceiling are computed once per model and subset, so all methods share the same denominator. On 8B, the context is worth .326/.361/.338 F1 on the three subsets; Table 5 gives every model.

Methods spend and save different resources, and there is no agreed way to convert one into another, so each cost is reported on its own. **Compute**: Compute saved $= 1 - R/D$, where R is the number of document tokens a method recomputes, counted once per layer, and D the number full recomputation computes, both summed over samples. **Memory**: Memory saved $= 1 - B_{\mathrm{method}}/B_{\mathrm{full}}$, measured on the actual tensors (scales, indices, and copies included) against the uncompressed BF16 cache. **Latency**: TTFT saved $= 1 - T_{\mathrm{method}}/T_{\mathrm{full}}$, where T is the time to first token (TTFT), measured per request with the cache already available, with the method and full recomputation on the same backend. Building the caches is a one-time cost outside this measure; how soon it pays off depends on how often a workload reads the cache (Appendix F.1).

Score: The leaderboard score combines quality and speed and ranks all methods in one list. As in RouterArena (Lu et al., 2025), the score is a weighted sum, $S = \frac{1}{2}Q + \frac{1}{2}V$. Q is the mean PGR over the five models and three subsets, and V is the TTFT saved, averaged over the same models and subsets, each relative to the model's own full recomputation. Memory saved is compared separately (Table 4). Position alignment, the reference, is not scored. Users can change the weight on the leaderboard, and Appendix N lists three presets. A new method enters through the three calls of Figure 4; Appendix L describes the submission steps.

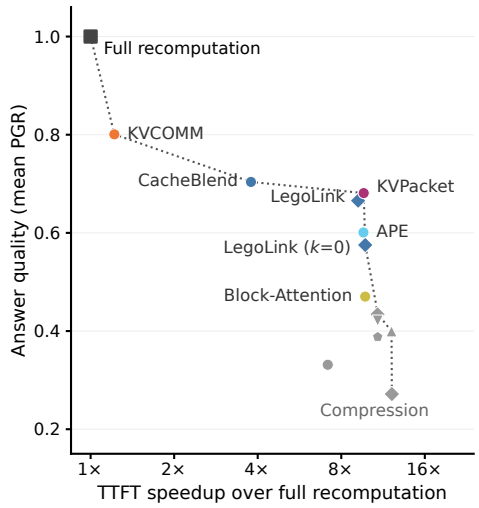


Figure 5: With the caches already available, most methods cut time to first token about tenfold relative to full recomputation and differ mainly in quality. CacheBlend gives up part of that speed for quality, and compression (gray) keeps less quality at a similar speed. Quality is the mean PGR over five models and three QA subsets; speed is the TTFT speedup over full recomputation, from the share of TTFT saved on the same models and subsets. Dotted line: methods that no other method beats on both quality and speed.

Table 2: The KVShareArena leaderboard: repair methods take the top six places, and compression keeps less quality at a similar speed. Score $= \frac{1}{2}$ quality $+\frac{1}{2}$ speed for every method. Twelve of the 14 entries have speed scores of .86–.92, so quality decides most of the order. Quality is as in Figure 5; speed is the share of TTFT saved, averaged over the same models and subsets. KV memory is compared separately (Table 4). LegoLink recomputes the first k tokens of each block after the first (k=2 unless marked). *Recomputes the questions whose documents it has not seen.

#	Method	Quality	Speed	Score
1	⑦ KVPacket	.681	.896	.789
2	① LegoLink	.666	.891	.778
3	② APE	.601	.896	.748
4	① LegoLink (k=0)	.575	.898	.736
5	① CacheBlend	.704	.736	.720
6	⑥ Block-Attention	.470	.897	.684
7	⑨ SnapKV (r=.25)	.435	.908	.671
8	⑨ Knorm (r=.25)	.434	.907	.671
9	⑨ TOVA (r=.25)	.421	.908	.664
10	⑨ SnapKV (r=.5)	.399	.918	.659
11	⑨ StreamingLLM (r=.25)	.388	.908	.648
12	⑨ Quant. 4-bit	.331	.860	.596
13	⑨ Knorm (r=.5)	.272	.918	.595
14	④ KVCOMM*	.801	.179	.490

3 EXPERIMENTS

After the methods and settings (§3.1), we rank the methods after a context change and break the ranking into quality and costs (§3.2–§3.4). We then look at the effect of the receiver model (§3.5) and of the producer model (§3.6).

3.1 METHODS AND SETTINGS

The main evaluation compares six method classes against position alignment (③, the reference), marked by the same glyphs in all tables: ① selective recomputation, ② attention calibration, ④ anchor-delta reuse (correcting a cache with offsets stored from earlier requests), ⑥ fine-tuned repair, ⑦ trained soft-token adapter (a few trained tokens wrapped around each cached block; the model itself is unchanged), and ⑨ compression under reuse. Table 12 (appendix) lists every method, including two further classes (⑤, ⑧). Each method uses the settings of its paper or official implementation. Where a method has a budget to choose, we run LegoLink at k=2 and k=0 recomputed tokens per block and the eviction methods at r=.25, the share of cached tokens removed, with SnapKV and Knorm also at r=.5. Every comparison is paired: both sides answer the same questions. We write $n/5$ for the number of models on which a method significantly beats position alignment (paired bootstrap, 95%). Appendix H lists the run settings.

3.2 OVERALL RANKING

Repair methods take the top six places on the leaderboard, and compression keeps less quality at a similar speed (Table 2). Most methods cut time to first token about tenfold, so quality decides most of the order (Figure 5). KVPacket and LegoLink lead. CacheBlend and KVCOMM keep more quality but give up speed for it, so they rank fifth and last. With a quality weight of 0.7 or 0.3 instead

Table 3: **Repair beats position alignment on single-document and multi-hop QA, and the leading repairs outperform compression.** Mean PGR over the five models, each against its own floor and ceiling. Superscripts count the models on which the method is significantly above position alignment (paired bootstrap, 95%); **bold** = above on at least three models. Bold entries come from five method classes.

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	.630	.550	.209
① CacheBlend (15% recompute)	**.868**$^{\uparrow 5}$	**.807**$^{\uparrow 5}$	**.436**$^{\uparrow 4}$
① LegoLink (k=2, ≈0.4% recompute)	**.802**$^{\uparrow 4}$	**.732**$^{\uparrow 4}$	**.463**$^{\uparrow 4}$
① LegoLink (k=0, 0% recompute)	.718$^{\uparrow 2}$	**.642**$^{\uparrow 3}$	**.366**$^{\uparrow 3}$
② APE	.605	.702$^{\uparrow 2}$	**.496**$^{\uparrow 5}$
④ KVCOMM	.546$^{\uparrow 1}$	**.857**$^{\uparrow 5}$	**.999**$^{\uparrow 5}$
⑥ Block-Attention	.556	.551	**.304**$^{\uparrow 4}$
⑦ KVPacket (0% recompute)	**.788**$^{\uparrow 5}$	**.737**$^{\uparrow 4}$	**.518**$^{\uparrow 5}$
⑨ Quant. 4-bit	.543	.483	−.032
⑨ SnapKV (r=.25)	.600	.494	.212$^{\uparrow 1}$
⑨ SnapKV (r=.5)	.552	.431	.215
⑨ Knorm (r=.25)	.560	.470	.271$^{\uparrow 1}$
⑨ Knorm (r=.5)	.387	.265	.163
⑨ TOVA (r=.25)	.586	.482	.197
⑨ StreamingLLM (r=.25)	.628	.504	.032

of 0.5 (Appendix N), KVPacket and LegoLink stay first and second, and repair methods keep the top five places. Per-dataset and per-model results are in Appendix B.

3.3 Answer quality

Repair improves answer quality on all three QA subsets, and the largest unrecovered gap remains on multi-hop QA.

The comparisons below average the five models, each scored against its own floor and ceiling (Table 3; Figure 6 in the appendix): **1)** Position alignment alone recovers .63, .55, and .21 of the score gap on scientific-paper, single-document, and multi-hop QA. It leaves room for repair on all three subsets. **2)** Selective recomputation ① improves all three subsets. CacheBlend beats position alignment on 5/5, 5/5, and 4/5 models, respectively; LegoLink does so on 4/5 for each. Attention calibration ② improves multi-hop QA on all five models. These repairs lift the multi-hop mean to .44–.50. **3)** KVPacket ⑦ improves all three subsets without recomputing any document token. It reaches mean PGR .79/.74/.52 and beats position alignment on 5/5, 4/5, and 5/5 models, respectively. **4)** Compression ⑨ keeps less quality than the leading repairs (Table 4); its best multi-hop mean is .27.

