RAPID: EXTREME KV CACHE COMPRESSION AT NEAR-FULL QUALITY FROM THE CONTEXT ALONE

Anonymous authors
Paper under double-blind review

ABSTRACT

The size of KV caches is a dominant bottleneck in long-context model serving. Modern KV compression algorithms fall short on at least one of the following objectives. 1. To maintain uncompressed generation quality. 2. To compress fast, without heavy training, data generation, or model use (beyond context itself). 3. To achieve best possible compression: SOTA as of mid-2026 is over $10\times$ rate as compared to full KV cache storage at half-precision floating points.

We introduce RAPID algorithm (Rank And Precision allocation with Independent per-layer Distillation) which our evaluations show to be the closest to satisfying all three objectives. RAPID combines layer-wise latent representations, cross layer budget allocation, per token dimension reduction, per dimension quantization, and intra-layer optimization using prefill query vectors only. RAPID doesn't need training data and access to the model beyond each layer's attention projections. A variant of our algorithm, RAPID++, is always within 2.5 points of the full cache quality at $20\times$ compression, on every evaluated benchmark and model, including Qwen3-4B, a 117B-parameter mixture-of-experts GPT-OSS-120B, and an MLA model whose cache is already compressed.

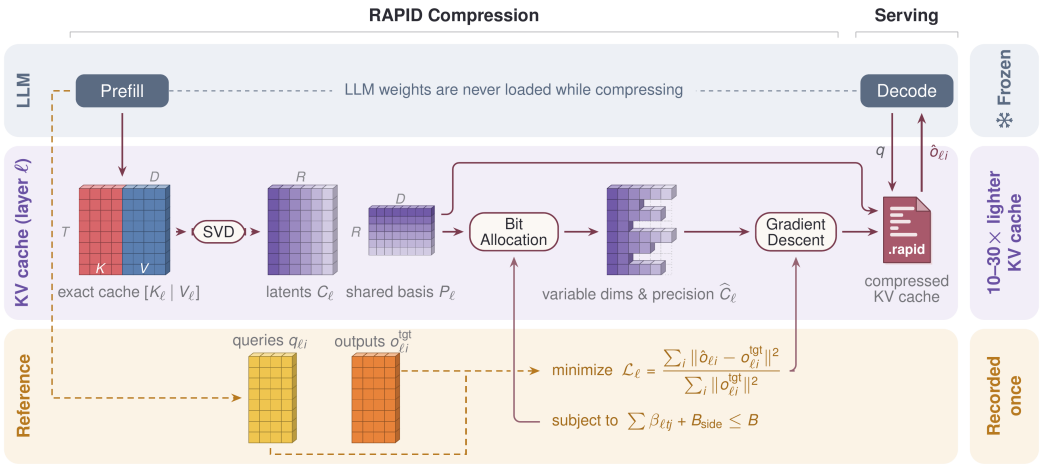
1 INTRODUCTION

KV cache compression emerged as a central challenge in transformers inference for long contexts. As the KV cache grows proportionately to the context length, it puts pressure on both GPU memory consumption/bandwidth and decode time, which is memory-bound. The main objective of KV cache compression is to alleviate these pressures by reducing the size of the KV cache.

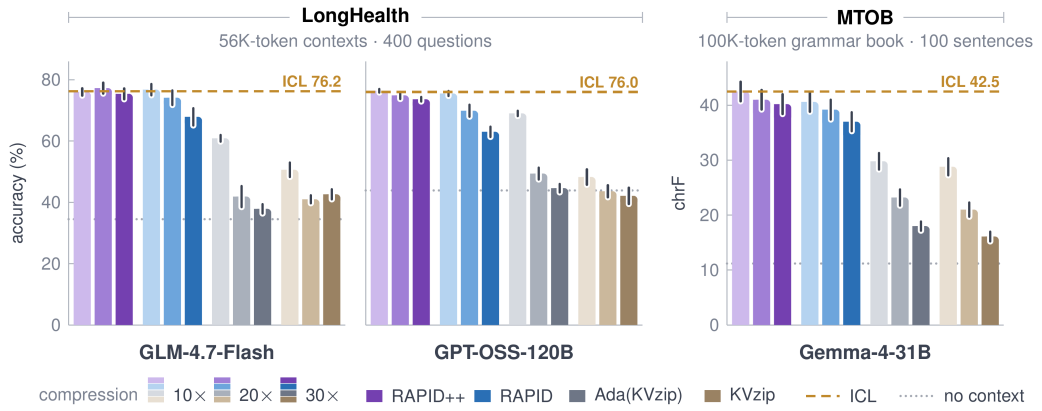
We identify three key properties of compression algorithms that are necessary for wider adoption in practice. These are pragmatic "real world" considerations regarding model serving economics and user preferences rather than fundamental hard limits. *Quality*: Compression should preserve the generation quality when compared to using the uncompressed KV cache. We find that even cost and latency sensitive users are reluctant to sacrifice generation quality, especially since that degradation is effectively impossible to quantify apriori. *Efficiency*: Producing the compressed format should be sufficiently fast and cost efficient. A natural reference cost is the prefill time (generating the uncompressed KV cache), and we require that compression is not substantially more demanding, and hence cannot use heavy end-to-end gradient flows, generated training data, or require the full model in-memory (beyond the cache itself). *Compression rate*: it should achieve competitive compression rates which are, at the time of writing this paper, $20\times$-$50\times$ depending on the model.

Existing techniques offer good trade-offs among the three objectives but, we argue none achieves all three at once. At one end of the spectrum, training-free methods compress the cache by evicting tokens (Zhang et al., 2023; Li et al., 2024; Cai et al., 2024; Feng et al., 2024; Kim et al., 2025a), reducing dimension (Saxena et al., 2024; Zhang et al., 2024a; Chang et al., 2025a;b), or quantizing coordinates (Liu et al., 2024b; Hooper et al., 2024; Zandieh et al., 2025), recently also in combination (Miao et al., 2026; Zhang et al., 2026). They are efficient, but preserve quality only at moderate rates (roughly $\leq 10\times$). We note that there are theoretical lowers bounds on required size (Chen et al., 2026; Liberty et al., 2026; Cohen, 2026). At the other end, Cartridges (Eyuboglu et al., 2025) trains a compact cache end to end on model-generated conversations about the context (*self-study*). It reaches high quality at high ratios, but each cache costs orders of magnitude more than

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) RAPID overview



(b) RAPID results across models and benchmarks

Figure 1: **(a)** RAPID Overview. **(b)** At $10\times$ to $30\times$ compression, RAPID++ stays at or near the uncompressed cache quality on medical question answering benchmarks (LongHealth), and on translation tasks at 100K context window (MTOB). Error bars are ± 1 standard error.

a prefill. In between, Attention Matching (Zweiger et al., 2026) and KVSculpt (Jiang & Jin, 2026) fit each attention head locally to queries drawn from the context. They are faster and are the closest methods to ours; they hold up at $20\times$ but fall well short of the uncompressed cache at $50\times$ and on questions that need more than one fact (Section 4.2). As we show in our evaluations, methods trained on generated conversations recover accuracy at the price of over-specialization (see Sec. 4, Appendix I.3).

There is a large design space for KV cache compression and we highlight four aspects that govern design of nearly all such algorithms.

- *Representation of the compressed cache:* the form in which the compressed key/values vectors are stored. The KV cache is typically stored as a set of compressed vectors as mentioned above (proper or synthetic coreset, reduced dimension or precision). Additional options include: co-compressing different heads in the same layer or even all the layers (cf. compressing each head separately). See other methods in Related Work (Sec. 2).
- *Allocation of the storage budget across the model:* whether each head's key/value cache takes equal space or the budgets are unequal (e.g., as a function of the influence on the overall error). The latter methods yield better results by compressing more the heads that tolerate it.

- *Optimization objective:* what loss objective drives the compression. Train-free methods typically seek to optimize for a head/layer-local objective (e.g. attention), where as end-to-end method optimize the overall model performance (e.g., cross-entropy). The former is preferred since the latter is very expensive, requiring storing the entire model in-memory and running gradient flow through all the layers.
- *Training data:* what data is used in the above optimization. Most algorithms, even many training-free ones, minimize the error captured by the optimization objective, for a given set of "training" queries Q. (There are a few exceptions that use $Q = \emptyset$, relying only on the original key/value pairs). The size of Q can vary significantly: from context queries (the query vectors from the context itself), to a generated set of queries from the context (such as the self-study). The latter option is expensive both to generate, as well as to train with. More subtly, a specialized training data may overspecialize the cache, and hurt quality on out-of-distribution queries. Hence prefill queries are much more desirable.

In our algorithm design, we approach the KV cache compression problem holistically, aiming to optimize for all four aspects at the same time to obtain an efficient and accurate cache compression.

1.1 Our approach

We set up a rate-distortion problem over the entire cache as follows. We represent our compressed KV cache of T tokens as: the cache from all the heads of layer ℓ are co-compressed into T shared latent vectors $\{C_{\ell,t}\}_{t\in[T]}$. Each latent vector $C_{\ell,t}$ gets a dimension budget, and each coordinate gets a bit budget. We capture this by parameter $\beta_{\ell tj} \geq 0$ that denotes the number of bits assigned to coordinate j of token t at layer ℓ; also, let θ collect the other shared (independent-of-T) parameters (e.g., latent space projection). Our training data consists of the context queries (obtained from prefill itself), denoted by Q_ℓ for layer ℓ. Let $o^{\text{tgt}}_{\ell,i}$ and $\hat{o}_{\ell,i}(\beta,\theta)$ denote the exact and compressed attention outputs computed for query at position $i \in [T]$, evaluated with the same exact (teacher-forced) layer inputs. The compressed representation is computed as an optimization problem minimizing the following error measure over admissible cache representations:

$$\begin{aligned} \min_{\beta,\theta} \quad & \frac{1}{L}\sum_{\ell=1}^{L} \frac{\sum_i \left\|\hat{o}_{\ell,i}(\beta,\theta) - o^{\text{tgt}}_{\ell,i}\right\|_2^2}{\sum_i \left\|o^{\text{tgt}}_{\ell,i}\right\|_2^2} \\ \text{s.t.} \quad & \sum_{\ell,t,j} \beta_{\ell tj} + B_{\text{side}}(\beta,\theta) \leq B, \end{aligned} \tag{1}$$

where B_{side} includes the stored projections, quantization codebooks, and layout metadata. We note that we normalize the error of attention output in order to prevent the activation scale alone from determining the allocation.

We introduce RAPID (Rank And Precision allocation with Independent per-layer Distillation), a three-step algorithm to find a good solution to this objective. First, for each layer, RAPID factorizes its key/value vectors into a shared latent representation (standard SVD). Second, it allocates the global budget using a separable distortion surrogate that trades off dimension allocation with measured quantization error. A Lagrangian allocator determines which latent coordinates are retained and at what precision, considering the cost of side information. In the second stage, we refine the solution using a small number of (quantization-aware) gradient steps against Eqn. (1), with the allocation and storage layout held fixed. Note that each layer can be optimized independently (and in parallel), without ever needing anything beyond the context itself.

The training dataset is composed of the queries from the context itself, extracted during the prefill. We also consider an extension, RAPID++ algorithm, that augments the training dataset with repeat-context queries that attend to the full cache, following the reference-query strategy of Zweiger et al. (2026) (extra T queries).

1.2 Our experimental results

We evaluate five models with diverse attention architecture: grouped-query attention (Qwen3-4B, Llama-3.1-8B), multi-head latent attention (GLM-4.7-Flash, whose cache is already a compressed latent), and hybrid local–global attention (Gemma-4-31B and the 117B-parameter mixture-of-experts GPT-OSS-120B), on LongHealth, MTOB, and QASPER. Full results appear Section 4.

Below we highlight the empirical findings in testing our algorithms RAPID and RAPID++.

Quality at high compression. At $10\times$, RAPID++ stays within 1.5 points of the uncompressed cache on every model and benchmark. At $20\times$, it retains on average 98% (and at least 95%) of the uncompressed cache's score on every model and benchmark (Figure 1). At $30\times$ some quality is lost, though RAPID++ still decidedly outperforms all other methods tested.

Generalization across tasks. RAPID++ is fitted only to queries from the context itself, never to an assumed training dataset. To test how far generalization carries, we extend LongHealth with new questions that requires precise details among close distractors, comparisons across records, and links between events in a the context (Section 4.2). These questions separate the methods far more than the original ones. At $10\times$, averaged over the models, RAPID++ keeps 99 percent of the uncompressed cache's accuracy on both the original and the new questions, while Attention Matching falls from 97 to 80 percent and the best eviction baseline, Ada(KVzip), from 88 to 64. The reason is visible in the quantity RAPID minimizes: on the question tokens at $20\times$, RAPID++ has 2.2 to $5.5\times$ lower attention-output error than the best baseline and 2.7 to $6.3\times$ lower than Attention Matching, and its attention output error on the new questions is the same as on the original ones.

Stronger initialization for end-to-end training. Our evaluations also suggest that our algorithm are good initializations for end-to-end training at extreme compression. In particular, when self-study data are available, starting from a RAPID++ cache and running the Cartridge-like end-to-end optimization reaches the quality of uncompressed cache within a few percent of an epoch and then exceeds it. On a 110K-token context at $53\times$, 16 steps (1.2% of an epoch) lift RAPID++ from 51.8 to 60.0 percent accuracy, above the uncompressed cache quality (53.0), while a Cartridge trained from its own initialization reaches 46 percent after 2,048 steps (see Sec. J.1).

Together, these results are a strong evidence that gradient descent on (per-layer) attention outputs recovers much of what initial compression loses. At the same time, the outcome of the optimization depends on where it starts: with a more favorable initialization from a better approximation, the per-layer refinement and end-to-end training can reach better overall solutions.

1.3 Conclusion

RAPID compresses the KV cache of a frozen model by factorizing each layer's cache into a shared basis and per-token latents, allocating bit budget across layers, tokens, and coordinates, and refining each layer independently against its own attention output on queries taken from the context. Our algorithm avoids any end-to-end training and generated data: compression is a few dozen attention passes per layer, with the model never loaded. At $20\times$, RAPID++ keeps on average 98% of the uncompressed cache's score across five models and three benchmarks; at $50\times$ it leads the strongest baseline on all four models run at that ratio; and its cache is the better starting point when end-to-end training is affordable. Remaining future work includes: implementing the decoding kernel, closing the gap that per-layer training leaves at $50\times$ and beyond without end-to-end data, and compressing a growing context in a streaming fashion. See Section 5 for limitations and future work.

2 Related Work

We compare our algorithm to the most related prior work. See Appendix A for further related work.

Trained and fitted caches. Cartridges (Eyuboglu et al., 2025) trains a compact KV cache end to end, with the model frozen, on *self-study* data (model-generated conversations about the context). Its quality improves over thousands of gradient steps, each requiring backpropagation through the model on self-study data. Diaz (Diaz, 2025) improves convergence through Sampled Chunk Initialization; GradMem (Kuratov et al., 2026) enables few-step memory writing by meta-training the memory initialization and model, which makes writing fast but ties the memory to the meta-trained model and budget. Attention Matching (Zweiger et al., 2026) and KVSculpt (Jiang & Jin, 2026) instead fit each head locally, in closed form or by gradient descent.

RAPID uses the cheaper local objective and context-derived queries only. It differs in the representation: a factorized, per-token variable-rank, quantized latent whose layout is allocated across the whole model within a bit budget, and refined afterwards. In our evaluations, we also found that self-study-trained caches exhibit failures on out-of-distribution multi-hop reasoning and on recog-

nizing when the context lacks the requested information, consistent with overfitting to the generated interactions (Appendix 4.2, I.3). We also run self-study on top of RAPID caches (Appendix J.1) to separate the effect of the training data from that of the representation.

Training-free compression. Eviction keeps a subset of KV pairs chosen by various scores: by attention (Zhang et al., 2023; Li et al., 2024; Cai et al., 2024; Feng et al., 2024), by key geometry (Park et al., 2025), or by the attention received while the model reconstructs the context (Kim et al., 2025a). A matched-budget audit (Luo et al., 2026) finds that query-aware scorers lose most of their advantage when the question is not visible at compression time, so we run every eviction baseline query-agnostically and report the query-aware variant separately. Quantization (Liu et al., 2024b; Hooper et al., 2024; Zandieh et al., 2025; Zuo et al., 2026) keeps every token and coordinate, so its ratio is bounded by the cheapest bit width, typically $4\times$–$8\times$. Low-rank methods project the cache onto a basis fixed per model (Saxena et al., 2024; Zhang et al., 2024a; Chang et al., 2025a; Bhatnagar et al., 2026) or computed per context (Chang et al., 2025b; Sun et al., 2024a; Zhu et al., 2026); in either case the rank is set per layer or head, not per token, and nothing is refined. In (1) the three are ways of spending the same bits; eviction is the case in which every retained coordinate keeps full precision, and the theoretical attention-coreset lower bounds of Chen et al. (2026); Liberty et al. (2026); Cohen (2026) show that this case can require many retained pairs.