Table 4: **Compression saves 25–72% of KV memory but keeps less quality than every repair method.** Means over five models and three QA subsets; quality is PGR.

Method	Memory saved	Quality
⑨ Quant. 4-bit	.719	.331
⑨ SnapKV (r=.5)	.500	.399
⑨ Knorm (r=.5)	.500	.272
⑨ SnapKV (r=.25)	.250	.435
⑨ Knorm (r=.25)	.250	.434
⑨ TOVA (r=.25)	.250	.421
⑨ StreamingLLM (r=.25)	.250	.388

3.4 Costs: compute, memory, and latency

Compute and latency separate methods by how many tokens they recompute; memory separates compression from repair. Each saving is measured against the same model's full recomputation and then averaged over the five models.

Compute. Most methods recompute none of the cached document tokens (Figure 6, appendix). Their main difference is answer quality. Among methods that do recompute, CacheBlend, run at a 15% recomputation budget, spends about 17% of the document computation of full recomputation

Table 5: **Models differ in how much position alignment leaves to repair.** Worth = ceiling−floor in F1; left = the share of that gap still missing after position alignment.

Board	Sci.-paper QA		Single-doc QA		Multi-hop QA	
	worth	left	worth	left	worth	left
Qwen3-8B	.326	.432	.361	.597	.338	.970
Qwen3-4B	.322	.496	.357	.592	.353	.684
Llama-3.1-8B-Instruct	.319	.249	.474	.352	.289	.591
Qwen3-14B	.366	.349	.327	.417	.327	.806
Qwen3-30B-A3B	.304	.324	.364	.289	.316	.904

and LegoLink with k=2 spends under 0.5%. Standard prefix caching must recompute all document tokens in this setting and therefore saves no document computation.

Memory. Only compression saves KV bytes, at a cost to quality (Table 4). The repair methods keep the full cache. KVPacket's soft tokens add about 3%.

Latency. Methods that recompute more tokens save less TTFT (Table 2). Repairs that recompute no document tokens save .81–.94 of TTFT across models and datasets. CacheBlend saves .66–.78.

Figures 8 and 9 in the appendix show the dataset-level results on Qwen3-8B.

3.5 Effect of the receiver model

Which repairs help depends on the receiver model. No method leads on every model: on multi-hop QA, LegoLink and APE reach .50 and .49 PGR on 8B, KVPacket reaches .54 on 4B, .56 on 14B, and .61 on Llama, and CacheBlend reaches .57 on 30B (appendix). Models also differ in how well they read separately cached blocks: position alignment alone recovers .03 of the multi-hop gap on 8B, compared with .32 on 4B, .19 on 14B, and .41 on Llama (Table 5). On Llama, KVPacket, APE, and Block-Attention significantly improve the multi-hop score, while CacheBlend and LegoLink do not. Choosing a method therefore needs results on more than one model. Some gains still hold on every model: CacheBlend gains at least .20 PGR over position alignment on single-document QA, and KVPacket and APE gain at least .20 and .11, respectively, on multi-hop QA (Table 14, appendix).

3.6 Effect of the producer model

Most repairs keep their quality when another version of the model writes the cache, while KVPacket's trained adapter needs retraining. The producer is either a sibling post-trained from the same base, as when a service updates its post-trained model, or the receiver's own base model before post-training, the largest weight change within one family; Table 16 (appendix) lists the six pairs. These tests show how existing reuse methods respond to a new producer. DroidSpeak, a system designed for weight changes, has not released its code (Appendix E). Without repair (Table 17, appendix), every cross-context score is lower when a different producer wrote the cache, but the drop size depends on task and producer both.

Re-running every repair method with a same-family producer on all five models shows which repairs depend on the producer (Table 14, appendix). Most lose less than .02 F1 on average and at most .04 in any test (Appendix J); CacheBlend has no significant drop in any test. Two repairs change more. KVPacket, whose adapter is trained on the receiver's own caches, falls significantly below its same-producer score in 12 of the 18 tests (six producer pairs × three QA subsets; worst −.146 F1). LegoLink loses up to .151 F1 without recomputation (k=0), while with k=2 it stays within .04.

Of the two trained repairs, KVPacket changes with the producer and Block-Attention does not (less than .01 F1 on average). Does the adapter need a new design, or only retraining? We retrained the adapter with its published recipe and budget on caches from each new producer and tested it on the six tests with 8B as the receiver (two producers × three subsets; appendix). We compare it with the original adapter and with a cross-trained adapter, trained on the other producer. Retraining on the actual producer substantially recovers the lost quality, by up to .25 F1 where the original adapter had lost the most, and we detect no remaining difference from the same-model baseline in any test.

The cross-trained adapter recovers only part of that loss and stays below the baseline in every test, so the adapter must be trained on the producer it will read. This retraining closes the gap at about 20 GPU-hours per producer.

A cache written by a checkpoint from a different pretraining run lowers every method, by up to .32 F1 (Table 15, appendix); reuse across such checkpoints is still open.

4 Related Work

Four forms of KV reuse. We group work on KV reuse into four forms. **(i) Prefix caching** (Zheng et al., 2024) reuses KV only under an exact match. It is lossless and aimed at throughput. In our setting the preceding text always changes, so it recomputes every document (Appendix K). **(ii) Reuse after the preceding text or the model changes, with repair**, the setting this paper evaluates: selective recomputation (Yao et al., 2025; Hu et al., 2025; Liu et al., 2025), attention calibration (Yang et al., 2025b), and anchor-delta reuse (Ye et al., 2025). **(iii) Cache storage**: caches moved across the memory hierarchy (Gao et al., 2024), a separate problem from repair. **(iv) Input rewriting**: reordering contexts to create common prefixes (Jiang et al., 2026); it changes the input rather than repairing the cache.

Repair methods and their own evaluations. Repair methods are evaluated in their own papers, each on its own tasks and cost measures. The earliest form of (ii) is modular reuse with position re-encoding (Gim et al., 2024). RAG-serving methods report F1 changes and TTFT on chunked QA. Fine-tuned repair (Lu et al., 2024; Yang et al., 2025a; Ma et al., 2025) retrains the model; we include it and report its training cost. The newest method is a trained soft-token adapter (Chen et al., 2026). RelayCaching (Geng et al., 2026), which repairs the KV an agent produced while writing its output, appears in our supplementary Agent Reports experiment (Appendix C). KVCOMM (Ye et al., 2025) corrects caches with offsets stored from earlier requests. Latent communication (Zou et al., 2026) changes what agents exchange, hidden states instead of text, and does not repair a cache. Cross-architecture fusion (Fu et al., 2026) targets models of different architectures.

Benchmarks and evaluations. SCBench (Li et al., 2025) evaluates long-context methods through shared-prefix, multi-turn, and multi-request reuse. Each cache is written and read by the same model. KVShareArena instead benchmarks reuse after the preceding text or model weights change. It compares the retained answer quality and repair cost across methods.

Cross-context reuse has also received focused empirical study. Cestola et al. (2026) compare eleven methods for cross-context reuse on answer quality but do not measure serving cost. Liang et al. (2026) study reuse failures in multi-agent evaluation. DroidSpeak (Liu et al., 2025) develops a cross-checkpoint sharing system and evaluates its own method. These studies motivate the problem but do not offer a shared benchmark that compares quality and cost when the context or the checkpoint changes.

RouterArena (Lu et al., 2025) provides a shared evaluation protocol and an open leaderboard for LLM routers. KVShareArena adopts this open comparison approach for KV reuse. HELMET and MemoryBench (Yen et al., 2025; Ai et al., 2026) benchmark long-context models and memory in LLM systems; LongBench (Bai et al., 2024) supplies our QA tasks.

5 Conclusion

KVShareArena compares KV-cache reuse methods under the same conditions when the preceding text, the producer, or both change, and reports quality, compute, memory, and latency separately. The experiments yield two findings. First, both the quality loss from reuse and which repairs help depend on the receiver, even between two 8B models. Second, most repairs keep their quality when another version of the model wrote the cache, but a trained repair adapter can need retraining for each new producer. The experiments also show that compression saves memory but recovers less quality than the leading repairs. These results point to repairs that also shrink the cache and to repairs that adapt to new producers at lower cost. Reuse across models from different pretraining runs remains open. KVShareArena's common interface and leaderboard let new methods join the same comparison.