Joint allocation of rank, precision, and tokens. Several recent methods allocate more than one axis under a common budget, and three of them, like RAPID, let the rank vary per token. MixedDimKV (Miao et al., 2026) projects each head onto a PCA basis of the current context, gives every token one of four candidate dimensions, and solves the allocation as a Lagrangian dual by bisection; VarRate (Esmat et al., 2026) water-fills a graded prefix rank per token, above a floor, over a basis fixed per model from calibration data; DynaKV (Lu et al., 2026) learns a gate that sets each token's rank from its content. All three allocate rank only, take their importance signal from the last tokens of the prompt (the question, when present) or from the token itself, stop at the allocation, and are evaluated mainly at $5\times$–$16\times$ with the question in the prompt; VarRate reports losing 3.5–5.5 points without it. RDKV (Zhang et al., 2026) treats eviction and quantization as endpoints of one bit-allocation problem solved by reverse water-filling; MoE-nD (Sun et al., 2026) picks a per-layer eviction and quantization configuration greedily; KV-COBRA (Ha et al., 2026) co-optimizes a rank and a bit width per head on the spectrum of the key projection, and JoLT (Krishnan & Schulz, 2026) allocates Tucker ranks and residual bit widths per layer group with a single Lagrangian dual. RAPID shares the rate–distortion view of all of these and differs in four respects. It allocates rank and precision together, per token and per coordinate, and across layers by a measured attention-output sensitivity rather than an even split. Its signal comes from the context's own queries at every position, so the cache is built without the question. The allocation is an initialization, refined by a few layer-local gradient steps against the objective that produced its layer signal. And its target is a reusable cache at $20\times$–$100\times$, where the allocated approximation alone is no longer sufficient (Figure 2) and the refinement carries most of the recovered quality.

3 Our compression algorithms: RAPID and RAPID++

We compress the prefilled cache of an L-layer decoder on a context of T tokens. Layer ℓ caches a key and a value row per token over its H_{KV} KV heads of dimension d, $K_\ell, V_\ell \in \mathbb{R}^{T\times H_{KV}d}$ (keys after RoPE). Let $D = 2H_{KV}d$ be the width one token occupies in one layer. The reference queries of RAPID are those the model forms while prefilling the context; RAPID++ adds *repeat-context* queries, from feeding the context once more after the exact cache, so that every query reads all T tokens (Kim et al., 2025a; Zweiger et al., 2026). The two variants differ only in this set. We approach solving objective Eqn. (1) in three steps (Fig. 1). The full description of the algorithm is in Appendix B, and an ablation of the contribution of each algorithmic aspect is in Appendix F.

3.1 Step 1: a per-token, variable-rank latent

In a layer ℓ, the key and value vectors of all heads are factorized jointly by one SVD after scaling to equal Frobenius norm. A token t stores a prefix of its latent row (dimension $r_{\ell,t}$). Each kept

coordinate j is quantized by a per-coordinate codebook $\mathcal{Q}_{\ell j}$ of $2^{b_{\ell j}}$ levels:

$$\big[K_\ell/s_{K,\ell} \,\big|\, V_\ell/s_{V,\ell}\big] = \underbrace{U_\ell \Sigma_\ell^{1/2}}_{C_\ell \in \mathbb{R}^{T\times R}} \underbrace{\Sigma_\ell^{1/2} W_\ell^\top}_{P_\ell \in \mathbb{R}^{R\times D}}, \qquad \widehat{C}_\ell[t,j] = \begin{cases} \mathcal{Q}_{\ell j}(C_\ell[t,j]), & j \le r_{\ell t}, \\ 0, & j > r_{\ell t}, \end{cases} \tag{2}$$

where $s_{K,\ell} = \|K_\ell\|_F, s_{V,\ell} = \|V_\ell\|_F$. The allocation therefore decides a rank $r_{\ell t}$ per token and a bit-width $b_{\ell j}$ per coordinate, so that $\beta_{\ell t j} = b_{\ell j}\mathbf{1}\{j \le r_{\ell t}\}$ in (1). Note that $r_{\ell t} = 0$ corresponds to eviction. Prefixes keep representation profile description: one length per token ($r_{\ell t}$) and one width code per coordinate ($b_{\ell j}$), which B_{side} counts together with P_ℓ and the codebooks ($\mathcal{Q}_{\ell j}$). At decode, attention runs in the latent space, as in MLA (DeepSeek-AI, 2024): the query is multiplied into the key half of P_ℓ and the attention-weighted latent into its value half, so K_ℓ, V_ℓ are never rebuilt and the data transferred to GPU memory shrinks with the compression ratio.

3.2 Step 2: budget allocation across layers, tokens, and coordinates

We choose r and b by minimizing a separable surrogate of (1), in which storing coordinate (ℓ, t, j) at b bits costs

$$c_{\ell t j}(b) = g_{\ell j}\, w_{\ell t}\, \sigma_{\ell j} \cdot \begin{cases} C_\ell[t,j]^2, & b = 0, \\ e_{\ell j}(b), & b > 0. \end{cases} \tag{3}$$

Here $w_{\ell t}$ is the importance of token t, which we define as the largest attention any reference query places on it; $e_{\ell j}(b)$ is the importance-weighted error of a b-bit Lloyd–Max codebook on coordinate j; $\sigma_{\ell j}$ is the energy the coordinate carries through P_ℓ; and $g_{\ell j}$ is measured to be proportional to the importance of latent coordinate j to the attention output of layer ℓ. Because g is measured on the attention output (vs SVD spectrum), it balances the errors across layers, allocating more of the budget to the layers that are harder to compress. See more allocation details in Appendix G.

A single price λ per stored bit, shared by all layers, turns the budget into a Lagrangian that splits into one problem per layer. Within a layer we alternate two exact updates, each token choosing its best prefix given the widths and each coordinate its best width given the prefixes, and bisect λ until the cache fits in B. At the result every stored bit, in any layer, saves at least λ of surrogate error and every dropped bit at most λ, the equal-slope condition of Lagrangian bit allocation (Everett, 1963; Shoham & Gersho, 1988). P_ℓ is then refit to the quantized latents by least squares.

3.3 Step 3: independent per-layer refinement

With the layout and codebooks fixed, Objective (1) is differentiable in the stored values. For each layer we optimize a continuous latent Z_ℓ, initialized at the Step 2 latent, and the projection P_ℓ:

$$\min_{Z_\ell, P_\ell} \; \mathcal{L}_\ell\big(\Pi_\ell \odot \mathcal{Q}_\ell(Z_\ell),\, P_\ell\big), \tag{4}$$

where $\mathcal{L}_\ell$ is the layer's term in (1), Π_ℓ the prefix mask, and the rounding $\mathcal{Q}_\ell$ is treated as the identity in the backward pass, so the loss is always measured on the cache that will be served. Queries and targets come from the uncompressed model (teacher forcing), so an error in layer ℓ does not reach layer $\ell + 1$: the layers are independent, can be refined in parallel, and never load the model's weights, so working memory scales with the cache rather than the model. We take 50 full-batch Adam steps per layer (Kingma & Ba, 2015), though we observe that 10 often suffice (Sec. 4.1). In total, compression costs one prefill (RAPID) or two (RAPID++), one SVD per layer, the allocation solve, and one attention pass over the reference queries per step (over the *compressed* cache).

3.4 Inference from the compressed cache

The compressed cache of a layer is its quantized latents $\widehat{C}_\ell$, its projection $P_\ell = [P_K \,|\, P_V]$, and the side information (codebooks, prefix lengths, and width codes), with the scales $s_{K,\ell}, s_{V,\ell}$ folded into P_ℓ. It represents the keys and values $\hat{K}_{\ell,g} = \widehat{C}_\ell P_{K,g}$ and $\hat{V}_{\ell,g} = \widehat{C}_\ell P_{V,g}$, where $P_{K,g}, P_{V,g} \in \mathbb{R}^{R\times d}$ are the columns of KV head g. Serving a query q of head h, with $g = g(h)$, requires the scores $q\hat{K}_{\ell,g}^\top$ and, given the attention weights a, the output $a\hat{V}_{\ell,g}$. By associativity,

$$q\,\hat{K}_{\ell,g}^\top = \big(q\,P_{K,g}^\top\big)\,\widehat{C}_\ell^\top, \qquad a\,\hat{V}_{\ell,g} = \big(a\,\widehat{C}_\ell\big)\,P_{V,g}. \tag{5}$$

We therefore project the query into the latent space once and take the dot products there, each token contributing only its $r_{\ell t}$ stored coordinates; for the values, we take the attention-weighted sum of the latents and apply the projection once to the result. The keys and values are never rebuilt. This is the inference path of multi-head latent attention (MLA), in which the up-projections are absorbed into the query and the output, so a RAPID cache is served like an MLA cache whose latent is fitted to the context and whose width varies per token. At high compression ratios these latents are extremely small, a short prefix at a few bits per coordinate (Appendix G). Decoding is memory-bound, and each step reads the compressed cache, projections and side information included, in place of the full one, so the bandwidth pressure of decoding falls with the compression ratio.

4 EXPERIMENTS

We evaluate RAPID and RAPID++ on LongHealth benchmakrk (Section 4.1), on a categorized extension of it whose questions combine several facts (Section 4.2), with additional end-to-end training (Appendix J.1), and on two more benchmarks (MTOB and QASPER, Appendix J). We chose models whose attention was designed to keep the cache small: grouped-query attention (GQA), multi-head latent attention (MLA) (DeepSeek-AI, 2024), whose cache is already a low-dimensional latent, and hybrid models that interleave global layers with sliding-window local layers. Appendix C lists the models and describes how RAPID handles each design. On Gemma-4-31B, whose layers are five-sixths sliding-window, with every layer compressed, and on the two free-form benchmarks, MTOB with a grammar book of 46K or 100K tokens and QASPER (Appendix J, Table 10), RAPID++ matches the uncompressed cache within one point at $10\times$ and stays within 3.3 points at $30\times$.

4.1 LONGHEALTH BENCHMARK

LongHealth (Adams et al., 2024) asks multiple-choice questions about the clinical records of fictional patients. Following Attention Matching (Zweiger et al., 2026), we use four contexts of five patients each, about 60K tokens and 100 questions per context. Each question is asked twice with the options shuffled, and the model answers directly, without chain-of-thought. (Appendix D.1).

We compare RAPID++ and RAPID with training-free baselines at the same stored size: token eviction with SnapKV (Li et al., 2024), its head-adaptive form Ada-SnapKV (Feng et al., 2024), KVzip (Kim et al., 2025a), and Ada(KVzip), which gives KVzip's scores Ada-KV's head budgets; and RDKV (Zhang et al., 2026), which allocates eviction and precision jointly. We also compare with Attention Matching (Zweiger et al., 2026), which fits each head's keys and values to reference queries. The released caches we use were built with two inputs beyond the context: per-head budgets calibrated in advance for each model, and self-study conversations the model generates about the context. Calibrated budgets exist for Qwen3-4B and Llama-3.1-8B, so we evaluate Attention Matching on those two models. Cartridges (Eyuboglu et al., 2025) trains the cache end to end on self-study data for thousands of steps. We evaluate RAPID++ in that setting, starting end to end training with RAPID++ as an initialization, and find that the initialization carries through: caches that start from RAPID++ end far above the cartridge trained from its own initialization, and require only a small fraction of the training data (Appendix J.1).

Figure 2(a) shows every method on the four models, and Appendix E lists every result. RAPID++ stays within the band of the uncompressed cache for higher compression ratios than any baseline, while the training-free baselines lose much of the benefit of the context already at $20\times$; the gap between the allocated cache before refinement and the refined one shows how much of this the refinement contributes.

Figure 2(b) shows how fast the refinement converges. We report every trained cache after 50 refinement steps, but fewer often suffice: at the middle ratio of each model, RAPID++ is within about one point of its step-50 accuracy after 10 steps on GLM-4.7-Flash, Qwen3-4B, and Llama-3.1-8B, and within 6 points on GPT-OSS-120B. At the highest ratios, accuracy is still rising at step 50.

To separate the effect of the refinement from that of the initialization (algorithm steps 1,2), we also apply the same refinement (same settings) to the caches of other methods. See ablations in App. H.

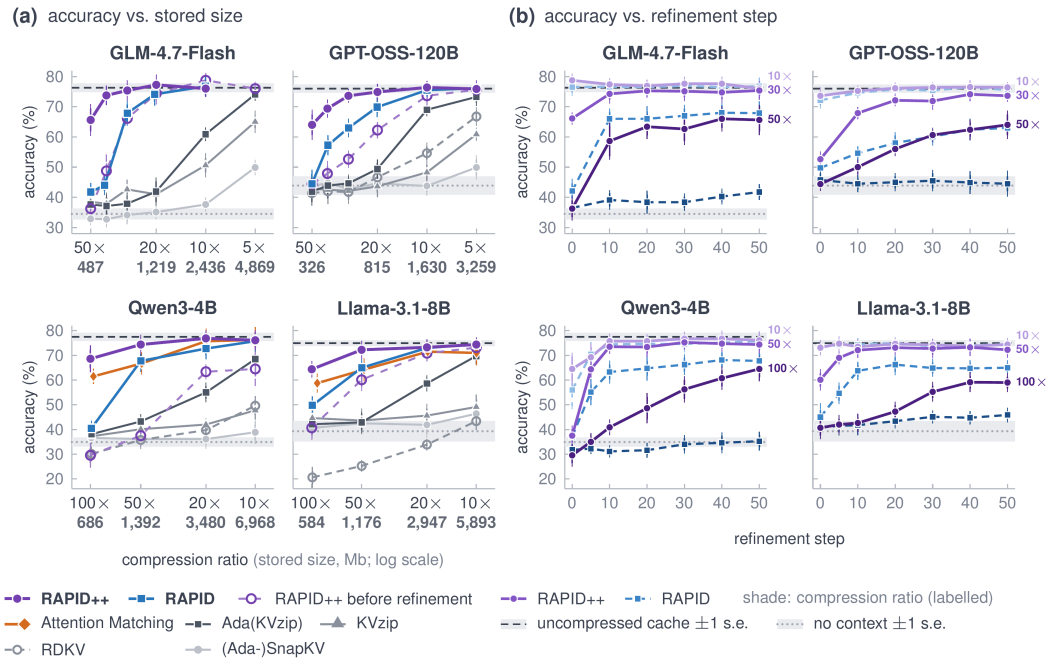


Figure 2: LongHealth accuracy (%) on four models. Lines are ± 1 standard error over the contexts. **(a)** Accuracy against compression ratio and stored size (Mb, log scale). "before refinement" is the allocated cache of RAPID++. Attention Matching was run on Qwen3-4B and Llama-3.1-8B only due limited compute. **(b)** Accuracy against refinement step at three ratios per model, darker for more compression; solid lines RAPID++, dashed RAPID;

4.2 Beyond Benchmark Scores

Every LongHealth question concerns a single patient, and most ask for one documented fact. To test which caches survive harder questions, we extend the benchmark with 400 categorized multiple-choice questions on the same four contexts (Appendix D.2): a precise detail whose distractors are values from the same record in a different role (*near miss*); a comparison across two or more patients (*cross-patient*); a link between events documented at different points of one patient's course (*timeline*); and questions that sound grounded but are not answered by the records (*unanswerable*). We report accuracy on the three answerable categories on Qwen3-4B and Llama-3.1-8B (Figure 3a); GLM-4.7-Flash and GPT-OSS-120B are in Figure 5, and every value in Table 6.

The extended questions require more than retrieving one fact: the answer may depend on several sections spread across records, which makes the attention lookups they need more complex. They separate the methods far more than the original questions do. Averaged over the models, and relative to the uncompressed cache, RAPID++ keeps 99 percent of its accuracy on both the original and the extended questions at $10\times$, and moves from 99 to 90 percent at $20\times$. At $20\times$, RAPID drops from 95 to 78 percent, Attention Matching from 97 to 75, and Ada(KVzip), KVzip, and RDKV from 67, 57, and 53 percent to 25, 21, and 28.

Attention reconstruction on test queries. To understand the difference between RAPID++ and the baselines, we measure, for every cache, the quantity RAPID minimizes, but on queries that no method fits: the relative attention-output error Eqn. 7, averaged over layers, on the tokens of each benchmark question (Figure 3b). At 20×, RAPID++ has the lowest error of every compressed cache on both models, on both benchmarks, and in every category: 5.5× lower than the best baseline on Llama-3.1-8B and 2.2× on Qwen3-4B, and 6.3× and 2.7× lower than Attention Matching. Its error on the extended questions is the same as on the original ones (0.011 and 0.012 on Llama-3.1-8B, 0.026 on Qwen3-4B), although the reference queries it was optimized on come from the context alone. Cross-patient questions have the lowest error for almost every cache; Attention Matching is an exception, with its highest error there on both models (8.5× that of RAPID++ on Llama-3.1-8B). Notably, the attention reconstruction metric can have value in practice: allow faster iterations and

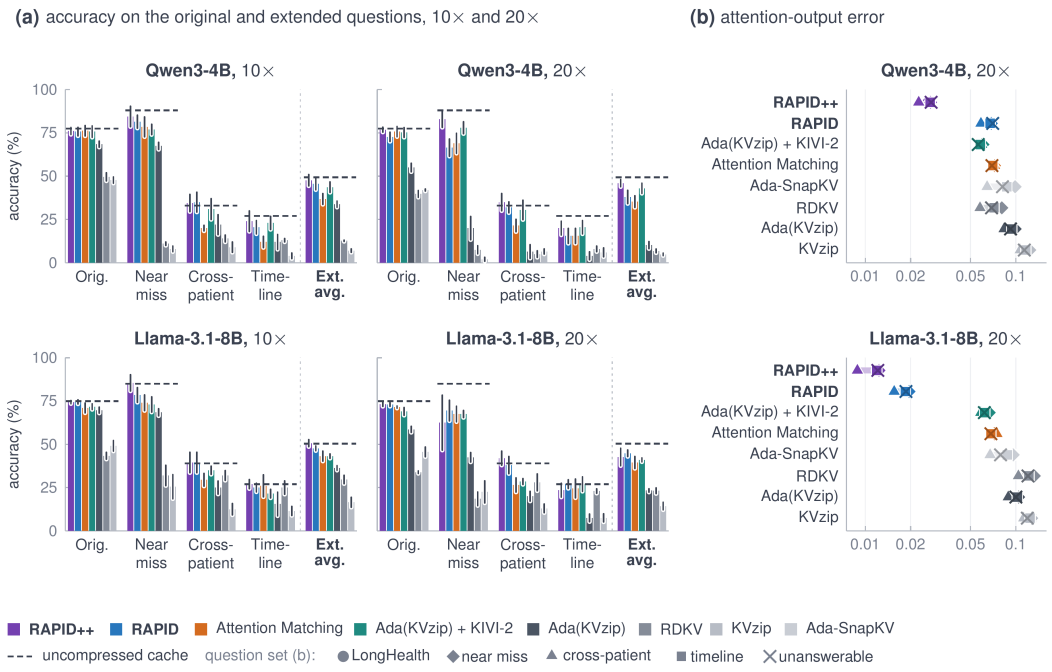


Figure 3: Harder questions on Qwen3-4B and Llama-3.1-8B. **(a)** Accuracy (%) at 10× (left) and 20× (middle) on the original LongHealth questions (Orig.) and on the extended questions. **(b)** Relative attention-output error on the benchmark question tokens at 20×, the quantity RAPID minimizes, $\|o - o^*\|^2/\|o^*\|^2$ averaged over layers. Error bars in (a) are ± 1 standard error over the four contexts; in (b) standard errors are below 7% of each value. GLM-4.7-Flash and GPT-OSS-120B are in Figure 5, and every value in Table 6.

evaluations of compressed caches, as it requires no labeled data or inference, and most importantly, it may reveal performance gaps and failure modes which binary benchmark questions conceal.

5 Limitations and Future Work

The optimization problem. Eqn. (1) is hard to optimize directly due to its discrete bit-allocation decisions (Wu et al., 2018). RAPID produces a feasible solution but not by minimizing Eqn. (1) directly. We expect that algorithms that more directly solve Eqn. (1), e.g., by developing a continuous relaxation, and apply gradient-based optimizations to it could further enhance performance. In particular, once we get to last-step refinement, the bit allocation is frozen, which may be suboptimal.
Improved quantization. The current approximation may benefit from more quantization strategies, including support for non-integer effective precision budgets. See, e.g., Padaki et al. (2026). Also, our quantization patterns are quite rigid, in order to save the space on storing the pattern itself. There could be alternative patterns: e.g., for each latent, specify the prefix p_b length of b-bit quantized coordinates (in our experiments, the coordinate bit-precision is almost decreasing in index j).

Decoding kernel and reference implementation. Supporting RAPID-compressed models in production inference systems requires adaptations to the inference stack and associated computational kernels. We do not have a full implementation of the decode kernel, though, we do not see major obstacles (especially since decode is memory-bound).

Online/streaming compaction. Long-context interactions provide a natural setting for online KV-cache compaction. Extending RAPID to support iterative compression, together with evaluating its feasibility in resource-constrained environments with limited memory and computational capacity, remains an important direction for future work.

Further compression through end-to-end adaptation. As observed in Sec. J.1, adding an additional end-to-end training stage using a relatively small dataset can improve the achievable compression rate while preserving the original model quality. Further investigation is needed to characterize and improve the extent of compression achievable through this adaptation stage.

AI USE STATEMENT

In this work, we used generative AI tools primarily for algorithm implementation, and running experiments, as well as for limited polishing of writing. AI tools were also used for finding all the (quite substantial) relevant literature.

We have not used generative AI tools for ideating, and generating the main steps of the algorithm. Nearly all fundamental ideas (e.g., using latent decomposition, with variable dimension and bit allocations, refinement) came from the authors.

We have reviewed all AI-assisted work. *Implementations:* the code was verified and tested by at least 2 persons. In order to ensure the claimed compression ratios, our code outputs the compressed KV cache in a special format file `.rapid`, which is how we measure the compression. *Writing:* while AI had limited input into the writing process, the authors rewrote or polished all AI output.

We take full responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work focuses on efficiency of now-standard parts of LLMs. We do not feel that our work raises questions regarding the Code of Ethics.

REPRODUCIBILITY STATEMENT

We release the full source code of RAPID and RAPID++, including every baseline we ran, as supplementary material. The repository contains the configuration of every experiment in the paper. The algorithm is specified in Section 3; the reference queries, the storage accounting, and every hyperparameter of the allocation and the refinement are given in Appendix B, and the models and the adaptations for each architecture in Appendix C. The benchmarks, baselines, and evaluation settings are described in Section 4, the answer protocol in Appendix D.1, and every result behind the figures is listed in Appendix E. The construction and validation of the extended LongHealth questions are described in Appendix D.2, and the questions themselves and the prompts that generated them are included with the code. All models and public benchmarks we use are openly available.

REFERENCES

Lisa Adams, Felix Busch, Tianyu Han, Jean-Baptiste Excoffier, Matthieu Ortala, Alexander Löser, Hugo J. W. L. Aerts, Jakob Nikolas Kather, Daniel Truhn, and Keno Bressem. LongHealth: A question answering benchmark with long clinical documents. *arXiv preprint arXiv:2401.14490*, 2024. URL https://arxiv.org/abs/2401.14490.

Nikita Agrawal and Ruben Mayer. Benchmarking KV-cache optimizations across task quality and system performance for long-context serving, 2026. URL https://arxiv.org/abs/2607.05399.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*, 2023. URL https://arxiv.org/abs/2305.13245.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2025. URL https://arxiv.org/abs/2501.00663.

Priyansh Bhatnagar, Ashkan Moradifirouzabadi, Se-Hyun Yang, SeungJae Lee, Jungwook Choi, and Mingu Kang. STAR-KV: Low-rank KV cache compression via soft thresholding for adaptive rank control, 2026. URL https://arxiv.org/abs/2606.08382.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.12981.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling, 2024. URL https://arxiv.org/abs/2406.02069.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. Palu: KV-cache compression with low-rank projection. In *International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2407.21118.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, and Mohamed S. Abdelfattah. xKV: Cross-layer KV-cache compression via aligned singular vector extraction. *arXiv preprint arXiv:2503.18893*, 2025b. URL https://arxiv.org/abs/2503.18893. Revised May 2026.

Justin Y. Chen, Ying Feng, Piotr Indyk, Michael Kapralov, Ekaterina Kochetkova, and Boris Prokhorov. Towards tight bounds for streaming attention, 2026. URL https://arxiv.org/abs/2606.07205.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Léon Bottou, Zhihao Jia, and Beidi Chen. MagicPIG: LSH sampling for efficient LLM generation. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.16179.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of EMNLP*, 2023. URL https://arxiv.org/abs/2305.14788.

Ofek I. Cohen. Query-oblivious coresets for softmax attention: Improved bounds and efficient constructions, 2026. URL https://arxiv.org/abs/2609.06327.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2nd edition, 2006.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of NAACL-HLT*, 2021. URL https://arxiv.org/abs/2105.03011.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: KV cache compression by estimating attention from future queries distribution, 2025. URL https://arxiv.org/abs/2510.00636.

Maurizio Diaz. Learned structure in Cartridges: Keys as shareable routers in self-studied representations, 2025. URL https://arxiv.org/abs/2508.17032.

Shahrzad Esmat et al. VarRate: Training-free variable-rate KV cache compression for long-context LLMs, 2026. URL https://arxiv.org/abs/2607.15498.

Hugh Everett. Generalized Lagrange multiplier method for solving problems of optimum allocation of resources. *Operations Research*, 11(3):399–417, 1963.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025. URL https://arxiv.org/abs/2506.06266.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference, 2024. URL https://arxiv.org/abs/2407.11550. NeurIPS 2025.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*, 2024a. URL https://arxiv.org/abs/2310.01801.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations*, 2024b. URL https://arxiv.org/abs/2307.06945.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Sihyeon Ha, Jaeho Lee, and Yo-Seb Jeon. KV-COBRA: KV cache compression via co-optimized bit-rank allocation, 2026. URL https://arxiv.org/abs/2609.24298.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization, 2024. URL https://arxiv.org/abs/2401.18079.

Bo Jiang and Sian Jin. KVSculpt: KV cache compression as distillation, 2026. URL https://arxiv.org/abs/2603.27819.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of EMNLP*, 2023. URL https://arxiv.org/abs/2310.05736.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. RestoreKV: Recovering full-cache behavior under aggressive query-agnostic KV cache eviction.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. GEAR: An efficient KV cache compression recipe for near-lossless generative inference of LLM, 2024. URL https://arxiv.org/abs/2403.05527.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction, 2025a. URL https://arxiv.org/abs/2505.23416.

Junhyuck Kim, Jongho Park, Jaewoong Cho, and Dimitris Papailiopoulos. Lexico: Extreme KV cache compression via sparse coding over universal dictionaries. In *International Conference on Machine Learning*, 2025b. URL https://arxiv.org/abs/2412.08890.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Rahul Krishnan and Volker Schulz. A JoLT for the KV cache: Near-lossless KV cache compression via joint Lagrangian allocation of Tucker ranks and a rotated residual for LLMs, 2026. URL https://arxiv.org/abs/2607.12550.

Yuri Kuratov, Matvey Kairov, Aydar Bulatov, Ivan Rodkin, and Mikhail Burtsev. GradMem: Learning to write context into memory with test-time gradient descent. *arXiv preprint arXiv:2603.13875*, 2026. URL https://arxiv.org/abs/2603.13875.

Ang Li, Sean McLeish, Haozhe Chen, Nimit Kalra, Zaiqian Chen, Artem Gazizov, Venkata Anoop Suhas Kumar Morisetty, Bhavya Kailkhura, Harshitha Menon, Zhuang Liu, Brian R. Bartoldson, Tom Goldstein, Sanae Lotfi, Micah Goldblum, and Pavel Izmailov. End-to-end context compression at scale, 2026. URL https://arxiv.org/abs/2606.09659.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL*, 2021. URL https://arxiv.org/abs/2101.00190.

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. In *International Conference on Learning Representations*, 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.

Edo Liberty, Alexandr Andoni, and Eldar Kleiner. Nearly optimal attention coresets, 2026. URL https://arxiv.org/abs/2605.05602.

Bokai Lin, Zihao Zeng, Zipeng Xiao, Siqi Kou, Tianqi Hou, Xiaofeng Gao, Hao Zhang, and Zhijie Deng. MatryoshkaKV: Adaptive KV compression via trainable orthogonal projection, 2024. URL https://arxiv.org/abs/2410.14731.

Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. RetrievalAttention: Accelerating long-context LLM inference via vector retrieval, 2024a. URL https://arxiv.org/abs/2409.10516.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.17118.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache, 2024b. URL https://arxiv.org/abs/2402.02750.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982.

Lu et al. One size does not fit all: Token-wise adaptive compression for KV cache, 2026. URL https://arxiv.org/abs/2603.04411.

Daming Luo, Christy Liang, and Junyu Xuan. How query visibility changes KV-cache compression rankings: A matched-budget audit, 2026. URL https://arxiv.org/abs/2607.11942.

Joel Max. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6(1):7–12, 1960.

Ruijie Miao, Zhiming Wang, Wang Li, Shiwei Wu, Shufan Liu, Yanbing Jiang, and Tong Yang. Beyond token eviction: Mixed-dimension budget allocation for efficient KV cache compression, 2026. URL https://arxiv.org/abs/2603.20616.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2304.08467.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized KV cache compaction in a single forward pass, 2026. URL https://arxiv.org/abs/2606.07878.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

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

Antonio Ortega and Kannan Ramchandran. Rate-distortion methods for image and video compression. *IEEE Signal Processing Magazine*, 15(6):23–50, 1998.

Ashwin Padaki, Amir Ingber, and Edo Liberty. Vq-bench: A composable vector quantization framework, 2026. URL https://arxiv.org/abs/2608.11240.

Junyoung Park, Dalton Jones, Matthew J. Morse, Raghavv Goel, Mingu Lee, and Chris Lott. KeyDiff: Key similarity-based KV cache eviction for long-context LLM inference in resource-constrained environments, 2025. URL https://arxiv.org/abs/2504.15364. NeurIPS 2025.

Junjie Peng et al. GRKV: Global regression for training-free KV cache compression in long-context LLMs, 2026. URL https://arxiv.org/abs/2605.31105.

Maja Popović. chrF: Character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pp. 392–395, 2015.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for KV cache compression, 2024. URL https://arxiv.org/abs/2408.05646.

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150.

Yair Shoham and Allen Gersho. Efficient bit allocation for an arbitrary set of quantizers. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(9):1445–1453, 1988.

Prabhakant Sinha and Andris A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context, 2022. URL https://arxiv.org/abs/2209.15189.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference, 2024a. URL https://arxiv.org/abs/2410.21465.

Libo Sun, Peixiong He, Po-Wei Harn, and Xiao Qin. MoE-nD: Per-layer mixture-of-experts routing for multi-axis KV cache compression. *arXiv preprint arXiv:2604.17695*, 2026. URL https://arxiv.org/abs/2604.17695.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems*, 2024b. URL https://arxiv.org/abs/2405.05254.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2406.10774.

Garrett Tanzer, Mirac Suzgun, Eline Visser, Dan Jurafsky, and Luke Melas-Kyriazi. A benchmark for learning to translate a new language from one grammar book. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.16575.

TODO: pull from arXiv. KV cache transform coding for compact storage in LLM inference, 2025. URL https://arxiv.org/abs/2511.01815.

TODO: pull from arXiv. eOptShrinkQ: Near-lossless KV cache compression through optimal spectral denoising and quantization, 2026a. URL https://arxiv.org/abs/2605.02905.

TODO: pull from arXiv. S^4R: Selective sampling, subspaces, and sparse reconstruction for compressed long-context KV caching, 2026b. URL https://arxiv.org/abs/2608.00528.

Zhongwei Wan, Xinjian Wu, Yu Zhang, Yi Xin, Chaofan Tao, Zhihong Zhu, Xin Wang, Siqi Luo, Jing Xiong, and Mi Zhang. D2O: Dynamic discriminative operations for efficient long-context inference of large language models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.13035.

Heng Wang, Jielin Qiu, Wenting Zhao, Cheng Qian, Liangwei Yang, Jiawei Han, Heng Ji, Silvio Savarese, Shelby Heinecke, and Huan Wang. Random attention: Rethinking KV cache eviction for efficient reasoning. *arXiv preprint arXiv:2609.03430*, 2026. URL https://arxiv.org/abs/2609.03430.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. Model tells you where to merge: Adaptive KV cache merging for LLMs on long-context tasks, 2024. URL https://arxiv.org/abs/2407.08454.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP: System Demonstrations*, pp. 38–45, 2020.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.17453.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.10819.

June Yong Yang, Byeongwook Kim, Jeongin Bae, Beomseok Kwon, Gunho Park, Eunho Yang, Se Jung Kwon, and Dongsoo Lee. No token left behind: Reliable KV cache compression via importance-aware mixed precision quantization, 2024. URL https://arxiv.org/abs/2402.18096.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate, 2025. URL https://arxiv.org/abs/2504.19874.

Junkai Zhang, Hang Guo, Luca Benini, and Yawei Li. RDKV: Rate-distortion bit allocation for joint eviction and quantization of the KV cache, 2026. URL https://arxiv.org/abs/2605.08317.

Rongzhi Zhang, Kuang Wang, Liyuan Liu, Shuohang Wang, Hao Cheng, Chao Zhang, and Yelong Shen. LoRC: Low-rank compression for LLMs KV cache with a progressive compression strategy, 2024a. URL https://arxiv.org/abs/2410.03111.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. CaM: Cache merging for memory-efficient LLMs inference. In *International Conference on Machine Learning*, 2024b.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Yuxuan Zhu, David H. Yang, Mohammad Mohammadi Amiri, Keerthiram Murugesan, Tejaswini Pedapati, and Pin-Yu Chen. OjaKV: Context-aware online low-rank KV cache compression. In *Findings of ACL*, 2026. URL https://arxiv.org/abs/2509.21623.

Fei Zuo, Zikang Zhou, Hao Cong, Xiaoyan Xi, and Ho Fai Leung. RateQuant: Optimal mixed-precision KV cache quantization via rate-distortion theory, 2026. URL https://arxiv.org/abs/2605.06675.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via attention matching, 2026. URL https://arxiv.org/abs/2602.16284.

The appendix follows the order in which the main text relies on it: further related work (Appendix A), the algorithm (Appendix B), the models (Appendix C), the benchmarks and the answer protocol (Appendix D), and every result behind the figures (Appendix E); then the evidence for the design choices (Appendices F and G), our components added to the baselines (Appendix H), end-to-end training (Appendix I).

A Additional related work

Compressing the context rather than the cache. Text-space methods shorten the prompt itself, for example by dropping low-information tokens (Jiang et al., 2023). Soft-prompt methods train the model to write the context into a few continuous vectors: gist tokens (Mu et al., 2023), AutoCompressors (Chevalier et al., 2023), and the in-context autoencoder (Ge et al., 2024b). Both descend from prefix tuning (Li & Liang, 2021), which optimizes a short KV prefix per task, and from context distillation (Snell et al., 2022), which trains the model to behave as if the context were present; Cartridges (Eyuboglu et al., 2025) combines the two per context. All of these train the model, or train per model. RAPID operates on the cache the frozen model produced and needs only each layer's attention projections.