Reproducibility Statement

The anonymized supplementary material provides an installable evaluation package, the fixed question sets, reference scores for the five receiver models, an example driver for position alignment, and a static copy of the leaderboard with the results in this paper. The package re-scores per-sample outputs on CPU and computes F1 and PGR. Section 2 and Appendices E, G, and H give the models, method settings, datasets, and run settings; Appendix L describes the release and submission steps.

Ethics Statement

This work evaluates methods for reusing KV caches on public question-answering data (LongBench, Qasper, HotpotQA, and Wikipedia text) with publicly released model checkpoints. It involves no human subjects and collects no personal data. Sharing caches across requests, users, or models raises privacy and security questions, which we discuss in Appendix A. The released question sets keep the licenses of their sources.

AI Use Statement

AI assistance was used for engineering, writing, and auxiliary data analysis under author review. All experimental decisions, claims, and final text are the authors' responsibility.

References

Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. Cache-Craft: Managing Chunk-Caches for Efficient Retrieval-Augmented Generation, 2025. URL https://arxiv.org/abs/2502.15734.

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. MemoryBench: A Benchmark for Memory and Continual Learning in LLM Systems, 2026. URL https://arxiv.org/abs/2510.17281.

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, 2024. URL https://arxiv.org/abs/2308.14508.

Samuel Cestola, Tianxiang Xia, Zheng Weiyan, Zheng Pengfei, and Diego Didona. An experimental study of KV cache reuse strategies in chunk-level caching systems, 2026. URL https://arxiv.org/abs/2603.20218.

Chuangtao Chen, Grace Li Zhang, Xunzhao Yin, Cheng Zhuo, Bing Li, and Ulf Schlichtmann. KV Packet: Recomputation-Free Context-Independent KV Caching for LLMs, 2026. URL https://arxiv.org/abs/2604.13226.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A Simple and Effective L_2 Norm-Based Strategy for KV Cache Compression, 2024. URL https://arxiv.org/abs/2406.11430.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-Cache: Direct Semantic Communication Between Large Language Models, 2026. URL https://arxiv.org/abs/2510.03215.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention, 2024. URL https://arxiv.org/abs/2403.19708.

Yingsheng Geng, Yuchong Gao, Weihong Wu, Guyue Liu, and Jiang Liu. RelayCaching: Accelerating LLM Collaboration via Decoding KV Cache Reuse, 2026. URL https://arxiv.org/abs/2603.13289.

In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt Cache: Modular Attention Reuse for Low-Latency Inference, 2024. URL https://arxiv.org/abs/2311.04934.

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, 2025. URL https://arxiv.org/abs/2410.15332.

Yinsicheng Jiang, Yeqi Huang, Liang Cheng, Cheng Deng, Xuan Sun, and Luo Mai. ContextPilot: Fast Long-Context Inference via Context Reuse, 2026. URL https://arxiv.org/abs/2511.03475.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation, 2024. URL https://arxiv.org/abs/2404.12457.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. WILDS: A Benchmark of in-the-Wild Distribution Shifts, 2021. URL https://arxiv.org/abs/2012.07421.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV Cache-Centric Analysis of Long-Context Methods, 2025. URL https://arxiv.org/abs/2412.10319.

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, 2024. URL https://arxiv.org/abs/2404.14469.

Sichu Liang, Zhenglin Wang, Jiajia Chu, Pengfei Xia, Hui Zang, and Deyu Zhou. When KV Cache Reuse Fails in Multi-Agent Systems: Cross-Candidate Interaction is Crucial for LLM Judges, 2026. URL https://arxiv.org/abs/2601.08343.

Yifei Liu, Juntong Wu, Yang Liu, Junhao Hu, Minghao Li, Xiaoxu Chen, and Weihang Chen. HYPIC: Accelerating Hybrid-Attention LLM Serving with Position-Independent Caching, 2026. URL https://arxiv.org/abs/2607.01299.

Yuhan Liu, Yuyang Huang, Jiayi Yao, Shaoting Feng, Zhuohan Gu, Kuntai Du, Hanchen Li, Yihua Cheng, Junchen Jiang, Shan Lu, Madan Musuvathi, and Esha Choukse. DroidSpeak: KV Cache Sharing for Cross-LLM Communication and Multi-LLM Serving, 2025. URL https://arxiv.org/abs/2411.02820.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating Retrieval-Augmented Generation with Precomputed KV Caches for Chunked Text, 2024. URL https://arxiv.org/abs/2410.07590.

Yifan Lu, Rixin Liu, Jiayi Yuan, Xingqi Cui, Shenrun Zhang, Hongyi Liu, and Jiarong Xing. RouterArena: An Open Platform for Comprehensive Comparison of LLM Routers, 2025. URL https://arxiv.org/abs/2510.00202.

Dongyang Ma, Yan Wang, and Lan Tian. Block-Attention for Efficient Prefilling, 2025. URL https://arxiv.org/abs/2409.15355.

Nathan Ordonez and Thomas Parnell. MiniPIC: Flexible Position-Independent Caching in <100LOC, 2026. URL https://arxiv.org/abs/2606.13126.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are Multi-State RNNs, 2024. URL https://arxiv.org/abs/2401.06104.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient Streaming Language Models with Attention Sinks, 2024. URL https://arxiv.org/abs/2309.17453.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating Large Language Models via Efficient KV Cache Reuse, 2025a. URL https://arxiv.org/abs/2502.16002.

Xinyu Yang, Tianqi Chen, and Beidi Chen. APE: Faster and Longer Context-Augmented Generation via Adaptive Parallel Encoding, 2025b. URL https://arxiv.org/abs/2502.05431.

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, 2025. URL https://arxiv.org/abs/2405.16444.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online Cross-context KV-cache Communication for Efficient LLM-based Multi-agent Systems, 2025. URL https://arxiv.org/abs/2510.12872.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to Evaluate Long-Context Language Models Effectively and Thoroughly, 2025. URL https://arxiv.org/abs/2410.02694.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient Execution of Structured Language Model Programs, 2024. URL https://arxiv.org/abs/2312.07104.

Jiaru Zou, Ruizhong Qiu, Gaotang Li, Xiyuan Yang, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent Collaboration in Multi-Agent Systems, 2026. URL https://arxiv.org/abs/2511.20639.

Appendix Overview

A Limitations . . . 13
B Detailed Leaderboard Results . . . 13
C Supplementary Experiment: Agent Reports . . . 17
D When Position Alignment Is Enough . . . 19
E Method Matrix . . . 19
F Latency and Memory by Dataset . . . 19
G Dataset Cards . . . 21
H Run Settings . . . 22
I Notes on the Per-Board Results . . . 22
J Robustness: Does a Repair Stay Effective When the Setting Changes? . . . 23
K Relation to Prefix Caching . . . 23
L Open Platform and Submission Workflow . . . 24
M Cross-Checkpoint Supplement . . . 24
N Leaderboard Score Under Other Weights . . . 25

A Limitations

Sharing caches across requests, users, or models also raises security and privacy questions. A cache is computed from its text, so it may reveal that text to anyone who can read or probe it. A cache hit is faster than a miss, so response times alone can hint at what another user has sent. A cache written by one party and read by another can also be altered, and the reader cannot easily verify it. Deployments therefore need to decide which caches may be shared, with whom, and for how long. These questions are separate from the quality and cost trade-offs we measure, and we leave them to future work.

B Detailed Leaderboard Results

Table 3 gives the numbers behind Figure 6: each method's mean PGR over the five models and the number of models on which it is significantly above position alignment. One table per model follows; we call each model's results a *board*.