Architectures with smaller caches. Multi-query (Shazeer, 2019) and grouped-query attention (Ainslie et al., 2023) share KV heads across query heads; multi-head latent attention (DeepSeek-AI, 2024) caches one low-dimensional latent per token from which every head's keys and values are up-projected; cross-layer attention (Brandon et al., 2024) and YOCO (Sun et al., 2024b) share the cache across layers; hybrid models interleave global layers with sliding-window layers (Gemma Team, 2026; OpenAI, 2025). These choices are made at training time. Every model in Table 1 uses at least one of them, so the ratios we report are on top of the architecture's own savings. MLA is the template for our per-layer latent (Section 3.1), applied after the fact to a realized cache, with a rank chosen per token.

Eviction and merging. StreamingLLM (Xiao et al., 2024) keeps a window plus the first few tokens; Scissorhands (Liu et al., 2023) and H_2O (Zhang et al., 2023) keep tokens that have received the most attention; FastGen (Ge et al., 2024a) chooses a policy per head from its attention profile; TOVA (Oren et al., 2024) evicts by the attention of the most recent token; SnapKV (Li et al., 2024) scores tokens from an observation window at the end of the prompt; PyramidKV (Cai et al., 2024) shrinks the budget with depth and Ada-KV (Feng et al., 2024) redistributes it across heads; KeyDiff (Park et al., 2025), KVzip (Kim et al., 2025a), and Expected Attention (Devoto et al., 2025) score without the question, respectively by key dissimilarity, by the attention received while the model reconstructs the context, and by the expected attention under a Gaussian model of future queries; DuoAttention (Xiao et al., 2025) keeps full caches only for retrieval heads; Random Attention (Wang et al., 2026) evicts generated reasoning tokens rather than the prompt. Merging methods fold evicted pairs into retained ones instead of dropping them (Zhang et al., 2024b; Wan et al., 2025; Wang et al., 2024); GRKV (Peng et al., 2026) does so by a ridge regression that minimizes the attention-output discrepancy between the compressed and the full cache, the closed-form relative of our objective among merging methods. The audit of Luo et al. (2026) shows that scorers which see the question at compression time lose most of their advantage when it is withheld, the setting that cache reuse requires.

Precision. KIVI (Liu et al., 2024b) quantizes keys per channel and values per token at 2–4 bits; KVQuant (Hooper et al., 2024) adds non-uniform codebooks and outlier handling; GEAR (Kang et al., 2024) adds a low-rank residual and a sparse outlier term; MiKV (Yang et al., 2024) stores low-importance tokens at low precision instead of evicting them; TurboQuant (Zandieh et al., 2025) rotates before scalar quantization and RateQuant (Zuo et al., 2026) allocates bit widths across heads by reverse water-filling. Lexico (Kim et al., 2025b) replaces scalar quantization by sparse coding over a universal dictionary. All keep every token and every coordinate; RAPID uses scalar Lloyd–Max codebooks as one axis of a budget in which precision competes with rank and with the token itself.

Low rank. One family fixes the basis per model: Eigen Attention (Saxena et al., 2024), LoRC (Zhang et al., 2024a), and Palu (Chang et al., 2025a) derive it from the weights or a calibration set; MatryoshkaKV (Lin et al., 2024) trains orthogonal projections with nested ranks; STARKV (Bhatnagar et al., 2026) learns per-head ranks by soft thresholding. Another computes it per con-

text, as we do: xKV (Chang et al., 2025b) takes one SVD across a group of layers; ShadowKV (Sun et al., 2024a) keeps low-rank pre-RoPE keys on the GPU and offloads the values; OjaKV (Zhu et al., 2026) updates the basis online by Oja's rule during prefill and decoding; S^{4}R (TODO: pull from arXiv, 2026b) takes a truncated SVD over sampled tokens and reconstructs only the tokens a query needs; eOptShrinkQ (TODO: pull from arXiv, 2026a) chooses the rank by optimal singular-value shrinkage and quantizes the residual. In all of these the rank is set per layer or head, not per token, and the projected cache is served as it is. The methods that do vary the rank per token, MixedDimKV, VarRate, and DynaKV (Miao et al., 2026; Esmat et al., 2026; Lu et al., 2026), are discussed in Section 2. RAPID differs from both families in the same way: a rank per token and a precision per coordinate under one budget, allocated across layers by measured sensitivity, and a refinement of the stored values against the attention output.

Sparse and retrieval attention. Quest (Tang et al., 2024), RetrievalAttention (Liu et al., 2024a), and MagicPIG (Chen et al., 2025) keep the full cache and read only a query-dependent part of it at each decode step, through page metadata, a vector index, or LSH sampling. They cut the bytes read per step, not the bytes stored, and are complementary: a compressed latent can be indexed and read sparsely in the same way.

Test-time memory. GradMem (Kuratov et al., 2026) and Titans (Behrouz et al., 2025) write context into a memory by gradient descent at test time, with the memory and its update rule meta-trained; Cartridges is per-context test-time training of a KV prefix. RAPID's refinement is also test-time gradient descent, but on a layer-local objective, with the model frozen and never loaded.

Bit allocation. Trading a hard budget for a price per bit is the generalized Lagrange multiplier method of Everett (1963); its use for allocating bits among quantizers is due to Shoham & Gersho (1988), surveyed by Ortega & Ramchandran (1998), and reverse water-filling is its closed form for Gaussian sources (Cover & Thomas, 2006, Thm. 10.3.3). KVTC (TODO: pull from arXiv, 2025) applies the same transform-coding pipeline, a PCA basis, per-dimension bit allocation, and entropy coding, to caches in storage, decompressing them before use. Problem (13) is a multiple-choice knapsack (Sinha & Zoltners, 1979) with two structural constraints, a prefix per token and one width per coordinate. RDKV, RateQuant, KV-COBRA, and JoLT use the same tools with a rank or bit width per head or layer group, and MixedDimKV with a dimension per token at a fixed width; in each of them the relaxed problem separates, and for MixedDimKV a Shapley–Folkman argument bounds the duality gap by a constant. Ours does not separate, because the width of a coordinate is shared by every token that keeps it, which is why the relaxed problem is solved by alternation (Appendix B.3).

Encoder–decoder and amortized compression. A separate line trains the model to read a shorter sequence of continuous tokens in place of the context (Ge et al., 2024b; Chevalier et al., 2023; Mu et al., 2023). LCLM (Li et al., 2026) scales this up: a 0.6B encoder maps each block of $N \in \{4, 8, 16\}$ tokens to one latent that a 4B decoder consumes, with encoder and decoder continually pre-trained together on about 350B tokens. The decoder prefills $N\times$ fewer positions, so compression is faster than a full prefill and the result is an ordinary KV cache that standard engines serve. In exchange, the ratio is fixed at training time, every base model and ratio needs its own training run, and the decoder is no longer the original model. At $16\times$ on Qwen3-4B, LCLM loses 8 points on LongHealth with mean pooling (concat pooling matches on LongHealth but not on RULER or LongBench) and 19–28 points on RULER, against the uncompressed Qwen3-4B-Instruct it starts from. Two amortized compressors work on the cache itself rather than on the tokens. Still (O'Neill et al., 2026) trains a small Perceiver per layer, once per model against the frozen base model, that reads the full cache and emits compact keys and values in one forward pass, at $8\times$–$200\times$, and can be applied again as the context grows; RestoreKV (Jiang et al.) keeps query-agnostic eviction and adds a small restore cache, generated in one pass by LoRA adapters trained once by self-distillation. All of these amortize a training run per model over every later context. RAPID is the complementary point in this space: no training, any frozen model, any ratio chosen per context, and within 2–3 points of the uncompressed cache at $20\times$–$30\times$, at the price of a per-context optimization that a feed-forward compressor does not pay. The two compose: RAPID compresses the cache of an LCLM decoder, or the compact cache Still emits, like any other.

Benchmarks and protocols. SCBench (Li et al., 2025) evaluates long-context methods across the KV-cache lifecycle, with shared contexts reused over multi-turn and multi-request workloads, and reports that sub-linear-memory methods degrade once the cache is reused; Agrawal & Mayer

(2026) measure eviction, quantization, and merging methods jointly on task quality and serving throughput and find that the compression ratio alone predicts neither. Together with the query-visibility audit above, these studies motivate our protocol: one compressed cache per context, all questions answered from it, budgets matched in bytes including side information, and paired per-question tests against every baseline.

B ALGORITHM DETAILS

This appendix gives the full description of the three steps of Section 3, with every hyperparameter.

B.1 NOTATION, OBJECTIVE, AND REFERENCE QUERIES

Each attention layer has H_Q query heads and H_{KV} key–value (KV) heads of dimension d; in grouped-query attention (GQA) (Ainslie et al., 2023) $H_{KV} < H_Q$ and query head h reads KV head $g(h)$. Keys are stored after positional encoding. For a query $q_{\ell ih}$ of head h at position i, the layer computes

$$a_{\ell ih} = \mathrm{softmax}\Big(\tfrac{1}{\sqrt{d}}\, q_{\ell ih} K_{\ell,g(h)}^{\top} + m_i\Big), \qquad o_{\ell i} = \mathrm{Concat}_{h=1}^{H_Q}\big(a_{\ell ih} V_{\ell,g(h)}\big)\, W_{O,\ell}, \tag{6}$$

where $K_{\ell,g}$ is the block of KV head g and m_i is the causal mask; $a_{\ell iht}$ is the weight query i of head h places on cached token t. Only the cached rows are approximated; everything else in (6) stays frozen.

Per-layer objective. For layer ℓ, $\mathcal{I}_\ell$ indexes the reference queries, $o_{\ell i}^{\mathrm{tgt}}$ is (6) on the exact cache, and $\hat{o}_{\ell i}$ is the same computation on the compressed cache. Both are taken after $W_{O,\ell}$, where attention enters the residual stream, and both use the same query, formed from the layer input that the uncompressed model produced (*teacher forcing*). The layer's term in (1) is the relative error

$$\mathcal{L}_\ell = \frac{\sum_{i\in\mathcal{I}_\ell} \|\hat{o}_{\ell i} - o_{\ell i}^{\mathrm{tgt}}\|_2^2}{\sum_{i\in\mathcal{I}_\ell} \|o_{\ell i}^{\mathrm{tgt}}\|_2^2}, \qquad \mathcal{L} = \tfrac{1}{L}\textstyle\sum_\ell \mathcal{L}_\ell. \tag{7}$$

Under teacher forcing a compression error in layer ℓ does not change the queries of layer $\ell+1$, so each $\mathcal{L}_\ell$ can be measured, and optimized, on its own.

Reference queries. During prefill we record, at every layer and position, the layer input and the attention output, and rebuild the query from the input with the layer's own weights when it is needed. These prefill queries are the reference set of RAPID: every position of the context, reading the tokens before it, as in H2O (Zhang et al., 2023). RAPID++ adds repeat-context queries, introduced by KVzip (Kim et al., 2025a) and used as reference queries by Attention Matching (Zweiger et al., 2026): the context is fed once more after the exact cache, so that every query reads all T tokens. Unlike KVzip, we feed the raw context without a repeat instruction, in blocks of 2,048 tokens. The allocation and the refinement use the same reference set, so the two variants differ only in it.

B.2 STEP 1: REPRESENTATION

Shared layer basis. The keys and values of layer ℓ, scaled to equal Frobenius norm and concatenated, form $X_\ell \in \mathbb{R}^{T\times D}$, one row per token. We split its SVD $X_\ell = U_\ell \Sigma_\ell W_\ell^{\top}$ evenly between two factors,

$$X_\ell = \big[K_\ell/s_{K,\ell} \;\big|\; V_\ell/s_{V,\ell}\big] = C_\ell P_\ell, \qquad C_\ell = U_\ell \Sigma_\ell^{1/2} \in \mathbb{R}^{T\times R},\; P_\ell = \Sigma_\ell^{1/2} W_\ell^{\top} \in \mathbb{R}^{R\times D}, \tag{8}$$

of rank $R = \min\{T, D\}$, with coordinates ordered by $\sigma_{\ell j} = \Sigma_\ell[j,j]$. Row t of C_ℓ represents token t, and P_ℓ is shared by the layer and rebuilds all H_{KV} heads. Since $\|P_\ell[j,:]\|_2^2 = \sigma_{\ell j}$, dropping coordinate j of token t removes $\sigma_{\ell j} C_\ell[t,j]^2 = \sigma_{\ell j}^2 U_\ell[t,j]^2$ from $\|X_\ell\|_F^2$. Appendix C adapts the basis to GLM-4.7-Flash, whose cache is already a latent.

Layout. Token t keeps the prefix of its first $r_{\ell t}$ coordinates, and coordinate j is quantized to $b_{\ell j}$ bits by a codebook $\mathcal{Q}_{\ell j}$ of $2^{b_{\ell j}}$ centers, shared by all tokens that keep it:

$$\widehat{C}_\ell[t,j] = \begin{cases} \mathcal{Q}_{\ell j}\big(C_\ell[t,j]\big), & j \le r_{\ell t}, \\ 0, & j > r_{\ell t}, \end{cases} \qquad \text{hence} \qquad \beta_{\ell tj} = b_{\ell j}\cdot \mathbf{1}\{j \le r_{\ell t}\}. \tag{9}$$

A token with $r_{\ell t} = 0$ would be evicted; instead every token keeps at least 8 coordinates, because a coordinate that is not stored receives no gradient in Step 3 (Appendix F removes this floor). Prefixes keep the side information small: naming $r_{\ell t}$ arbitrary coordinates out of R takes $\log \binom{R}{r_{\ell t}} \approx r_{\ell t} \log R$ bits, a prefix length $\log R$.

Storage accounting. The models store their cache in bf16, so the uncompressed cache of a context occupies $B_{\text{full}} = 16\, D\, T\, L$ bits when every layer is compressed. A compressed cache stores $\beta_{\ell t j}$ bits for coordinate j of token t in layer ℓ, plus the side information B_{side}: the projections P_ℓ in bf16, the codebooks in fp16, one 16-bit prefix length per token, and a 4-bit width code per coordinate. A compression ratio of ρ means a budget $B = B_{\text{full}}/\rho$ for everything that is served, relative to the cache the model stores natively (see Appendix C for caches that are not a plain $[K \mid V]$). Our code writes every compressed cache to a `.rapid` file, and the ratios we report are measured on that file.

B.3 Step 2: bit allocation

Cost model. The cost $c_{\ell t j}(b)$ approximates the increase in (1) caused by storing coordinate (ℓ, t, j) at b bits ($b = 0$: dropped). It has three parts.

1. *Token importance.* On the exact cache,
$$w_{\ell t} \;\propto\; \max_{i,h} a_{\ell i h t}, \tag{10}$$
the largest weight any reference query places on the token, scaled to mean one over the compressed tokens of the layer.
2. *Quantization error.* For every coordinate j and width b in the menu $\mathcal{B} = \{0, 2, 3, 4, 5, 6, 8, 12, 16\}$, coded in 4 bits, a Lloyd–Max codebook (Lloyd, 1982; Max, 1960) of 2^b centers is fitted to the column $C_\ell[:, j]$, weighted by $w_{\ell t}$; $e_{\ell j}(b)$ is its weighted mean squared error. 16-bit coordinates are stored raw, without a codebook.
3. *Coordinate importance.* $g_{\ell j}$ is the rate at which latent energy in coordinate j becomes attention-output error. We use five probe ranks $r_0 < r_1 < \cdots < r_5 = R$, spread evenly from the prefix floor to the offered width (8, 256, 504, 752, and 1,000 on GPT-OSS-120B) to cut the coordinates into bands. At each probe we truncate every token of the layer to r_k coordinates, keep every other layer exact, and measure $\mathcal{L}_\ell^{(r_k)}$ (7). The drop in loss across a band, divided by the weighted energy the band holds, is
$$g_{\ell j} = \frac{\mathcal{L}_\ell^{(r_{k-1})} - \mathcal{L}_\ell^{(r_k)}}{\sum_{j' \in [r_{k-1}, r_k)} \sigma_{\ell j'} \sum_t w_{\ell t}\, C_\ell[t, j']^2}, \qquad j \in [r_{k-1}, r_k), \tag{11}$$
with the coordinates below r_0 joining the first band. Each probe costs one attention pass over the layer's reference queries, as much as one refinement step, so probing every rank would cost hundreds of passes per layer. Hence we use five probes to resolve the shape of the rate within a layer (see ablation against one probe in Appendix F).

Combined,
$$c_{\ell t j}(b) = g_{\ell j}\; w_{\ell t}\; \nu_{\ell j} \cdot \begin{cases} C_\ell[t, j]^2, & b = 0, \\ e_{\ell j}(b), & b \in \mathcal{B} \setminus \{0\}. \end{cases} \tag{12}$$
The cost sees the softmax and $W_{O,\ell}$ only through $g_{\ell j}$, which measures them band by band; Appendix G shows what the probes measure and the allocation they produce.

The allocation problem and its solution. The allocation minimizes the total cost within the budget,
$$\min_{\mathbf{r}, \mathbf{b}} \sum_{\ell, t, j} c_{\ell t j}\big(\beta_{\ell t j}\big) \quad \text{s.t.} \quad \sum_{\ell, t} \sum_{j \le r_{\ell t}} b_{\ell j} + B_{\text{side}}(\mathbf{r}, \mathbf{b}) \le B, \tag{13}$$
with $\beta_{\ell t j}$ given by (9). For a price $\lambda \ge 0$ per stored bit we drop the constraint and minimize
$$\sum_{\ell, t, j} c_{\ell t j}\big(\beta_{\ell t j}\big) \;+\; \lambda\Big(\sum_{\ell, t, j} \beta_{\ell t j} + B_{\text{side}}\Big), \tag{14}$$
which splits into one problem per layer. Within a layer it does not separate, since $b_{\ell j}$ is shared by every token that keeps coordinate j, so we alternate two exact block updates:

- *Token prefix.* With the widths fixed, token t chooses the prefix $r_{\ell t}$ maximizing $\sum_{j \le r} \big[c_{\ell t j}(0) - c_{\ell t j}(b_{\ell j}) - \lambda\, b_{\ell j}\big]$, one cumulative sum per token.
- *Bits per coordinate.* With the prefixes fixed, coordinate j takes the width in $\mathcal{B}$ minimizing the summed cost of its keepers plus λ times their bits and its codebook; it enters the layer's basis only if its keepers' gain exceeds λ times the bits of its projection row.