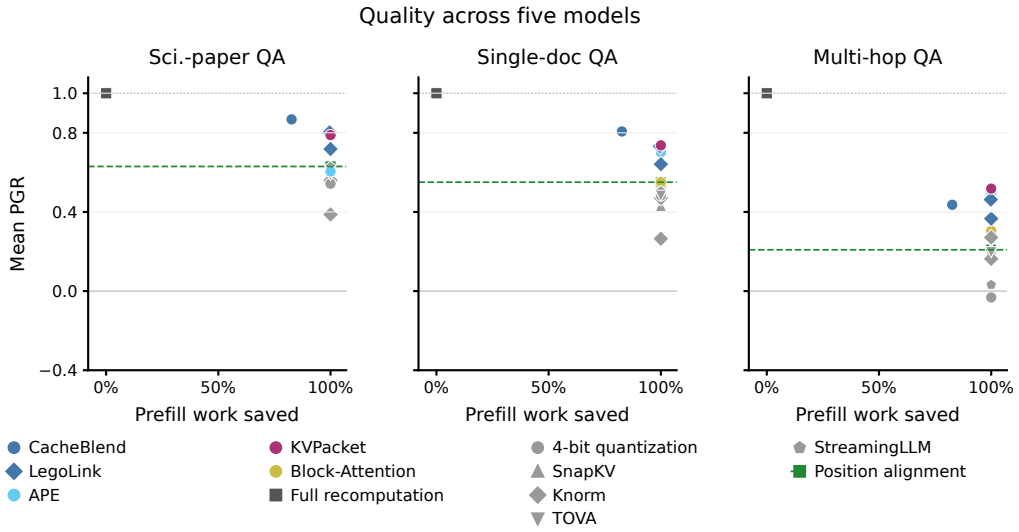


Figure 6: Dataset-level quality and recomputation cost. Each panel is one QA subset of the main evaluation. PGR is averaged over the five models, each against its own floor and ceiling. The horizontal axis shows Compute saved on the Qwen3-8B board. Higher and farther right is better. The dashed line marks position alignment. Configurations of the same method share a symbol. Table 3 lists scores and paired comparison counts.

Table 6: Qwen3-8B board (reference board). PGR (F1); $^{\uparrow}$ = significantly above position alignment (paired, 95%). KVCOMM keeps the offsets it stores from earlier questions and recomputes the questions whose documents it has not seen (Table 2).

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	.568	.403	.030
① CacheBlend (15% recompute)	.857$^{\uparrow}$	.748$^{\uparrow}$	.440$^{\uparrow}$
① LegoLink (k=2, ≈0.4% recompute)	.853$^{\uparrow}$	.674$^{\uparrow}$	.496$^{\uparrow}$
① LegoLink (k=0, 0% recompute)	.569	.489	.397$^{\uparrow}$
② APE	.549	.633$^{\uparrow}$	.492$^{\uparrow}$
④ KVCOMM	.539	.847$^{\uparrow}$	1.001$^{\uparrow}$
⑥ Block-Attention	.489	.471	.124$^{\uparrow}$
⑦ KVPacket (0% recompute)	.750$^{\uparrow}$	.645$^{\uparrow}$	.469$^{\uparrow}$
⑨ Quant. 4-bit	.388	.382	−.115
⑨ SnapKV (r=.25)	.541	.352	.026
⑨ SnapKV (r=.5)	.499	.301	.041
⑨ Knorm (r=.25)	.512	.425	.071
⑨ Knorm (r=.5)	.338	.215	−.075
⑨ TOVA (r=.25)	.533	.359	.016
⑨ StreamingLLM (r=.25)	.599	.398	−.054

Table 7: Qwen3-4B board. PGR (F1); $^{\uparrow}$ = significantly above position alignment (paired, 95%).

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	.504	.408	.316
① CacheBlend (15% recompute)	.794$^{\uparrow}$	.685$^{\uparrow}$	.453$^{\uparrow}$
① LegoLink (k=2, ≈0.4% recompute)	.802$^{\uparrow}$	.685$^{\uparrow}$	.505$^{\uparrow}$
① LegoLink (k=0, 0% recompute)	.763$^{\uparrow}$	.554$^{\uparrow}$	.288
② APE	.413	.525	.545$^{\uparrow}$
④ KVCOMM	.576$^{\uparrow}$	.851$^{\uparrow}$	.998$^{\uparrow}$
⑥ Block-Attention	.423	.479	.397$^{\uparrow}$
⑦ KVPacket (0% recompute)	.798$^{\uparrow}$	.658$^{\uparrow}$	.543$^{\uparrow}$
⑨ Quant. 4-bit	.307	.268	−.390
⑨ SnapKV (r=.25)	.470	.303	.366$^{\uparrow}$
⑨ SnapKV (r=.5)	.450	.286	.362
⑨ Knorm (r=.25)	.468	.272	.356
⑨ Knorm (r=.5)	.281	.125	.174
⑨ TOVA (r=.25)	.453	.331	.322
⑨ StreamingLLM (r=.25)	.484	.401	.198

Table 8: Llama-3.1-8B-Instruct board. PGR (F1); $^{\uparrow}$ = significantly above position alignment (paired, 95%).

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	.751	.648	.409
① CacheBlend (15% recompute)	.926$^{\uparrow}$	.850$^{\uparrow}$	.332
① LegoLink (k=2, ≈0.4% recompute)	.753	.639	.394
① LegoLink (k=0, 0% recompute)	.812	.775$^{\uparrow}$	.418
② APE	.820	.870$^{\uparrow}$	.519$^{\uparrow}$
④ KVCOMM	.461	.840$^{\uparrow}$	.997$^{\uparrow}$
⑥ Block-Attention	.644	.560	.552$^{\uparrow}$
⑦ KVPacket (0% recompute)	.863$^{\uparrow}$	.825$^{\uparrow}$	.611$^{\uparrow}$
⑨ Quant. 4-bit	.744	.616	.380
⑨ SnapKV (r=.25)	.744	.601	.395
⑨ SnapKV (r=.5)	.667	.538	.401
⑨ Knorm (r=.25)	.615	.480	.452
⑨ Knorm (r=.5)	.456	.326	.463
⑨ TOVA (r=.25)	.701	.591	.419
⑨ StreamingLLM (r=.25)	.766	.632	.102

Table 9: Qwen3-14B board. PGR (F1); $^{\uparrow}$ = significantly above position alignment (paired, 95%).

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	.651	.583	.194
① CacheBlend (15% recompute)	.858$^{\uparrow}$	.811$^{\uparrow}$	.386$^{\uparrow}$
① LegoLink (k=2, ≈0.4% recompute)	.853$^{\uparrow}$	.818$^{\uparrow}$	.434$^{\uparrow}$
① LegoLink (k=0, 0% recompute)	.682	.569	.274$^{\uparrow}$
② APE	.607	.742	.504$^{\uparrow}$
④ KVCOMM	.609	.862$^{\uparrow}$	.999$^{\uparrow}$
⑥ Block-Attention	.617	.542	.295$^{\uparrow}$
⑦ KVPacket (0% recompute)	.746$^{\uparrow}$	.736$^{\uparrow}$	.555$^{\uparrow}$
⑨ Quant. 4-bit	.620	.556	.125
⑨ SnapKV (r=.25)	.619	.518	.185
⑨ SnapKV (r=.5)	.554	.441	.153
⑨ Knorm (r=.25)	.683	.611	.320$^{\uparrow}$
⑨ Knorm (r=.5)	.431	.290	.249
⑨ TOVA (r=.25)	.599	.509	.152
⑨ StreamingLLM (r=.25)	.621	.469	−.007

Table 10: Qwen3-30B-A3B-Instruct-2507 board (mixture of experts). PGR (F1); $^{\uparrow}$ = significantly above position alignment (paired, 95%).

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	.676	.711	.096
① CacheBlend (15% recompute)	.905$^{\uparrow}$	.941$^{\uparrow}$	.572$^{\uparrow}$
① LegoLink (k=2, ≈0.4% recompute)	.751$^{\uparrow}$	.843$^{\uparrow}$	.485$^{\uparrow}$
① LegoLink (k=0, 0% recompute)	.765$^{\uparrow}$	.821$^{\uparrow}$	.454$^{\uparrow}$
② APE	.634	.738	.421$^{\uparrow}$
④ KVCOMM	.546	.883$^{\uparrow}$	1.002$^{\uparrow}$
⑥ Block-Attention	.606	.702	.153
⑦ KVPacket (0% recompute)	.787$^{\uparrow}$	.820	.412$^{\uparrow}$
⑨ Quant. 4-bit	.656	.595	−.160
⑨ SnapKV (r=.25)	.627	.693	.086
⑨ SnapKV (r=.5)	.589	.590	.119
⑨ Knorm (r=.25)	.524	.563	.156
⑨ Knorm (r=.5)	.430	.369	.004
⑨ TOVA (r=.25)	.643	.618	.074
⑨ StreamingLLM (r=.25)	.669	.622	−.080

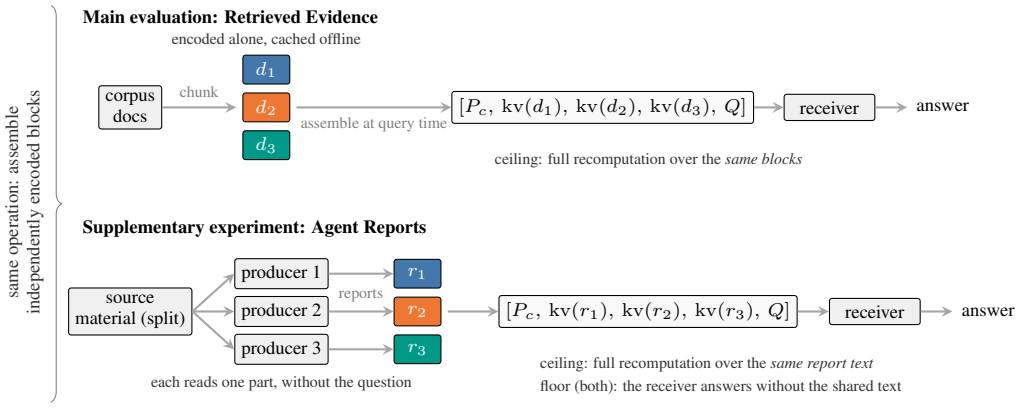


Figure 7: Document assembly in the main evaluation and report assembly in the supplement. The first uses separately cached document blocks. The second uses fixed reports written without the question. The diagram shows how most methods build the caches; RelayCaching instead reuses the KV produced while each report was written. Each experiment has its own ceiling (§2.3), so scores are not averaged across the two.

Table 11: Agent Reports: context value and room for repair, under the same definitions as Table 5. Worth = ceiling minus floor; left = the share of that gap still missing after position alignment. ‡ worth less than .25 F1.

Board	worth	left
Qwen3-8B	.252	.757
Qwen3-4B	.316	.490
Llama-3.1-8B-Instruct	.152‡	1.030
Qwen3-14B	.303	.425
Qwen3-30B-A3B	.237‡	.796

C Supplementary Experiment: Agent Reports

This experiment asks whether repair methods also help when the shared text consists of specialist reports rather than raw document chunks. It uses the same HotpotQA questions as the main evaluation, which we call Retrieved Evidence here. Results are reported separately and do not enter the main Retrieved Evidence means. On each board, the producers and the receiver (the agent that reads the reports and answers) use the same model.

C.1 Report construction and evaluation

Specialist agents each read a disjoint part of the source material and write a report without seeing the question. Each board's reports are written once and then fixed, so every method receives the same report text.

Most methods encode each report on its own and pass its cache to the receiver (Figure 7). RelayCaching instead reuses the KV produced while each report was written, obtained by replaying the fixed report text, and repairs it. The two procedures can give different KV for the same report. The benchmark records the cost of cache construction separately from per-request reuse.

The no-context floor is unchanged. The ceiling is full recomputation over the *same report text*, not the original documents. Reports can leave out information. A ceiling over the original documents would count that loss too, so the ceiling uses the report text. PGR is therefore not averaged across Retrieved Evidence and Agent Reports. Answer scoring uses the same token-level F1. Table 11 gives the report-specific context value and the room left for repair.

C.2 METHOD COVERAGE

Table 12 lists which methods run here. RelayCaching repairs the KV an agent produced while writing its report, so it runs on Agent Reports and not on Retrieved Evidence. KVCOMM is not included here: it corrects caches with offsets stored from earlier requests, and the fixed reports do not repeat.

C.3 DIAGNOSTIC CONTROLS

On Qwen3-8B, the share of the gap that position alignment leaves rises with the number of producers, reaching .76 (95% interval $[.46, 1.19]$) when the material is split across all producers. If CacheBlend recomputes randomly chosen tokens instead of the tokens it selects, at the same 15% budget, most of its gain disappears where reuse loses the most.

Every method reads the same fixed reports, written without the question, so score differences come from reuse. These results supplement the Retrieved Evidence evaluation.

D When Position Alignment Is Enough

When one agent hands a single cache to another, position alignment alone is enough. In this test, a Qwen3-8B agent reads the documents and writes a note, and a second agent answers from the note's KV cache without the documents (100 questions per subset). Direct reuse falls below the floor on all three QA subsets, and position alignment alone brings it back close to the ceiling. The loss therefore comes from positions, and position alignment removes it without recomputation. Room for repair appears when several separately cached blocks are combined.

E Method Matrix

Table 12 gives the full method matrix behind the class glyphs of the main boards and the supplementary report experiment. Every method follows its paper and official implementation, with minimal modifications to run on our models and inputs; trained methods use their published training recipes. Methods that cannot run in our setting keep a row, with the reason in a footnote.

Table 12: Method matrix. Classes: ① selective recomputation, ② attention calibration, ③ position alignment, ④ anchor-delta reuse, ⑤ latent communication, ⑥ fine-tuned repair, ⑦ trained soft-token adapter, ⑧ repair of generated KV, ⑨ compression under reuse. Prefix caching is not listed: on our inputs it equals full recomputation (Appendix K). Cell legend: ✓ = evaluated; × = not evaluated (reason in the lettered footnote); N/A = does not apply to this setting; ref. = reference, not ranked.

Method	Class	Train	Retr. Evid.	Reports (supp.)
CacheBlend (Yao et al., 2025)	①	none	✓	✓
LegoLink (EPIC) (Hu et al., 2025)	①	none	✓	✓
DroidSpeak (Liu et al., 2025)	①	none	×[f]	×[f]
APE (Yang et al., 2025b)	②	none	✓	✓
Position alignment	③	none	ref.	ref.
KVCOMM (Ye et al., 2025)	④	none	✓[a]	N/A[a]
LatentMAS (Zou et al., 2026)	⑤	none	N/A[d]	N/A[d]
Block-Attention (Ma et al., 2025)	⑥	8.2 GPU-h	✓	✓
TurboRAG (Lu et al., 2024)	⑥	training	N/A[b]	N/A[b]
KVLink (Yang et al., 2025a)	⑥	training	×[c]	×[c]
KVPacket (Chen et al., 2026)	⑦	adapter	✓	✓
RelayCaching (Geng et al., 2026)	⑧	none	N/A[e]	✓
4-bit cache quantization	⑨	none	✓	✓
SnapKV / Knorm (eviction) (Li et al., 2024; Devoto et al., 2024)	⑨	none	✓	✓
TOVA / StreamingLLM (eviction) (Oren et al., 2024; Xiao et al., 2024)	⑨	none	✓	✓

[a] KVCOMM stores offsets from earlier requests and reuses a cache when its document appears again. The leaderboard runs it on the questions in order, where 45%, 25%, and 0% of scientific-paper, single-document, and multi-hop questions reuse the cache (Table 2). The fixed reports do not repeat, so it does not apply to Agent Reports.
[b] No released training code.
[c] Not run at the training scale of its published recipe.
[d] LatentMAS changes what agents exchange (latent states instead of text) rather than repairing a cache.
[e] RelayCaching repairs the KV an agent produced while generating its output; Retrieved Evidence has no generated text to hand over.
[f] No released code.

F Latency and Memory by Dataset

These dataset-level views complement the main-text Pareto overview in Figure 5. They use the Qwen3-8B board; Table 4 in Section 3.3 averages KV memory over all five models.