Each price runs four alternations (one loses 4 points, and sixteen return the same allocation; Appendix F), and 80 bisection steps find the smallest λ whose allocation fits in B. An exact minimizer of (14) would be optimal among all allocations of its size (Everett, 1963), the standard guarantee of Lagrangian bit allocation (Shoham & Gersho, 1988). The alternation instead returns a point at which no token can improve its prefix and no coordinate its width at price λ: every stored bit, in any layer, saves at least λ of surrogate error and every unstored bit at most λ, the equal-slope condition that reverse water-filling instantiates for Gaussian sources (Cover & Thomas, 2006, Thm. 10.3.3).

Refitting the projection. The SVD projection P_ℓ is optimal for the unquantized latents, but the cache serves $\widehat{C}_\ell$: the latents truncated to the solved prefixes and rounded to the solved codebooks. With $\widehat{C}_\ell$ fixed, the projection that best rebuilds the layer's scaled rows X_ℓ (8) is a ridge least-squares problem with a closed form,

$$P_\ell^\star = \big(\widehat{C}_\ell^\top \widehat{C}_\ell + \mu_\ell I\big)^{-1} \widehat{C}_\ell^\top X_\ell, \qquad \mu_\ell = 10^{-6} \cdot \tfrac{1}{R}\,\mathrm{tr}\big(\widehat{C}_\ell^\top \widehat{C}_\ell\big). \tag{15}$$

We solve it once per layer, with every token weighted equally, and keep $P_\ell^\star$ only if it lowers $\|X_\ell - \widehat{C}_\ell P\|_F^2$. It changes the values of the projection only: the prefixes, widths, codebooks, and stored size are unchanged, and a second solve on the same latents returns the same projection. On Qwen3-4B and Llama-3.1-8B it lowers the reconstruction error by 0.2–2.2 percent, more at low ratios. The refitted cache is the *before refinement* point in our figures and tables, and the initialization of Step 3.

B.4 Step 3: refinement

In (4), Π_ℓ is the solved prefix mask and $\mathcal{Q}_\ell$ rounds every kept coordinate to its solved codebook. The rounding has no useful gradient, so the backward pass treats it as the identity (a straight-through estimator), while the forward pass reads the quantized values, so the loss is measured on the cache that will be served. Only the latents are quantized; P_ℓ is trained in full precision and stored in bf16. Each layer's subproblem holds only its latents, projection, and reference queries.

The update. Every step accumulates the gradient over all reference blocks of 2,048 positions before one Adam update. The step is relative: tensor p receives the learning rate $\eta_p = \eta\,\mathrm{RMS}(p)/\operatorname{median}_{p'}\mathrm{RMS}(p')$, so the latents and the projection move by the same fraction of their size. We use $\eta = 3 \times 10^{-3}$ (10^{-3} for Gemma-4-31B) and 50 steps (100 at $100\times$) on the contexts of this paper, about 60K tokens.

C Models and architecture adaptations

Table 1 lists the models and their caches. RAPID needs only the rows a layer caches and the attention that reads them; three designs depart from a plain $[K \mid V]$ cache and need small adaptations.

Model	Params	Attention	Local window	Layers	L	H_Q	H_{KV}	D
Llama-3.2-3B	3.2B	GQA	–	28	28	24	8	2048
Qwen3-4B	4.0B	GQA	–	36	36	32	8	2048
Llama-3.1-8B	8.0B	GQA	–	32	32	32	8	2048
GLM-4.7-Flash	30B (3B active)	MLA	–	47	47	20	–	576
Gemma-4-31B	31B	GQA, local 5:1 global	1,024, compressed	60	60	32	16 / 4	8192 / 2048
GPT-OSS-120B	117B (5.1B active)	GQA, local 1:1 global	128, exact	36	18	64	8	1024

Table 1: The models, in increasing size. Attention: grouped-query (GQA) or multi-head latent (MLA); for the hybrid models, the ratio of local (sliding-window) to global layers and the window length in tokens. L is the number of layers we compress, H_Q and H_{KV} the query and key–value heads, and D the row width: the number of bf16 values one token occupies in one layer's uncompressed cache, $D = 2H_{KV}d$ for d the head dimension. Gemma-4-31B lists local / global layers.

Multi-head latent attention (GLM-4.7-Flash). GLM-4.7-Flash does not cache keys and values. It caches one 576-dimensional row per token and layer: a 512-dimensional latent, from which the model's up-projection produces every head's keys and values, and a 64-dimensional key that carries RoPE and is shared by all heads. We compress this row, and B_{full} counts it, not the keys and values it expands to. Its two parts have different statistics (the latent is RMS-normalized), so instead of one SVD of the row we take one SVD of each part and sort the union of their coordinates by singular value into a single latent. Each coordinate then belongs to one part only, and the prefix rule and the allocation apply unchanged. On the layers we measured, this halves the attention error of a single joint SVD at the same width (0.051 against 0.099 at width 544). At inference, the model's up-projection is multiplied into P_ℓ once, and (5) applies with the product in place of P_ℓ.

Local layers. GPT-OSS-120B alternates global layers with local layers that attend only to the last 128 tokens. These windows are 0.2 percent of its cache on our contexts, so we keep them exact and count them, $16 \cdot 128 \cdot D$ bits per layer, in both B_{full} and the compressed size. Gemma-4-31B interleaves local and global layers at 5:1: 50 of its 60 layers attend a window of 1,024 tokens, with 16 KV heads of dimension 256, and 10 attend the whole context, with 4 KV heads of dimension 512. On a 58K-token LongHealth context the windows hold about a quarter of the cache, so compressing only the global layers at $30\times$ would shrink the whole cache by $3.5\times$, and removing the global cache entirely by $3.8\times$. We therefore compress both kinds of layer and split the allocation into two problems, one budget for the local layers and one for the global layers, each at the target ratio, so that every layer competes for bits only with layers of its own type. The refinement runs unchanged.

Keys and values from one projection (Gemma-4-31B). In Gemma-4-31B's global layers, keys and values come from the same projection of the token. The keys then receive a learned per-channel normalization and RoPE, and the values a normalization without a scale and no rotation. The reference implementation caches both (Wolf et al., 2020), so a global layer stores two vectors per head where one projection produced both. They are no longer equal but remain strongly related, which suits the shared basis. We compress the two stored vectors as they are, so the stored keys already carry the rotation and the decode path is unchanged. To stay conservative, B_{full} counts a single copy per global head; measured against the cache the implementation actually stores, our $10\times$, $20\times$, and $30\times$ are about $17\times$, $35\times$, and $52\times$ on the LongHealth context.

D BENCHMARKS AND EVALUATION PROTOCOL

D.1 ANSWER PROTOCOL

We score our algorithms and all baselines with a direct protocol: greedy decoding of at most 32 tokens after the answer tag. Attention Matching instead samples a brief chain of thought of up to 2,048 tokens, which makes every evaluation much slower. We scored the same caches under both protocols (Figure 4). The ranking of the methods is the same in all four cells, and under chain of thought the gap between RAPID++ and Attention Matching is larger (16 points instead of 8 on Llama-3.1-8B at $50\times$), so the direct protocol seems to be the conservative choice for RAPID, but it allows for cheaper and hence large scale evaluations.

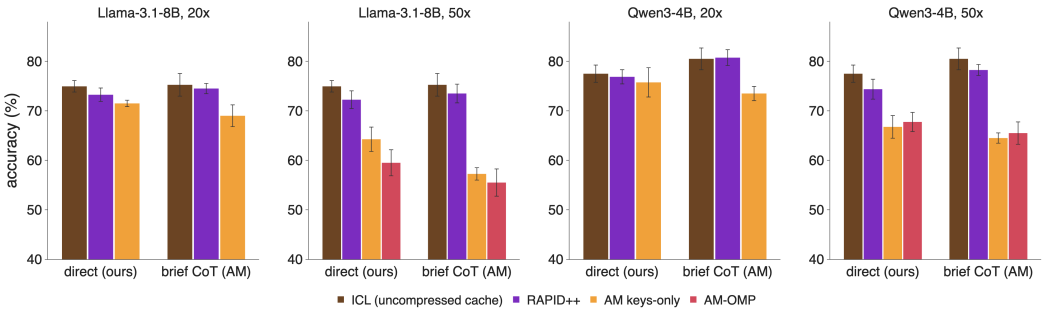


Figure 4: The same caches scored with our direct protocol and with the brief chain-of-thought protocol of Attention Matching, on the four LongHealth contexts; RAPID++ after 50 refinement steps. Bars pool 400 answers, except ICL and RAPID++ under the direct protocol, which pool 800 from two answer orders. AM-OMP was not run at $20\times$. Error bars are ± 1 standard error over the four contexts.

D.2 Extended LongHealth questions

The extension adds 100 questions to each of the four 5-patient contexts of Section 4.1 (patients 1–5, 6–10, 11–15, 16–20), 25 in each of four categories. Every question keeps LongHealth's format: four content options, shuffled, and a fifth option, always E, stating that the records do not contain enough information. It is the answer to the unanswerable questions and a distractor everywhere else, so its presence gives nothing away. Table 2 shows one question per category next to an original LongHealth question.

Category	Question and options (answer in bold)
LongHealth (original)	From the PSMA-PET-CT scan results for Mr. Williams, which vertebra showed the most notable PSMA uptake? C4 / L1 / **Th4** / Th7 / L5
Near miss	What was Mr. John Williams's serum potassium level upon his admission on November 12, 2022? 5.1 / 8.6 / 4.9 / **5.8** mEq/L
Cross-patient	Which two patients are explicitly documented to have a radiological finding of cardiomegaly? Mrs. Anderson and Mrs. Sample / Mrs. Sample and Mrs. Done / **Mrs. Anderson and Mr. Williams** / Mr. Miller and Mr. Williams
Timeline	How many days elapsed between the placement of Mrs. Anna Sample's port and the administration of Pegfilgrastim? 7 / 10 / **12** / 15 days
Unanswerable	What was Mrs. Jane Done's Lactate Dehydrogenase (LDH) level upon her discharge in January 2021? 342 / 450 / 366 / 244 U/L / **not enough information**

Table 2: One extended question per category and an original LongHealth question, all from the context of patients 1–5. Every extended question also offers the fifth option "The records do not contain enough information to answer this question", the answer only for unanswerable questions; options A–D are shuffled.

Generation. For each context and category, Gemini 3.1 Pro received the full context and the prompts of Appendix D.3, and wrote candidate questions in batches of 12, each with its answer, a rationale, and verbatim evidence quotes naming the patient and note; every batch was shown the questions already written. The four categories ask for:

- *Near miss:* one precise detail (a dose, a date, a count, a lab value with its unit, a side, a drug), with at least two distractors that appear in the same patient's records in a different role.
- *Cross-patient:* a comparison of documented facts of at least two patients (highest, earliest, shared, how many), not inferable from the diagnoses, with evidence quoted for every patient compared.
- *Timeline:* a link between events documented at different points of one patient's course (order, state at the time of another event, exact intervals between explicitly dated events).
- *Unanswerable:* a question that uses the patient's real diagnosis, treatments, and dates but asks for something the records do not contain, with four plausible options.

Gemini 3.1 Pro reached its daily request limit before one cell was complete, so 9 of the 100 cross-patient questions (all in the context of patients 1–5) were written by Gemini 3.8 Flash with the same prompts.

Validation. A second model, Gemini 3.8 Flash, checked every candidate. An answerable question was kept only if every evidence quote occurs verbatim in the named patient's notes, the checker given the records picked the gold answer on two independent runs, and the checker without the records, shown only the four content options, did not pick it on both runs. An unanswerable question was kept only if two checks with the records found no passage that answers it. From the candidates that passed, 25 per category and context were kept, rotating over the patients and skipping near-duplicates of questions already kept. Table 3 gives the counts. The most common failure was a question the checker answered correctly without the records: 82 of 240 near-miss, 93 of 240 timeline, and 265 of 588 cross-patient candidates. Cross-patient candidates also failed when their evidence covered only one patient.

Category	Generated	Passed validation	Kept
Near miss	240	154	100
Cross-patient	588	153	100
Timeline	240	140	100
Unanswerable	240	238	100
Total	1,308	685	400

Table 3: Extended LongHealth candidates generated, passing validation, and kept, per category over the four contexts. Cross-patient ran more batches because fewer of its candidates passed.

Scoring. We ask each question in two orders of the content options, as in LongHealth, and report accuracy on the three answerable categories (600 answers per method and model); Appendix I.3 reports the unanswerable ones.

D.3 Prompts

These are the prompts exactly as sent. Placeholders in braces are filled per call: {`ctx`} is the context of the five patients, as the compressed caches see it; {`task`} is one of the four category prompts; {`who`} names the patient the batch is about (name, ID, birthday, and diagnosis), rotating over the five patients of the context; {`idk`} is the fifth option; {`avoid`} lists the questions already written for the context and category; and {`n`} is 12. The first five cross-patient batches used an earlier version of the cross-patient prompt without its three bulleted rules, which we added after most of those candidates failed validation (the answer followed from the diagnoses in the record headers, or the evidence covered one patient).

Generation prompt (Gemini 3.1 Pro, one call per batch of 12)

```
You are writing questions for a long-context clinical benchmark in the
    style of LongHealth: multiple-choice questions about the patient
    records above. A model will answer them while holding the whole panel
     in context.
```

```
General rules (all questions):
- Exactly FOUR answer options in "options". Do NOT write an option like "
    not documented", "none of the above" or "all of the above": the
    benchmark adds its own fifth option, "{I don't know}", to every
    question.
- The four options must match in type, format and length, so that none
    stands out.
- Nothing may be answerable from general medical knowledge or from the
    wording of the question and options alone.
- Refer to patients by full name as written in the records (e.g. "Mrs.
    Anna Sample") and list their patient IDs in "patients".
- "evidence": VERBATIM quotes (copied character for character, 5-40 words
     each) from the records, each with its patient ID and note id (e.g.
    text_0).
- "rationale": one or two sentences on why the answer is right.
- "subtype": a short label for the kind of question.
- Vary the questions: different parts of the records, different kinds of
    facts. Do not ask about the name, birthday or diagnosis given in the
    record header.

{task}

Do not repeat or rephrase any of these existing questions:
{avoid}

Write exactly {n} questions.
```

Category prompt: Near miss ({task})

```
Task: NEAR-MISS questions about {who}. Each asks for one precise detail:
    a dose, a date, a count, a lab value with its unit, a measurement, a
    side or location, a specific drug, procedure or stage. The four
    options must all be close to each other and of the same form, so that
     only precise retrieval tells them apart: at least TWO distractors
    must be values that literally appear in this patient's records in a
    DIFFERENT role (the dose at another time, the value of another lab,
    another date in the course, the other side), the rest close
    variations (adjacent numbers, similar names). Make each question
    specific enough that exactly one option is right, and interesting (
    not the first sentence of a note). "correct" is the exact text of the
     right option.
```

Category prompt: Cross-patient ({task})

```
Task: CROSS-PATIENT questions about the panel. Answering each must
    require reading and COMPARING documented facts of at least two
    different patients. Vary the form: which of these patients had the
    highest / lowest / earliest / latest value of a specific documented
    quantity (a lab value, a dose, a score, a measurement, the date of a
    kind of event); which two patients share a specific documented detail
     (the same drug and dose, the same complication, the same finding);
    which finding is documented for one patient but not for another; how
    many patients received something specific; ordering patients by a
    documented quantity or date. When the options are patients or pairs
    of patients, use their names. The comparison must be exact from the
    records (no rounding, no dates the notes state inconsistently).
- The answer must NOT be inferable from the patients' names, ages, sexes
    or the diagnoses in the record headers (e.g. "Which patient had a
    Whipple procedure?" is inferable from "pancreatic cancer": do not ask
     it). Ask about details deep in the notes.
- Do not ask "which patient has X" where only one patient could plausibly
     have X; the question must require looking up the same kind of fact
    for several patients.
```

```
- "evidence" must quote the compared fact for EVERY patient the
    comparison relies on (for "which of these patients had the highest X
    ", quote X for each patient in the options).
"correct" is the exact text of the right option.
```

Category prompt: Timeline ({task})

```
Task: TIMELINE questions about {who}. Each must require linking two or
    more events of this patient's course that are documented at different
     points of the record (preferably in different notes or far apart):
    which of several events came first or last; what treatment or state
    the patient had at the time of another event; what happened
    immediately before or after an event; how many events of a kind
    occurred between two dated events; or the interval between two
    EXPLICITLY dated events (whole days, weeks or months, computed
    exactly, with distractor intervals clearly different from each other)
    . The answer must follow unambiguously from dates stated in the notes
    . "correct" is the exact text of the right option; "evidence" quotes
    each linked event.
```

Category prompt: Unanswerable ({task})

```
Task: UNANSWERABLE questions about {who}. Each question must SOUND
    grounded (use this patient's real diagnosis, procedures, drugs, dates
     and events) but ask for something the records do NOT contain, so the
     correct response is "{idk}". Good ways:
- a value that was never measured or reported (a lab or score that does
    not appear for this patient);
- a missing detail of a real event (a dose where no dose is given, a
    result that is not reported, a time or location that is not stated);
- an event that happened to ANOTHER patient in the panel, asked about
    this patient as if it happened to them;
- a follow-up, result or decision the records do not reach.
Avoid questions that are trivially unknowable (things no clinical record
    would contain) and avoid anything that can be derived from the
    records. The four options must be specific, plausible answers of the
    right type (ideally values or names that appear elsewhere in the
    records), none of them supported for this patient. Set "correct" to
    exactly "INSUFFICIENT". In "evidence" quote the passages closest to
    what is asked (what IS documented), and in "rationale" say exactly
    what is missing.
```

The checker answers each question twice with the records and, for answerable questions, twice without them. With the records, {pre} is LongHealth's own preamble for the patients the question concerns, all five options are shown, and the prompt is preceded by the context and a line of five equals signs; {instr} is "Answer using the patient records above. Reply with the letter only (A, B, C, D or E)." Without the records, only A–D are shown and {instr} is "No records are available. Make your best guess. Reply with the letter only (A, B, C or D)." For unanswerable questions, a second check looks for any passage that answers the question.