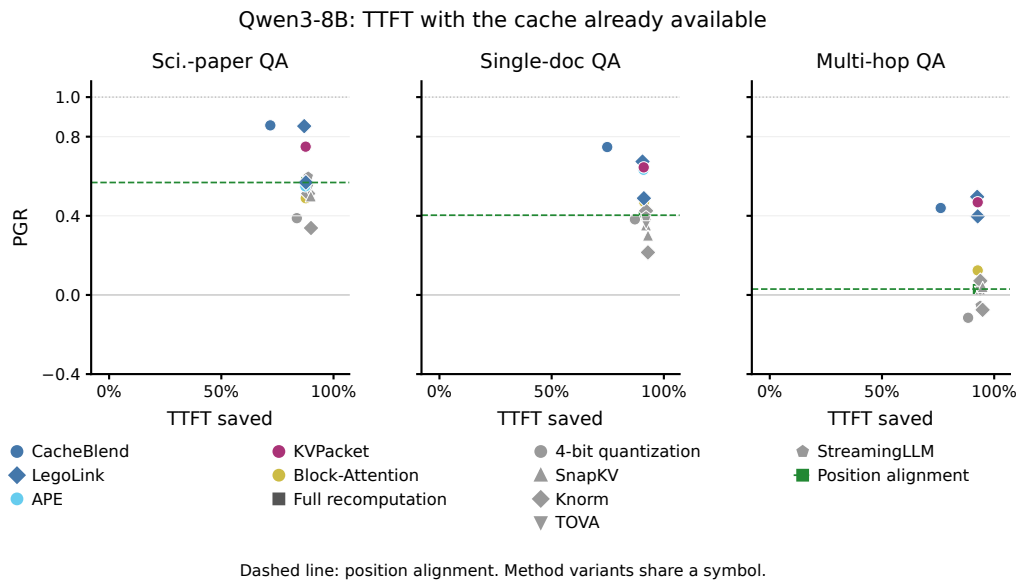


Figure 8: On Qwen3-8B, methods that recompute more tokens save less TTFT. Repairs that recompute no document tokens save .87–.93 of TTFT; CacheBlend, run at a 15% recomputation budget, saves .72–.76. Per-request time to first token with the cache already available, all methods on one backend against the same full recomputation. Color = method class as in Figure 6; configurations of a method share a symbol.

F.1 Break-even with deployment numbers

A deployer can compute the break-even point with their own numbers: a cache pays for itself after $T_{\text{build}}/(T_{\text{full}} - T_{\text{method}} - B_{\text{method}}/\text{bandwidth})$ reads, where T_{build} is the time to build the cache and bandwidth is the speed of loading it. The benchmark reports T_{full}, T_{method}, and B_{method} (§2.3); the build time and the bandwidth depend on the deployment. How often a workload reads a cache decides how soon the cache reaches this count. Prior work has measured both read patterns and loading speed, in serving engines and on traces. RAGCache reports skewed retrieval: on its MMLU workload, the top 3% of documents serve 60% of requests. Its cache hits are up to 3.9× faster than full prefill, counting the host-to-GPU transfer (Jin et al., 2024). CacheBlend loads one layer's KV cache for a 4K-token Llama-7B context from an NVMe SSD in 16 ms. It overlaps selective recomputation with this loading and reports a 2.2–3.3× TTFT reduction against full recomputation in its serving stack (Yao et al., 2025). KVCOMM measures 68–88% of tokens reusable across two to five agents in multi-agent RAG, math, and coding pipelines (Ye et al., 2025), and DroidSpeak reports 3.1× faster prefill across two GPU nodes over InfiniBand (Liu et al., 2025). Our per-request savings with the cache already available are in Figure 8. A cache streamed from slower storage adds $B_{\text{method}}/\text{bandwidth}$ to every read, and Table 4 gives B_{method}.

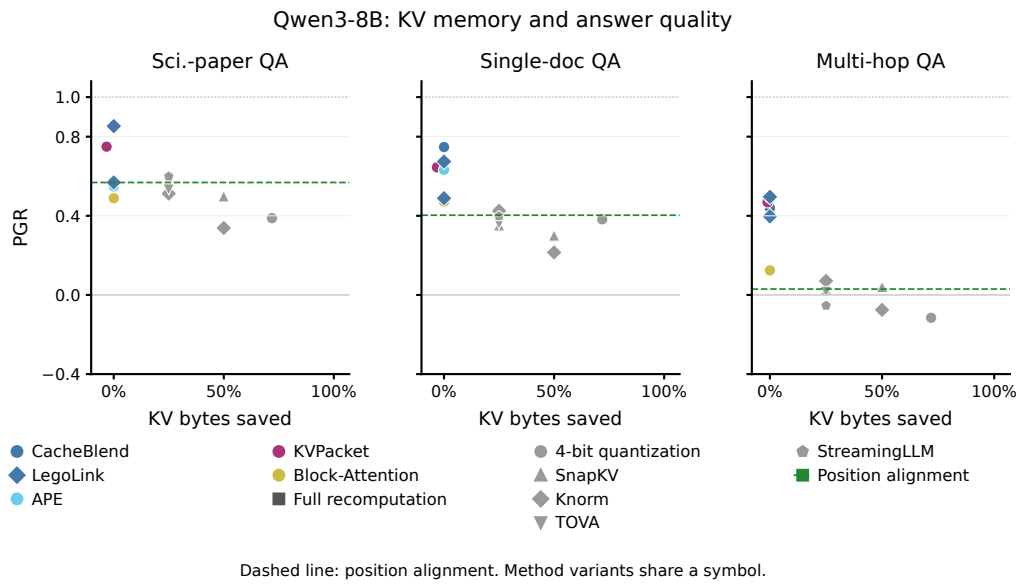


Figure 9: Memory cost: only compression (gray) saves KV memory, at a cost in quality; every other method stores the full cache, and KVPacket stores about 3% extra. Bytes are measured on the actual tensors, so 4-bit quantization saves a measured 71.9% rather than the nominal 75%. Color = method class; configurations of a method share a symbol. The dashed line marks position alignment.

G Dataset Cards

Retrieved Evidence subsets. The question sets contain 1,000 Qasper questions, 150 MultiFieldQA questions, and 1,000 HotpotQA questions: 2,150 unique questions in total, covering scientific-paper, single-document, and multi-hop QA. LongBench v1 provides 200, 150, and 200 of them. The added Qasper questions come from the official Qasper development and test splits, formatted with LongBench's own procedure, which reproduces LongBench's Qasper samples exactly; the 1,000 questions cover 547 papers. HotpotQA keeps LongBench's 200 questions unchanged. The added questions come from the HotpotQA development split (distractor setting) and are assembled as in LongBench: each passage is the full Wikipedia article of a supporting or distractor title, taken from a fixed Wikipedia dump, under a length budget matched to LongBench's. We drew 800 of them with a fixed seed from the candidates whose supporting titles are in the dump and that fit the budget. MultiFieldQA uses all 150 LongBench questions. No question appears in the training data of the trained methods: the Block-Attention fine-tuning pool, HotpotQA train, and MuSiQue train; KVPacket's other training data are synthetic. We use natural passage boundaries where provided and 512-token windows otherwise, with an input cap of 29K tokens. Context value (ceiling−floor) on the 8B reference board is .326/.361/.338 F1.

Dataset composition. Figure 10 shows the topics, the number of blocks per question, and the full-recomputation F1 of the question sets. Topics fall into nine categories. Each question has 2–43 blocks, each a passage or a 512-token window encoded on its own. The F1 distribution uses Qwen3-8B's full-recomputation answers.

Supplementary reports. The report construction, its floor and ceiling, and the results appear together in Appendix C.

G.1 Context value and room for repair

Two measured quantities describe each board and subset (Table 5, §3.5). The context is *worth* ceiling−floor. Position alignment *leaves* the share $(\text{ceiling} - S_{\text{align}})/(\text{ceiling} - \text{floor})$ to repair, where S_{align} is the score of position alignment. If a model answers nearly as well without the documents, the context is worth little. If a model reads separately encoded caches well, little is left

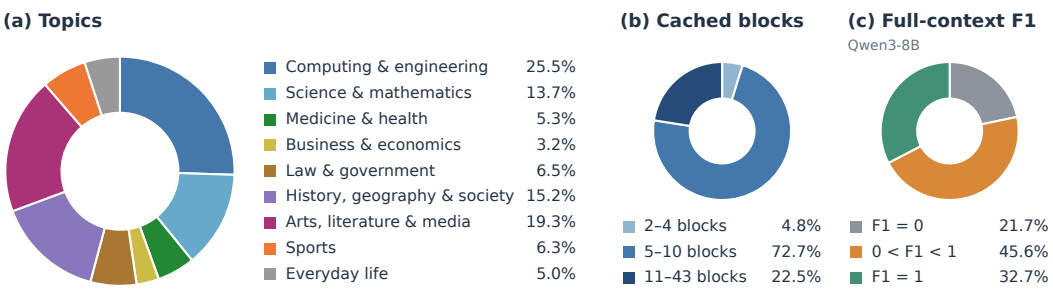


Figure 10: **Dataset composition.** **(a)** Topic distribution. **(b)** Number of independently cached blocks per question. **(c)** Answer F1 after full recomputation with Qwen3-8B.

to repair. In both cases every method scores close to position alignment, so the methods cannot be told apart. The table also helps read the results: repair matters where both numbers are large and fades where little is left. The floor, ceiling, position alignment, and prefix caching are reference points and are not ranked (Appendix K).

H Run Settings

Table 13 lists the settings shared by every method.

Table 13: **Every method runs on the same models, inputs, and scoring, and is timed on one shared backend.** Cross-checkpoint producers are listed in Table 16.

Setting	Value
Models	Qwen3-4B, Qwen3-8B, Qwen3-14B, Qwen3-30B-A3B-Instruct-2507 (mixture of experts), and Llama-3.1-8B-Instruct; on each board the model writes and reads its own caches
Input	up to 29K tokens; natural passage boundaries where provided, otherwise 512-token windows; 2–43 cached blocks per question
Scoring	LongBench's official token-level F1; PGR against each board's floor and ceiling
Timing	time to first token with the cache already available, on NVIDIA H100 GPUs with one shared vLLM backend; each method and full recomputation run on the same model and GPU

I Notes on the Per-Board Results

The per-board tables (Tables 6–10) show which gains hold on every model and which depend on the model. CacheBlend significantly improves scientific-paper and single-document QA on all five models. KVPacket and APE significantly improve multi-hop QA on all five models, and Block-Attention on four. LegoLink improves single-document and multi-hop QA on the four Qwen boards. KVCOMM gains on single-document and multi-hop QA mainly by recomputing the questions whose documents it has not seen. The supplementary report results are discussed in Appendix C.

Fine-tuning with Block-Attention's published recipe lowers the fine-tuned model's own ceiling on four of the five models (scientific-paper / single-document / multi-hop QA: 4B $-.20/-.12/-.33$, Llama $-.20/-.14/-.39$, 30B $-.10/-.05/-.17$; 14B $-.08$ on scientific-paper QA only). The 8B ceiling does not drop. With the fine-tuned model's own floor and ceiling, the multi-hop gap (ceiling$-$floor) falls below .10 on 4B and Llama. We therefore score every Block-Attention result against the base model's floor and ceiling.

J Robustness: Does a Repair Stay Effective When the Setting Changes?

Methods respond differently to changes of model and producer. A mean over boards hides whether a repair works everywhere or only somewhere, so Table 14 reports the worst case over settings beside the means, as WILDS reports worst-group accuracy beside average accuracy (Koh et al., 2021). We ask whether a repair's effect survives two changes of setting. *Across models*: a method's gain on a board is its PGR minus that of position alignment on the same board; the table gives the worst gain over the five boards and the number of boards on which the gain is positive. *Across producers*: in each of 18 tests (the six pairs of Table 16 on three subsets), we take the paired change in the method's own F1 when another checkpoint of the same family writes the cache. The table gives the mean and worst change with 95% intervals and the number of tests whose interval lies below zero. CacheBlend improves single-document QA on every model (worst gain $+.20$ PGR) and multi-hop QA on four of five. KVPacket improves on position alignment on every model and QA subset. It is also hurt most often by a producer change (12 of 18 tests significantly below zero; worst change $-.146$ F1). APE improves multi-hop QA on all five models, with a worst gain of $+.11$ PGR. LegoLink with k=0 also has one large producer drop ($-.151$ F1). A cache written by a model from a different pretraining run lowers every method (Table 15).

Table 14: Retrieved Evidence: worst-case results across the tested settings. Left: the worst gain over position alignment across the five boards (PGR units); superscript = boards with a positive gain / all boards; **bold** = positive on every board and significant on a majority. Right: the mean and worst paired change of the method's own F1 when a same-family checkpoint writes the cache ($\pm$ 95% interval), and the number of the 18 tests whose interval lies below zero. KVCOMM is left out: it recomputes most questions, so these columns would not measure reuse.

	Worst gain over position alignment		Producer change (F1)		
Method	Single-doc	Multi-hop	mean	worst	tests sig. below
③ Position alignment	–	–	$-.008\pm.005$	$-.035\pm.032$	4/18
① CacheBlend (15% recompute)	$\mathbf{.202}^{5/5}$	$-.077^{4/5}$	$-.005\pm.005$	$-.025\pm.030$	0/18
① LegoLink (k=2, ≈0.4% recompute)	$-.009^{4/5}$	$-.016^{4/5}$	$-.011\pm.005$	$-.037\pm.031$	5/18
① LegoLink (k=0, 0% recompute)	$-.014^{4/5}$	$-.028^{4/5}$	$-.019\pm.006$	$-.151\pm.024$	8/18
② APE	$.027^{5/5}$	$\mathbf{.110}^{5/5}$	$-.006\pm.005$	$-.040\pm.037$	6/18
⑥ Block-Attention	$-.088^{2/5}$	$\mathbf{.057}^{5/5}$	$-.004\pm.004$	$-.034\pm.022$	6/18
⑦ KVPacket (0% recompute)	$\mathbf{.109}^{5/5}$	$\mathbf{.201}^{5/5}$	$-.039\pm.007$	$-.146\pm.023$	12/18
⑨ Quant. 4-bit	$-.140^{0/5}$	$-.706^{0/5}$	$-.011\pm.005$	$-.025\pm.033$	2/18
⑨ SnapKV (r=.25)	$-.105^{0/5}$	$-.014^{1/5}$	$-.007\pm.005$	$-.038\pm.028$	2/18
⑨ SnapKV (r=.5)	$-.141^{0/5}$	$-.040^{3/5}$	$-.005\pm.004$	$-.038\pm.017$	4/18
⑨ Knorm (r=.25)	$-.168^{2/5}$	$.041^{5/5}$	$-.016\pm.005$	$-.034\pm.037$	8/18
⑨ Knorm (r=.5)	$-.342^{0/5}$	$-.142^{2/5}$	$.000\pm.004$	$-.007\pm.016$	0/18
⑨ TOVA (r=.25)	$-.093^{0/5}$	$-.042^{2/5}$	$-.006\pm.005$	$-.037\pm.036$	4/18
⑨ StreamingLLM (r=.25)	$-.113^{0/5}$	$-.307^{0/5}$	$.001\pm.004$	$-.030\pm.036$	0/18

K Relation to Prefix Caching

On our boards, prefix caching coincides with full recomputation, the reference point of the quality–cost plots. Prefix caching is lossless because it reuses KV only after the same text. On our inputs, a prefix cache matches only the fixed instruction at the start of each prompt, so a single request saves no document computation. We report it in two ways. Per request, it keeps full quality and saves no document computation. Over the full sequence of questions, its hit rate is 0.6% to 21.4% per subset. Hits beyond the instruction come from prefixes that samples share, and its quality does not differ significantly from full recomputation. Prefix caching and repair methods address different cases: prefix caching saves work when the preceding text is unchanged, and repair methods recover quality when it changes.

Serving engines and repair methods are complementary. CacheBlend already ships as a cache-layer plugin for these serving engines, and other repair classes could ship the same way. We read each method against full recomputation: how much cost it saves and how much quality it keeps.

Table 15: A cache written by a model from a different pretraining run lowers the F1 of every method. The receiver is Qwen3-30B-A3B-Instruct-2507 and the cache is written by Qwen3-Coder-30B-A3B-Instruct, which has the same architecture. Entries are the paired change in F1 relative to the receiver's own cache, on 100 questions per subset; $^{\downarrow}$ = the 95% interval lies below zero.

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	$-.165^{\downarrow}$	$-.183^{\downarrow}$	$-.214^{\downarrow}$
① LegoLink (k=2, ≈0.4% recompute)	$-.254^{\downarrow}$	$-.224^{\downarrow}$	$-.309^{\downarrow}$
① LegoLink (k=0, 0% recompute)	$-.275^{\downarrow}$	$-.230^{\downarrow}$	$-.320^{\downarrow}$
② APE	$-.255^{\downarrow}$	$-.195^{\downarrow}$	$-.190^{\downarrow}$
⑥ Block-Attention	$-.129^{\downarrow}$	$-.179^{\downarrow}$	$-.005$
⑦ KVPacket (0% recompute)	$-.141^{\downarrow}$	$-.217^{\downarrow}$	$-.287^{\downarrow}$
⑨ Quant. 4-bit	$-.197^{\downarrow}$	$-.183^{\downarrow}$	$-.138^{\downarrow}$
⑨ SnapKV (r=.25)	$-.160^{\downarrow}$	$-.199^{\downarrow}$	$-.210^{\downarrow}$
⑨ SnapKV (r=.5)	$-.164^{\downarrow}$	$-.158^{\downarrow}$	$-.235^{\downarrow}$
⑨ Knorm (r=.25)	$-.164^{\downarrow}$	$-.144^{\downarrow}$	$-.142^{\downarrow}$
⑨ Knorm (r=.5)	$-.171^{\downarrow}$	$-.051^{\downarrow}$	$-.116^{\downarrow}$
⑨ TOVA (r=.25)	$-.153^{\downarrow}$	$-.156^{\downarrow}$	$-.186^{\downarrow}$
⑨ StreamingLLM (r=.25)	$-.128^{\downarrow}$	$-.143^{\downarrow}$	$-.128^{\downarrow}$

L Open Platform and Submission Workflow

New methods use the same inputs and scoring rules as existing entries. The anonymized supplementary material provides the evaluation package, the fixed question sets, reference scores, and a static copy of the leaderboard. The package installs with pip (`pip install -e .`).