Answer check (Gemini 3.8 Flash)

```
{pre}

{q}
A. {a}
B. {b}
C. {c}
D. {d}{e}

{instr}
```

Presence check for unanswerable questions (Gemini 3.8 Flash)

```
{ctx}
```

```
=====

A benchmark question claims that the records above do NOT contain the
    information needed to answer it. Check carefully, including
    information that can be derived by simple reading or arithmetic from
    what the notes state.

Question: {q}
Candidate answers: {opts}

Reply in JSON: {{"answered": true or false, "answer": "the answer if the
    records contain it, else empty", "quote": "a verbatim supporting
    quote if answered, else empty"}}
```

E FULL RESULTS

This appendix lists every number behind the figures of Section 4. Tables 4 and 5 give every LongHealth result of Section 4.1, at every ratio each method was run. Appendix E.1 gives the extended questions of Section 4.2 for every model and method at $10\times$ and $20\times$, and Appendix E.2 the attention-output error on the question tokens. Throughout, every method is charged for the bits it stores, RAPID++ and RAPID are shown after 50 refinement steps (100 at $100\times$), the best compressed cache is in bold, and – marks a cell that was not run.

Method	$10\times$	$20\times$	$50\times$	$100\times$
Qwen3-4B (uncompressed cache 77.5, no context 34.9)				
Ada-SnapKV	38.9	36.1	36.1	36.6
KVzip	47.8	42.0	40.1	37.2
RDKV	49.6	39.8	35.8	30.1
Ada(KVzip)	68.5	55.0	43.2	38.1
AM keys-only	**76.2**	75.8	66.8	61.5
AM-OMP	–	–	67.8	61.5
RAPID++ before refinement	64.5	63.4	37.5	29.6
RAPID	75.8	72.8	67.8	40.5
RAPID++	76.1	**76.9**	**74.4**	**68.6**
Llama-3.1-8B (uncompressed cache 75.0, no context 39.3)				
Ada-SnapKV	46.4	41.9	42.5	40.6
KVzip	49.1	45.6	43.6	44.6
RDKV	43.4	33.9	25.2	20.6
Ada(KVzip)	69.8	58.6	42.9	42.1
AM keys-only	71.0	71.5	64.2	58.8
AM-OMP	–	–	59.5	53.2
RAPID++ before refinement	73.1	70.9	60.1	40.7
RAPID	**74.5**	73.0	65.0	49.8
RAPID++	74.4	**73.2**	**72.2**	**64.4**

Table 4: LongHealth accuracy (%) on Qwen3-4B and Llama-3.1-8B, pooled over the four 5-patient contexts and two answer orders (800 answers per cell).

Method	5×	10×	20×	30×	40×	50×
GLM-4.7-Flash (uncompressed cache 76.2, no context 34.5)						
SnapKV	49.9	37.6	35.1	34.1	32.8	32.9
KVzip	65.0	50.6	41.0	42.6	37.9	38.5
Ada(KVzip)	74.1	60.9	41.9	37.9	37.1	37.6
RAPID++ before refinement	**76.0**	**78.8**	74.4	66.1	48.8	36.2
RAPID	–	76.8	74.1	67.9	44.0	41.8
RAPID++	–	76.0	**77.2**	**75.4**	**73.8**	**65.6**
GPT-OSS-120B (uncompressed cache 76.0, no context 43.9)						
Ada-SnapKV	49.9	43.8	44.6	42.8	44.8	43.8
KVzip	60.9	48.2	43.6	42.1	42.5	42.9
RDKV	66.8	54.6	46.6	41.8	42.0	41.4
Ada(KVzip)	73.2	69.0	49.4	44.6	43.9	41.9
RAPID++ before refinement	75.6	73.6	62.2	52.6	47.9	44.4
RAPID	**76.1**	75.5	69.9	63.0	57.2	44.5
RAPID++	75.9	**76.4**	**74.9**	**73.6**	**69.4**	**64.0**

Table 5: LongHealth accuracy (%) on GLM-4.7-Flash and GPT-OSS-120B, pooled over the four 5-patient contexts and two answer orders (800 answers per cell).

E.1 Extended questions

Figure 3a shows the extended questions on Qwen3-4B and Llama-3.1-8B; Figure 5 shows the same comparison on GLM-4.7-Flash and GPT-OSS-120B, and Table 6 lists every value. The pattern of the two small models holds on both large ones. At $10\times$, RAPID++ is within about two points of the uncompressed cache on the extension average (55.5 against 57.0 on GLM-4.7-Flash, 43.7 against 43.8 on GPT-OSS-120B). At $20\times$ it stays within five points (52.2 and 40.3), while RAPID falls to 46.3 and 28.8, and the eviction baselines to between 4 and 21 percent, below a quarter of the uncompressed cache on GPT-OSS-120B. The gap opens mostly on the near-miss questions, where GPT-OSS-120B at $20\times$ answers 60.0 percent with RAPID++, 36.0 with RAPID, and at most 11.0 with an eviction baseline, against 69.0 uncompressed.



Figure 5: Accuracy (%) on the original and extended questions for GLM-4.7-Flash (top) and GPT-OSS-120B (bottom) at 10× (left) and 20× (right); Qwen3-4B and Llama-3.1-8B are in Figure 3a. "Original questions" is the same cache on LongHealth (800 answers); the three extended categories have 100 questions over the four contexts (200 answers in two orders), and the extension average pools them (600 answers). Attention Matching was run on Qwen3-4B and Llama-3.1-8B only, and RDKV not on GLM-4.7-Flash. Dashed ticks are the uncompressed cache, with its extension average printed above; error bars are ±1 standard error over the four contexts. All values are in Table 6.

Ratio	Method	Original	Ext. average	Near miss	Cross-patient	Timeline
Qwen3-4B						
	Uncompressed cache	77.5	49.3	88.0	33.0	27.0
	No context	34.9	0.0	0.0	0.0	0.0
10×	KVzip	47.8	7.0	8.0	9.0	4.0
	RDKV	49.6	12.7	11.0	14.0	13.0
	Ada(KVzip)	68.5	33.8	67.5	22.0	12.0
	AM keys-only	**76.2**	36.8	78.5	20.0	12.0
	RAPID	75.8	45.7	81.5	**35.0**	20.5
	RAPID++	76.1	**47.7**	**84.5**	34.5	**24.0**
20×	KVzip	42.0	4.8	2.0	6.5	6.0
	RDKV	39.8	6.8	7.5	5.0	8.0
	Ada(KVzip)	55.0	10.2	20.0	6.5	4.0
	AM keys-only	75.8	35.3	69.0	21.5	15.5
	RAPID	72.8	38.0	66.5	32.0	15.5
	RAPID++	**76.9**	**46.0**	**83.0**	**35.0**	**20.0**
Llama-3.1-8B						
	Uncompressed cache	75.0	50.3	85.0	39.0	27.0
	No context	39.3	4.0	2.0	7.0	3.0
10×	KVzip	49.1	16.5	25.5	12.5	11.5
	RDKV	43.4	29.7	32.0	32.0	25.0
	Ada(KVzip)	69.8	36.3	68.5	25.0	15.5
	AM keys-only	71.0	43.2	74.0	29.5	26.0
	RAPID	**74.5**	47.7	78.5	**39.5**	25.0
	RAPID++	74.4	**50.7**	**85.5**	**39.5**	**27.0**
20×	KVzip	45.6	14.3	22.5	13.0	7.5
	RDKV	33.9	23.2	18.5	28.0	23.0
	Ada(KVzip)	58.6	23.3	42.5	20.0	7.5
	AM keys-only	71.5	39.5	67.5	26.5	24.5
	RAPID	73.0	**44.8**	**69.5**	38.0	**27.0**
	RAPID++	**73.2**	42.7	62.5	**42.0**	23.5
GLM-4.7-Flash						
	Uncompressed cache	76.2	57.0	86.0	51.0	34.0
	No context	34.5	2.5	0.5	1.0	6.0
10×	KVzip	50.6	30.3	34.0	32.5	24.5
	Ada(KVzip)	60.9	29.8	43.5	27.5	18.5
	RAPID	**76.8**	52.7	79.0	47.0	32.0
	RAPID++	76.0	**55.5**	**84.5**	**47.5**	**34.5**
20×	KVzip	41.0	21.0	18.5	23.5	21.0
	Ada(KVzip)	41.9	12.0	16.0	7.5	12.5
	RAPID	74.1	46.3	69.0	43.0	27.0
	RAPID++	**77.2**	**52.2**	**79.0**	**44.0**	**33.5**
GPT-OSS-120B						
	Uncompressed cache	76.0	43.8	69.0	39.0	23.5
	No context	43.9	1.8	1.5	2.0	2.0
10×	KVzip	48.2	5.2	9.0	5.5	1.0
	RDKV	54.6	18.7	17.5	26.0	12.5
	Ada(KVzip)	69.0	26.8	39.5	27.0	14.0
	RAPID	75.5	39.3	61.0	35.0	22.0
	RAPID++	**76.4**	**43.7**	**68.5**	**37.5**	**25.0**
20×	KVzip	43.6	3.8	5.5	4.5	1.5
	RDKV	46.6	10.5	11.0	11.0	9.5
	Ada(KVzip)	49.4	5.3	8.5	5.5	2.0
	RAPID	69.9	28.8	36.0	31.0	19.5
	RAPID++	**74.9**	**40.3**	**60.0**	**38.0**	**23.0**

Table 6: Every model and method at 10× and 20× (accuracy, %), on the four contexts and two answer orders. Original: the same cache on the LongHealth questions (800 answers). Ext. average: the three categories of extended questions (600 answers); each category has 200 answers.

E.2 ATTENTION-OUTPUT ERROR ON THE QUESTION TOKENS

Table 7 lists the attention-output errors of Figure 3b, for every method on Llama-3.1-8B and Qwen3-4B at $20\times$, and Figure 6 sets each method's error on the answerable extended questions against its accuracy on them. RAPID++ has the lowest error on both models, $2.3\times$ below the nearest baseline, Ada(KVzip) with KIVI-2, on Qwen3-4B (0.026 against 0.059) and $5.7\times$ below it on Llama-3.1-8B (0.011 against 0.062), where the two differ by less than two points of accuracy (42.7 against 41.0). The error separates RAPID++ from the baselines, but it does not rank caches that are close in error. On Llama-3.1-8B, RAPID has a higher error than RAPID++ (0.018) and a higher accuracy (44.8). On Qwen3-4B, RDKV has about the error of Attention Matching (0.070 against 0.072) and a fifth of its accuracy (6.8 against 35.3). The error of RAPID++ is the same on the extended questions as on the original ones on both models, and cross-patient questions have the lowest error for almost every cache; Attention Matching is the exception, with its highest error there on both models.

Method	LongHealth	Extended	Near miss	Cross-patient	Timeline	Unanswerable
Llama-3.1-8B						
RAPID++	**0.0120**	**0.0113**	**0.0123**	**0.0089**	**0.0119**	**0.0121**
RAPID	0.0189	0.0180	0.0195	0.0156	0.0185	0.0186
Ada(KVzip) + KIVI-2	0.0654	0.0623	0.0666	0.0585	0.0622	0.0618
Attention Matching	0.0707	0.0708	0.0719	0.0749	0.0682	0.0683
Ada-SnapKV	0.0958	0.0826	0.0961	0.0675	0.0878	0.0792
RDKV	0.1305	0.1199	0.1327	0.1040	0.1223	0.1204
Ada(KVzip)	0.1047	0.0990	0.1044	0.0891	0.1023	0.1001
KVzip	0.1283	0.1210	0.1275	0.1128	0.1243	0.1193
No context	0.4616	0.4508	0.4667	0.4112	0.4657	0.4597
Uncompressed cache	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
Qwen3-4B						
RAPID++	**0.0262**	**0.0260**	**0.0271**	**0.0226**	**0.0270**	**0.0274**
RAPID	0.0676	0.0663	0.0687	0.0587	0.0675	0.0702
Ada(KVzip) + KIVI-2	0.0600	0.0585	0.0605	0.0593	0.0581	0.0563
Attention Matching	0.0714	0.0710	0.0720	0.0723	0.0704	0.0694
Ada-SnapKV	0.0986	0.0830	0.0990	0.0645	0.0870	0.0816
RDKV	0.0802	0.0698	0.0808	0.0578	0.0716	0.0690
Ada(KVzip)	0.0974	0.0922	0.0991	0.0835	0.0940	0.0923
KVzip	0.1218	0.1153	0.1238	0.1069	0.1166	0.1139
No context	1.8904	1.8229	1.8840	1.5618	1.9010	1.9447
Uncompressed cache	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Table 7: Relative attention-output error on the question tokens at $20\times$, the values behind Figure 3b: the mean over layers of $\|o - o^*\|^2/\|o^*\|^2$, each layer read from the exact cache's inputs, averaged over the questions of the four LongHealth contexts (400 original questions; 100 per extended category; Extended pools all 400, unanswerable included). Standard errors over the four contexts are below 7% of each value. Lowest compressed cache per column in bold. The uncompressed cache is not exactly zero because its read runs through the same bf16 path.

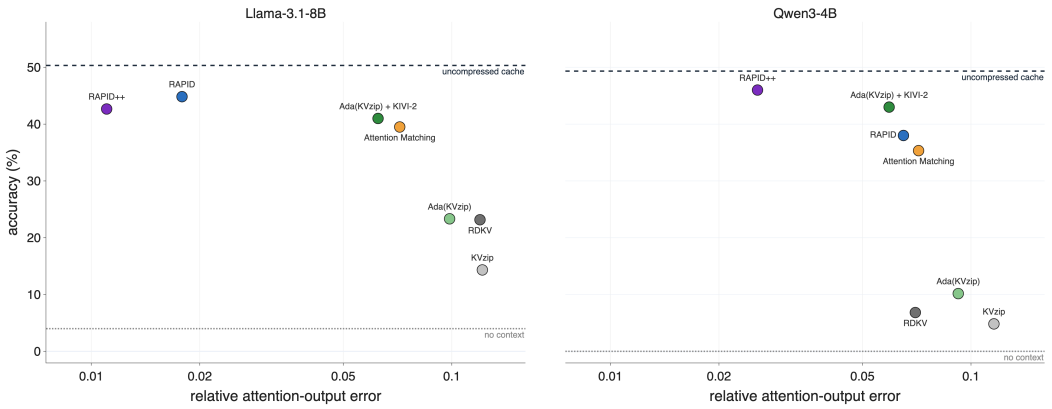


Figure 6: Accuracy (%) on the answerable extended questions (600 answers) against the relative attention-output error on the same questions (Table 7), at $20\times$, one point per method; log scale. Dashed: the uncompressed cache; dotted: no context.

F Ablations

We change one part of RAPID++ at a time on GPT-OSS-120B at $50\times$, on two LongHealth contexts of five patients, and refine each variant for 50 steps (Figure 7). The full method reaches 67.8 percent. Without the shared key–value basis it reaches 58.2, with a uniform rank across layers 61.8, and without the prefix floor 64.5. Without per-token prefixes, token importance, the five probes, or quantization it falls to 45–50 percent, the accuracy with no context. The alternations converge quickly: one reaches 63.8, and sixteen give the same allocation as four (mean rank, prefix, and bit width) and 66.5, within noise of the recipe.

The one-probe variant reads its rate at rank 48, while the $50\times$ solve operates near rank 190. Moving the probe to rank 200 gives 44.0 instead of 44.8, still at the no-context level, with the layer ranks as misordered as before (layer 3 receives 96–128 coordinates where the recipe gives about 380). The depth of the probe is not the problem: with one number per layer, the coordinates within a layer can only be ordered by their latent energy, while five probes give the shape of the rate across them.

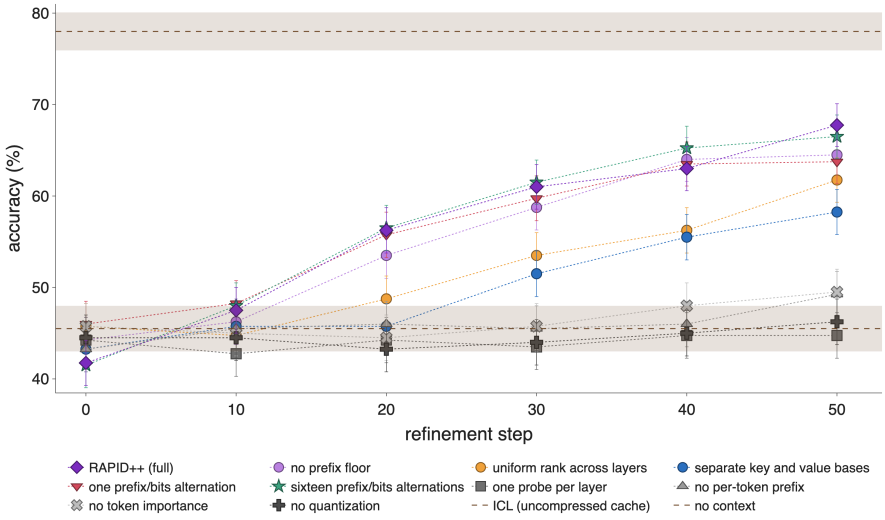


Figure 7: Accuracy during refinement on GPT-OSS-120B at $50\times$ with one part of RAPID++ changed at a time: no prefix floor (tokens may be evicted); the same rank in every layer (prefixes and bits still allocated); separate bases for keys and values; one or sixteen alternations instead of four; one probe per layer instead of five; no per-token prefixes (every token keeps the layer rank); no token importance (every token weighs one); and no quantization (coordinates in bf16). Step 0 is each variant's own allocated cache. Each point pools patients 1–10 in two contexts and two answer orders (400 answers); error bars and bands are ± 1 binomial standard error.

F.1 The same runs through the per-layer loss

The per-layer objective of Step 3 is a relative error, so it is comparable across layers (Figure 8). It does not rank the variants as accuracy does. A uniform rank ends with the lowest mean loss, 0.056 against 0.060 for the full method, and scores six points lower: equalizing the ranks equalizes the per-layer problems, which lowers an unweighted mean, but the layers it improves are not the ones the questions need. Split by sensitivity, the two change places. On the six layers the full method's probes rate most sensitive, the full method has the lower loss (0.063 against 0.067); on the six rated least sensitive, the uniform rank does (0.038 against 0.046), and it more than doubles the loss of layer 7 (0.041 to 0.096). The worst layer is the one view that tracks the failures: three of the four variants at the no-context level leave a layer above 0.2 after 50 steps (one probe 0.233, one probe at the operating rank 0.205, no quantization 0.329), against 0.129 for the full method. Without per-token prefixes or token importance the mean loss reaches only 0.087 and 0.079, yet accuracy is at the floor: refinement does its job, but the information was gone before it started.

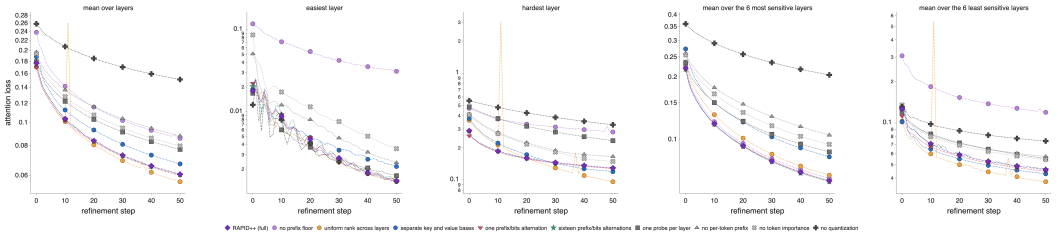


Figure 8: The runs of Figure 7 seen through the per-layer attention loss of Step 3: the mean over the 18 compressed layers; the easiest and the hardest layer at each step; and the mean over the six layers the full method's probes rate most and least sensitive, so that every variant is judged on the same layers. Axes are logarithmic. Curves average the two LongHealth contexts.

G How the budget is allocated

This appendix shows what the allocation measures and what it decides, on GPT-OSS-120B and LongHealth, where only the 18 full-attention layers are compressed.

What the probes measure. What matters to the solver is how fast a layer's error falls with rank, not how high it starts (Figure 9, left). Layer 1 barely improves with rank and keeps the 8-coordinate floor; layer 33 reaches almost zero error by rank 256 and stops there; layers 7 and 29 keep improving past 256 and receive ranks of about 380 and 580. The SVD does not give this ranking (right panel): layer 29 has the smallest key–value reconstruction error at rank 256 (0.018) but one of the largest attention-output errors (0.21), layer 33 the reverse (0.063 and 0.012), and across layers the two errors are uncorrelated (Spearman -0.15). Allocating by SVD energy would starve the two layers the allocation funds most.

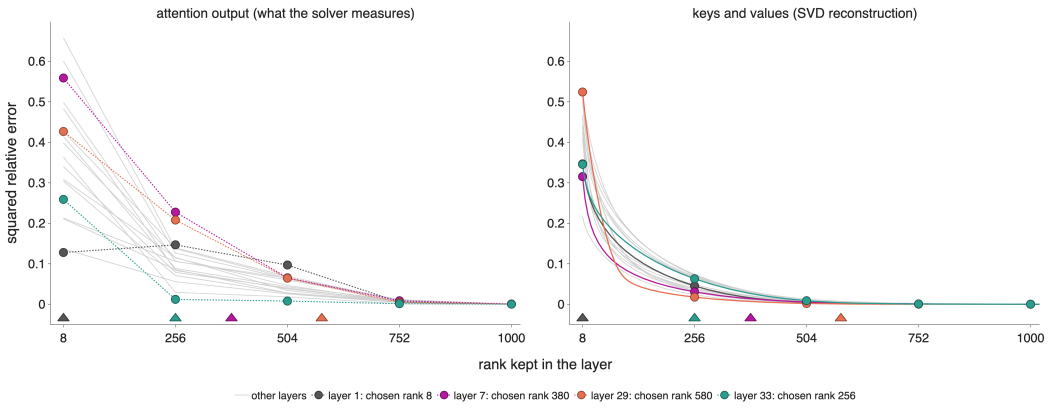


Figure 9: GPT-OSS-120B at $50\times$, each full-attention layer truncated to a given rank while every other layer is exact. Left: the relative error of the attention output at the five probe ranks, which the allocation uses. Right: the relative error of the layer's keys and values reconstructed from its SVD, at every rank. Averaged over two LongHealth contexts, which agree within about 1 percent. Triangles mark the rank the allocation gives each highlighted layer.

How the allocation changes with the budget. At $5\times$ every layer keeps nearly its full width and 3–7 percent of the bits (Figure 10). As the budget tightens, the bits concentrate where the probe curves fall slowest: layer 7 grows from 7.2 to 13.0 percent and layer 29 from 6.1 to 10.3, while layer 1 drops to 0.7 at $50\times$. The mean width of a stored coordinate falls from about 4 bits at $5\times$ to about 2.5 at $20\times$ and then stays there: beyond $20\times$ the solver cuts rank, not precision.

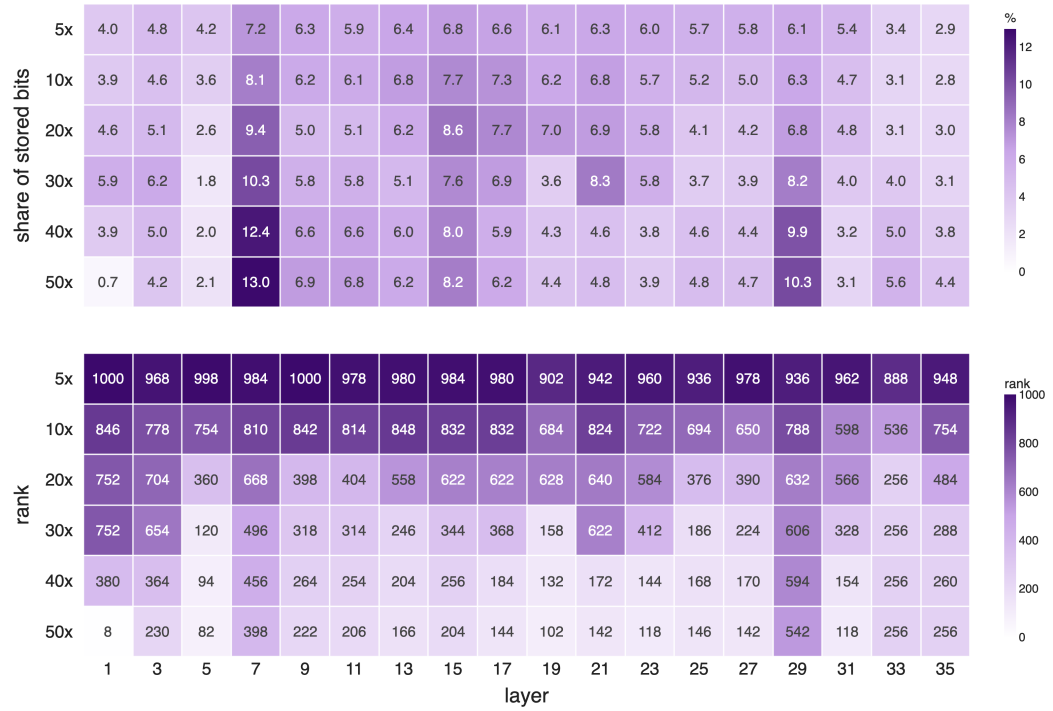


Figure 10: The allocation of RAPID++ on GPT-OSS-120B from $5\times$ to $50\times$: each layer's share of the stored bits (top, percent) and its rank (bottom), averaged over the four LongHealth contexts.

Ranks and per-token prefixes. The rank of a layer is the longest prefix any token keeps; the mean prefix is what a typical token keeps (Figure 11). RAPID++ keeps a wide basis but gives most tokens a short prefix (at $50\times$, layer 29 has rank 542 and mean prefix 150), while RAPID truncates the tokens of a layer to more similar lengths (rank 112, mean prefix 94). The two differ only in their reference queries, and so in the token importance the allocation receives; at $50\times$, RAPID++ reaches 64.0 percent after refinement and RAPID 44.5, the no-context level.

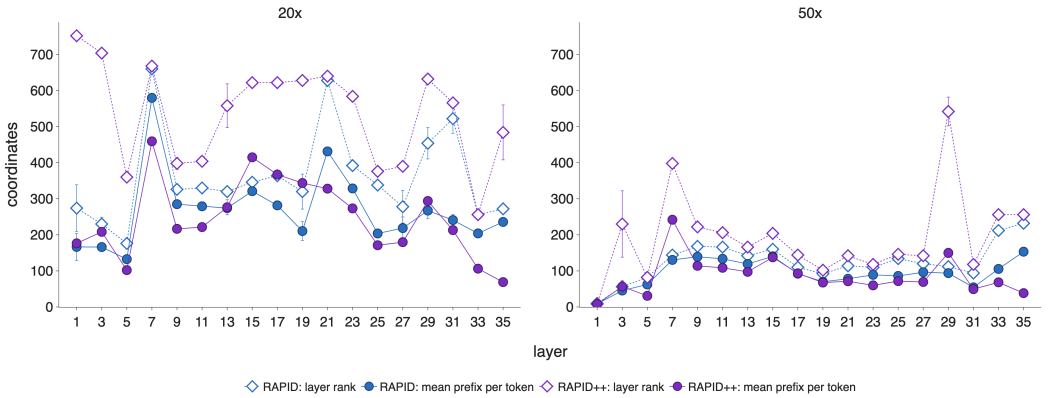


Figure 11: Layer rank (hollow diamonds) and mean prefix per token (filled circles) of RAPID and RAPID++ on GPT-OSS-120B at $20\times$ and $50\times$, averaged over the four LongHealth contexts; error bars are ± 1 standard error over contexts.

H OUR COMPONENTS ADDED TO THE BASELINES

The baselines of Section 4.1 are run as published. Here we add each of our two components to them, on Qwen3-4B.

Refinement. To separate the effect of the refinement from that of our allocation, we refine the $50\times$ cache of each baseline with the same training as RAPID++: the same reference queries, objective, optimizer, learning rate, and number of steps (Figure 12). Refinement raises the Attention Matching cache from 67.5 to 70.9 percent and Ada(KVzip) from 43.1 to 47.2, and barely moves RDKV (35.9 to 37.6, near the no-context level). RAPID and RAPID++ start lower, from their allocated caches, and end higher, at 67.8 and 74.4. Refinement helps every cache that keeps enough of the context, but it does not close the gap left by a weaker allocation.

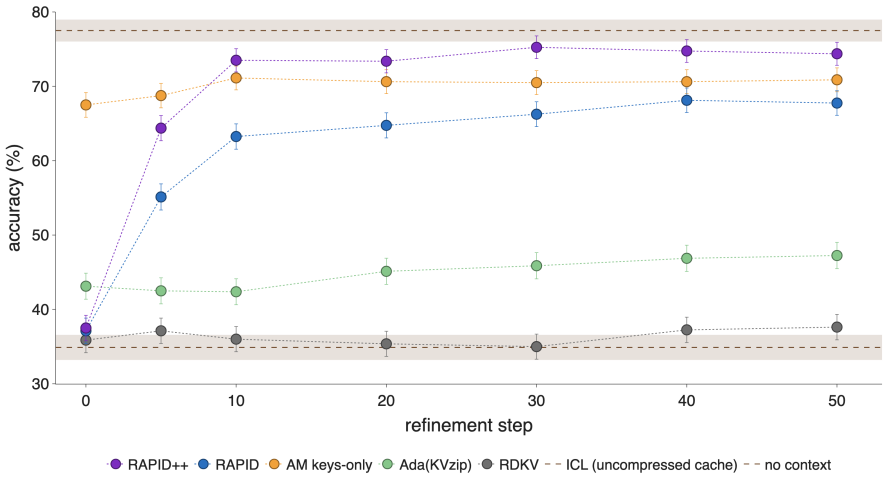


Figure 12: Qwen3-4B at $50\times$: accuracy against refinement step when the refinement of RAPID++ starts from each method's cache. Step 0 is the method's own cache: for Attention Matching its published $50\times$ keys-only cache; RDKV, which stores quantized rows, is trained together with its quantization grid. Each point pools the four contexts and two answer orders (800 answers); error bars and bands are ± 1 binomial standard error.

Quantization. Storing the pairs Ada(KVzip) keeps in KIVI-2 lets eviction keep about five times as many tokens at the same size, and helps it substantially: at $20\times$, from 55.0 to 75.2 percent on LongHealth and from 10.2 to 43.0 on the extended questions (Table 8). Quantizing Attention Matching's $10\times$ cache instead of compacting it further matches or slightly improves on its own caches of the same size: at 8 bits ($17.8\times$) it scores as its $20\times$ cache on LongHealth and 1.7 points higher on the extended questions, and at 2 bits ($52.7\times$) it is on par with its $50\times$ cache.

Method	Served ratio	LongHealth	Extended questions
Uncompressed cache	1×	77.5	49.3
About 20×			
Ada(KVzip)	20×	55.0	10.2
Ada(KVzip) + KIVI-2	20×	75.2	43.0
Attention Matching (keys-only)	20×	75.8	35.3
Attention Matching 10× + KIVI-8	17.8×	75.8	37.0
RAPID++	20×	**76.9**	**46.0**
About 50×			
Ada(KVzip)	50×	43.2	2.0
Ada(KVzip) + KIVI-2	50×	69.6	30.3
Attention Matching (OMP)	50×	67.8	23.0
Attention Matching 10× + KIVI-2	52.7×	68.0	24.7
RAPID++	50×	**74.4**	**39.7**

Table 8: Quantization added to two baselines on Qwen3-4B, each at the size it is served. Eviction: the pairs Ada(KVzip) keeps, stored in KIVI-2. Attention Matching: its own 10× cache (Zweiger et al., 2026) quantized with KIVI-8 or KIVI-2, next to its own 20× and 50× caches. LongHealth: 800 answers per cell; extended questions: the three answerable categories, 600 answers. Best compressed cache per block in bold.

Summary. Each component helps a baseline on its own, and neither closes the gap to RAPID++, which chooses how many tokens to keep, in how many dimensions, and at what precision within one budget, and then refines every layer against its attention output. Because these combinations are not published methods, and our own allocation could be extended in the same way, we keep them out of the main comparison.

I END-TO-END TRAINING

This appendix completes Section J.1: the Llama-3.2-3B curves at 53× and 106×, the Qwen3-4B curves at 50× and 100×, and abstention on the unanswerable questions. In every figure, the cartridge is trained with the recipe of Cartridges (Eyuboglu et al., 2025) from its own initialization, and the other caches continue with the same self-study data.

I.1 LLAMA-3.2-3B ON A 110K-TOKEN CONTEXT

Figure 13 uses the setting of Cartridges: the ten patients of the first two contexts in one 110K-token context, long enough that the model's own accuracy with the full context drops to 53 percent. After 100 refinement steps at 53×, RAPID++ reaches 51.8 percent; 8 self-study steps (0.6 percent of an epoch) raise it to 57.0, and 16 steps (1.2 percent) to 60.0, above the uncompressed cache. The cartridge, and self-study started from the unrefined cache, need far more steps and stay below the refined starts. Accuracy on the extended questions for this setting is in Table 9.