Adding a method. A method implements three calls: `setup` loads the model or engine once; `build_cache` encodes one block without seeing the question; `answer` reuses the caches and returns an answer with cost measurements. The cost fields record four numbers: document tokens recomputed, document tokens full recomputation would compute (both counted per layer), KV bytes, and time to first token. An example driver implements position alignment. Every method uses the same fixed inputs and reference scores.

Scoring and submission. `kva validate` checks the result format and fixed sample IDs. `kva score` re-scores the predictions on CPU and computes PGR from the released references. `kva submit` prepares a result file and a method card for a pull request. The card documents the method and how its costs were measured. Maintainers check the model and prompt settings, reference answers, recomputed scores, and documented costs before updating the leaderboard data.

Leaderboard and release coverage. The leaderboard page is built from a versioned result file. Speed is the share of TTFT saved, as in the main results, and users can set the weight between quality and speed. The static copy in the supplementary material holds the leaderboard results of this paper for the five receiver models. It includes the floor and ceiling of every model and subset, so a full run can be scored as PGR. The package generates answers for Retrieved Evidence and can also score Agent Reports answers.

M Cross-Checkpoint Supplement

Table 18 gives each method's F1 change on Llama-3.1-8B-Instruct when the base model writes the cache. Position alignment loses .026 on scientific-paper QA and .035 on single-document QA, both significant, and APE gains .051 on multi-hop QA.

M.1 How producer caches are injected

For a producer change, each method receives KV written by another checkpoint of the same architecture instead of the receiver's own. With the receiver's own KV, the same code reproduces every method's original outputs, so the swap itself changes nothing.

Table 16: **Every cross-checkpoint pair keeps the architecture and tokenizer and changes only the weights.** The receiver is the board's model; the producer writes the caches.

Receiver (board)	Producer	Relation
Qwen3-8B	Qwen3-8B-Base	base predecessor
Qwen3-8B	DeepSeek-R1-0528-Qwen3-8B	distilled sibling
Qwen3-4B	Qwen3-4B-Base	base predecessor
Qwen3-14B	Qwen3-14B-Base	base predecessor
Llama-3.1-8B-Instruct	Llama-3.1-8B	base predecessor
Qwen3-30B-A3B-Instruct-2507	Qwen3-30B-A3B-Thinking-2507	post-trained sibling

Table 17: Without repair, changing the context or the producer lowers F1. Scores are F1 on Retrieved Evidence with Qwen3-8B as the receiver; where the producer differs, the two values are for the base predecessor and the distilled sibling (Table 16).

Context	Producer weights	HotpotQA	Qasper
Same	Same as receiver	.559	.417
Same	Different	.484/.527	.353/.410
Different	Same as receiver	.394	.397
Different	Different	.361/.315	.381/.344

M.2 Answers of the trained adapter under a producer change

Where KVPacket's adapter loses quality under a different producer, its answers remain fluent but get facts wrong (for example, a wrong actress, a flipped yes/no, or a year attached to the wrong event).

M.3 Retraining the adapter per producer

To test whether retraining alone restores quality, we retrained the KVPacket adapter with its published recipe and budget on caches written by each new producer, with the receiver fixed. Three adapters were evaluated on the same six tests under the same setup: the original adapter of the main board, the adapter retrained on the producer under test, and the adapter retrained on the other new producer (cross). The retrained adapter shows no significant difference from the same-model baseline in any of the six tests. It is significantly above the original adapter in the five tests where the original adapter had lost quality, by up to .25 F1. The cross-trained adapter recovers part of the loss with the distilled sibling as producer and none with the base predecessor, and stays below the baseline in every test. Each retraining took about 20 GPU-hours.

N Leaderboard Score Under Other Weights

Table 20 lists the leaderboard score of §2.3 under three quality weights: 0.7 (quality first), 0.5 (the default), and 0.3 (speed first). KVPacket and LegoLink lead under all three weights, and repair methods take the top five places. CacheBlend, which gives up speed for quality, ranks third at 0.7 and eleventh at 0.3; KVCOMM ranks sixth at 0.7 and last at 0.5 and 0.3.

Table 18: Llama-3.1-8B base → Instruct: paired F1 change after replacing the cache producer. Each method is compared with its own same-model result on the same questions. $*$ marks a paired 95% confidence interval excluding zero.

Method	Sci.-paper QA	Single-doc QA	Multi-hop QA
③ Position alignment	−.026*	−.035*	−.009
① CacheBlend (15% recompute)	−.010	−.025	−.005
① LegoLink (k=2, ≈0.4% recompute)	−.023*	−.037*	−.003
① LegoLink (k=0, 0% recompute)	.003	−.039*	−.023*
② APE	.015	−.040*	.051*
⑥ Block-Attention	−.011*	−.034*	−.013*
⑦ KVPacket (0% recompute)	−.013	−.068*	−.027*
⑨ Quant. 4-bit	−.023*	−.025	.000
⑨ SnapKV (r=.25)	−.019*	−.038*	.009
⑨ SnapKV (r=.5)	−.011	−.037*	.017
⑨ Knorm (r=.25)	−.021*	−.034	−.001
⑨ Knorm (r=.5)	−.006	−.006	−.006
⑨ TOVA (r=.25)	−.018*	−.037*	−.003
⑨ StreamingLLM (r=.25)	−.008	−.030	.038*

Table 19: KVPacket adapter under a different producer (receiver Qwen3-8B; F1). Same-model baseline: .407/.401/.447 on Sci.-paper / Single-doc / Multi-hop QA. Bold = paired 95% CI against the same-model baseline excludes zero.

Producer	Adapter	Sci.-paper QA	Single-doc QA	Multi-hop QA
Distilled sibling	original	**.340**	**.292**	**.248**
	retrained on this producer	.398	.374	.497
	cross-trained (on the base producer)	.381	.353	.398
Base predecessor	original	.369	**.347**	.467
	retrained on this producer	.411	.427	.442
	cross-trained (on the sibling)	.371	.346	.408

Table 20: Leaderboard score under three quality weights; 0.5 is the default of Table 2. Rows are ordered by the default; ranks under each weight are in parentheses. *Recomputes the questions whose documents it has not seen.

Method	quality first (0.7)	balanced (0.5)	speed first (0.3)
⑦ KVPacket (0% recompute)	.746 (1)	.789 (1)	.832 (1)
① LegoLink (k=2, ≈0.4% recompute)	.733 (2)	.778 (2)	.824 (2)
② APE	.689 (4)	.748 (3)	.807 (3)
① LegoLink (k=0, 0% recompute)	.672 (5)	.736 (4)	.801 (4)
① CacheBlend (15% recompute)	.713 (3)	.720 (5)	.726 (11)
⑥ Block-Attention	.598 (7)	.684 (6)	.769 (5)
⑨ SnapKV (r=.25)	.577 (8)	.671 (7)	.766 (6)
⑨ Knorm (r=.25)	.576 (9)	.671 (8)	.765 (7)
⑨ TOVA (r=.25)	.567 (10)	.664 (9)	.762 (9)
⑨ SnapKV (r=.5)	.555 (11)	.659 (10)	.762 (8)
⑨ StreamingLLM (r=.25)	.544 (12)	.648 (11)	.752 (10)
⑨ Quant. 4-bit	.490 (13)	.596 (12)	.702 (13)
⑨ Knorm (r=.5)	.466 (14)	.595 (13)	.724 (12)
④ KVCOMM*	.614 (6)	.490 (14)	.365 (14)