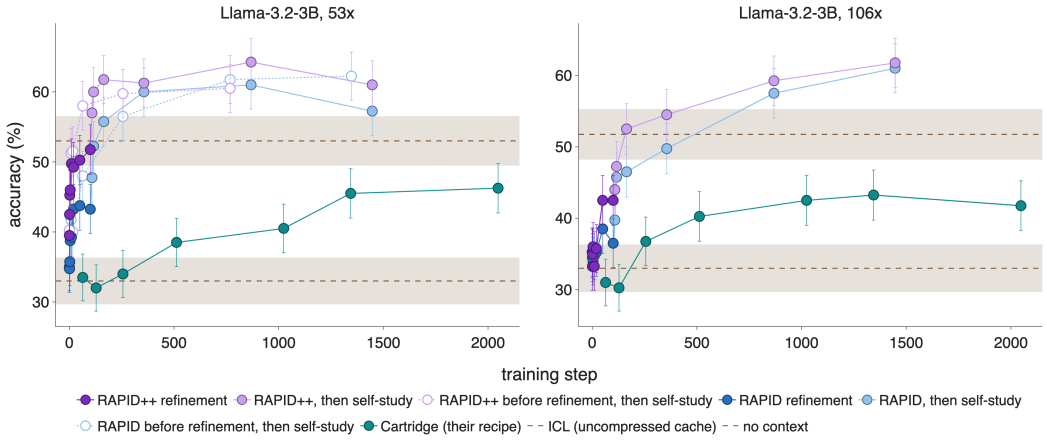


Figure 13: Llama-3.2-3B, one 110K-token context of 10 patients, at 53× (left) and 106× (right): per-layer refinement for 100 steps (dark), then self-study on the same cache (light), drawn at 100+ its own step; one step is a batch of 32 packed conversations and one epoch is 1,347 steps. Hollow markers run self-study directly on the allocated cache and start at step 0. The cartridge is shown up to 2,048 steps. Each point is 100 questions in two orders (200 answers); error bars and bands are ±1 binomial standard error.

I.2 QWEN3-4B AT 50× AND 100×

On Qwen3-4B, patients 1–5, at 50× (Figure 14), 32 self-study steps (4.6 percent of an epoch) raise the unrefined RAPID++ cache from 42.0 to 67.5 percent and the Attention Matching cache from 62.5 to 73.0, and the refined RAPID++ cache ends the epoch at 76.5, above the uncompressed cache on this context (75.0). The cartridge stays below 30 percent on the extended questions over the steps we scored.

At 100× (Figure 15) the cartridge is at the closest size its code allows, 106.9×. On LongHealth, self-study raises the refined RAPID++ cache from 59.5 to 75.5 percent within 64 steps (9 percent of an epoch) and ends the epoch at 72.0; the Attention Matching cache ends at 71.5, the unrefined RAPID++ cache at 64.5, and the cartridge at 44.5. On the extended questions, every cache that starts from a compressed context ends between 40 and 45 percent, against 28.7 for the cartridge and 54.7 for the uncompressed cache.

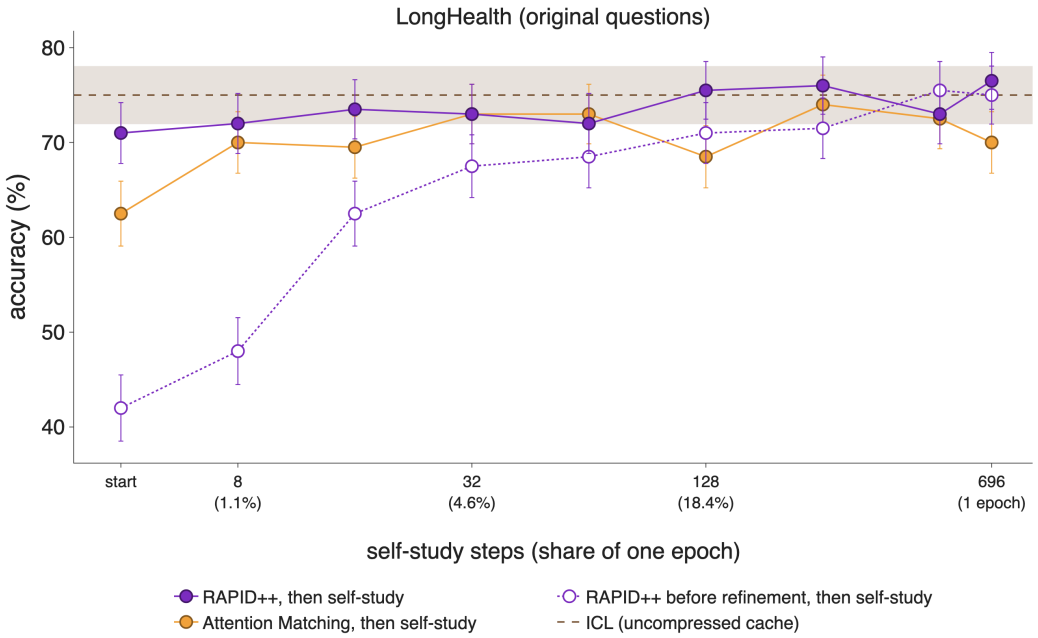


Figure 14: Self-study on each $50\times$ cache of Qwen3-4B, patients 1–5: accuracy on LongHealth (left, 200 answers per point) and on the answerable extended questions (right, 150 answers). "Start" is the cache before self-study; one step is one batch of the self-study data of Cartridges, and one epoch is 696 steps (the share of an epoch is in parentheses). The cartridge run stopped at step 448 and was scored on the extended questions only. Error bars and bands are ± 1 binomial standard error.

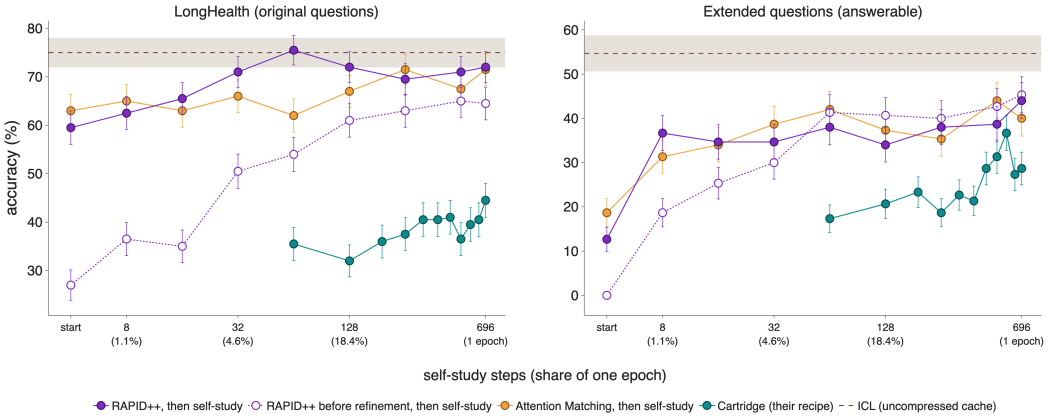


Figure 15: As Figure 14, at $100\times$; the cartridge is at $106.9\times$ and scored on both benchmarks.

I.3 Abstention on unanswerable questions

A quarter of the extended questions ask for something the records do not contain, and the right answer is the fifth option. We leave them out of the accuracies of the main text, because a cache that has lost the context abstains on everything. They do show what end-to-end training does to a cache (Table 9). Before self-study, the compressed caches abstain about as often as the uncompressed cache when the information is absent, and often more when it is present, where abstaining means the answer was lost in compression. Self-study lowers abstention on both kinds of question in all but one case: the caches answer more of what they can answer, and also answer questions the records do not support. The effect is largest for the cartridge, which abstains on 16–28 percent of

the unanswerable questions against 62–80 for the uncompressed cache, and for Attention Matching followed by self-study (36 and 8 percent on Qwen3-4B). RAPID++ followed by self-study keeps 49–72 percent while answering more of the answerable questions than both. End-to-end training on generated conversations trades calibration for coverage, and starting it from a strong per-layer approximation limits how much is lost.

Setting	Cache	Answerable acc.	Abstains, info absent	Abstains, info present
Qwen3-4B, 50×	Uncompressed cache	54.7	80.0	18.7
	RAPID++	38.7	80.0	26.7
	RAPID	20.0	100.0	60.0
	Attention Matching	28.7	88.0	50.7
	RAPID++ + self-study	52.0	68.0	9.3
	RAPID++ before refinement + self-study	55.3	40.0	6.7
	Attention Matching + self-study	45.3	36.0	6.7
	Cartridge	26.0	16.0	7.3
Qwen3-4B, 100×	Uncompressed cache	54.7	80.0	18.7
	RAPID++	12.7	92.0	66.7
	RAPID	1.3	100.0	92.7
	Attention Matching	18.7	92.0	52.0
	RAPID++ + self-study	44.0	72.0	10.7
	RAPID++ before refinement + self-study	45.3	56.0	12.0
	Attention Matching + self-study	40.0	8.0	2.7
	Cartridge	28.7	18.0	4.7
Llama-3.2-3B, 53×	Uncompressed cache	31.7	62.0	10.3
	RAPID++	27.0	68.0	14.7
	RAPID	19.0	76.0	24.3
	RAPID++ + self-study	33.7	56.0	8.0
	RAPID++ before refinement + self-study	34.7	56.0	3.3
	RAPID + self-study	29.7	46.0	4.7
	RAPID before refinement + self-study	30.7	66.0	8.7
	Cartridge	22.0	28.0	5.3
Llama-3.2-3B, 106×	Uncompressed cache	31.7	62.0	10.3
	RAPID++	21.7	73.0	17.0
	RAPID	15.7	57.0	16.3
	RAPID++ + self-study	26.7	49.0	9.3
	RAPID + self-study	25.7	58.0	14.7
	Cartridge	22.0	27.0	0.7

Table 9: Abstention on the extended questions, before and after one epoch of self-study (the 50× Qwen3-4B cartridge after 448 steps, the Llama-3.2-3B cartridges after 1,344, and Llama-3.2-3B RAPID++ before refinement + self-study after 1,024). Answerable acc.: accuracy on the three answerable categories. Abstains, info absent: the share of unanswerable questions answered "the records do not contain enough information", the right answer there (higher is better). Abstains, info present: the share of answerable questions answered that way (lower is better). Qwen3-4B: patients 1–5, 150 answerable and 50 unanswerable answers per row; Llama-3.2-3B: one 110K-token context of patients 1–10, 300 and 100.

J Additional benchmarks and architectures

We evaluate one more model, Gemma-4-31B (Gemma Team, 2026), chosen because its attention is strongly cache-aware: five of every six layers attend only to a window of the last 1,024 tokens. Appendix C describes how we adapt RAPID and the baselines to it. One adaptation matters for the accounting. In its global layers the key and the value of a token come from a single projection: the value is the projection normalized, and the key is the same vector scaled per channel and rotated by RoPE. The key is therefore determined by the value and the token's position, so one vector per head and token suffices, where the reference implementation caches two. This factor of two comes

from the architecture, not from compression, and we credit it to no method: the uncompressed size counts one vector per head, the baselines store only the values and rebuild the keys when serving, and RAPID compresses the stored keys and values jointly in its shared basis. We use two free-form benchmarks. MTOB (Tanzer et al., 2024) asks for translations between English and Kalamang, a language with almost no web presence, using a grammar book given in context; we use both the medium book, about 46K tokens, and the long one, about 100K tokens, and score 100 test sentences by chrF (Popović, 2015). QASPER (Dasigi et al., 2021) asks free-form questions about NLP papers; we use nine contexts of 10 papers, about 50K tokens each, and score their 290 questions by token F1.

Accuracy. Table 10 lists the results. RAPID++ stays within about one point of the uncompressed cache at $10\times$ on every benchmark, and within a few points at $30\times$, including with the 100K-token grammar book, while the best eviction baseline loses about half of the uncompressed score already at $20\times$. RAPID, run on the long grammar book, sits between them, two to three points below RAPID++.

Attention-output error. The bottom block of Table 10 measures the quantity RAPID minimizes, the relative attention-output error, on the question tokens, which no method was fit to, over all 60 layers: the 10 global layers and the 50 local windows. RAPID++ has the lowest error in every column, three to six times below the best eviction baseline at the same ratio, and its error is even across the two kinds of layer (0.015–0.029 on the global layers, 0.005–0.028 on the local windows). Most of the baselines' error sits in the local windows. At $20\times$ a window keeps 51 of its 1,024 tokens and at $30\times$ 34. SnapKV's 64 pinned recent tokens no longer fit at $20\times$, and its local-layer error rises from about 0.01 at $10\times$ to about 0.3; Ada(KVzip)'s 32 pinned tokens leave two to score at $30\times$, and its local-layer error rises the same way. As on LongHealth (Appendix E.2), the error separates RAPID++ from eviction but does not rank the evictors: SnapKV at $10\times$ has the lowest baseline error on every benchmark, and the lowest baseline accuracy on QASPER.

	MTOB, 46K tok.			MTOB, 100K tok.			QASPER		
	$10\times$	$20\times$	$30\times$	$10\times$	$20\times$	$30\times$	$10\times$	$20\times$	$30\times$
Uncompressed		48.8			42.5			48.5	
SnapKV	28.2	21.2	19.5	32.8	23.9	21.3	20.4	14.5	14.8
KVzip	30.0	21.6	17.1	28.8	21.0	16.1	36.3	25.9	20.7
Ada(KVzip)	28.5	21.0	16.8	29.8	23.2	18.0	32.6	19.8	16.5
RAPID	–	–	–	40.6	39.2	37.0	–	–	–
RAPID++	**49.1**	**49.1**	**46.8**	**42.5**	**41.0**	**40.2**	**47.6**	**46.1**	**45.2**
Attention-output error on the question tokens (↓)									
SnapKV	.020	.264	.267	.022	.256	.261	.030	.250	.274
KVzip	.047	.068	.081	.059	.086	.105	.051	.076	.094
Ada(KVzip)	.047	.084	.282	.057	.115	.297	.053	.107	.314
RAPID	–	–	–	.021	.046	.073	–	–	–
RAPID++	**.007**	**.013**	**.023**	**.007**	**.015**	**.028**	**.007**	**.012**	**.019**

Table 10: Gemma-4-31B with every layer compressed, local windows included. MTOB: chrF on 100 test sentences with the medium (about 46K tokens) or the long (about 100K tokens) grammar book in context; QASPER: token F1, 9 contexts of 10 papers, 290 questions, pooled. The baselines are untrained and byte-matched, each evicting the local windows at the same ratio; RAPID was run on MTOB with the long book only. RAPID++ and RAPID after 50 refinement steps. Best compressed cache per column in bold; – marks a cell not run. Bottom block: relative error of the attention output on the question tokens, $\|o - o^*\|^2 / \|o^*\|^2$ against the uncompressed cache, averaged over all 60 layers (the 10 global layers and the 50 local windows) and over the questions, each layer fed the exact input; lowest per column in bold.

J.1 Adding end-to-end training

In many practical settings, traces of how users query a context are available, and a few steps of end-to-end training on them may be affordable. We measure what such training adds using the self-

study data and objective of Cartridges (Eyuboglu et al., 2025): we continue training the compressed caches end to end, and compare with a cartridge trained by their recipe at the same size (Figure 16).

End-to-end training is most effective when it starts from a strong initialization. After a fraction of one epoch of self-study, the caches that start from a compressed context are far ahead of the cartridge trained from its own initialization, on both LongHealth and the extended questions, and the refined RAPID++ cache reaches the uncompressed cache on LongHealth and comes close to it on the extended questions. We also observe that self study end-to-end training without makes cache more willing to answer questions it has no information for. On the unanswerable extended questions, where the right answer is that the records do not contain the information, the uncompressed cache says so on 80 percent of them; after self-study, the refined RAPID++ cache still does on 68 percent, while the cache that skipped refinement does on 40, Attention Matching on 36, and the cartridge on 16. We hypothesize that the per-layer distillation of RAPID lets the end-to-end stage correct an already good cache rather than fit the generated queries (Appendix I.3).

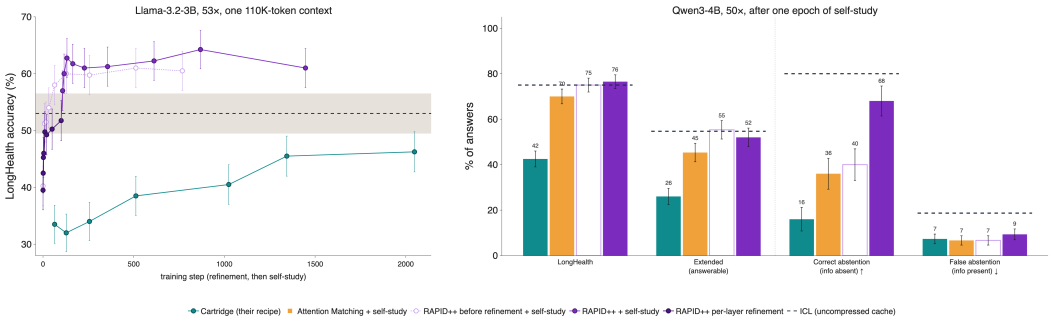


Figure 16: Left: Llama-3.2-3B on one 110K-token context of 10 patients at $53\times$: LongHealth accuracy (%, 200 answers per point) during 100 steps of per-layer refinement, then self-study on the same cache (drawn at $100+$ its own step; one epoch is 1,347 steps); the hollow curve runs self-study directly on the allocated cache, without refinement. Right: Qwen3-4B at $50\times$, patients 1–5, after one epoch of self-study (696 steps; the cartridge run stopped at step 448): accuracy on LongHealth (200 answers) and on the extended questions (50 answers per category, 150 for the average); the share of unanswerable questions answered with "the records do not contain enough information", the right answer there (50 answers, higher is better); and the share of answerable questions answered that way (150 answers, lower is better). In both panels the cartridge is trained with the recipe of Cartridges from its own initialization, and the other caches continue with the same self-study data. Dashed lines and ticks are the uncompressed cache; error bars and bands are ± 1 binomial standard error.

Cartridges showed that end-to-end training on self-study data can bring a compact cache close to the full context. Started from a strong per-layer initialization, it can even exceed the uncompressed cache when the context is long enough to degrade the model's own use of it (Figure 16, left): on Llama-3.2-3B with a 110K-token context, where the uncompressed cache answers 53 percent, 16 self-study steps (1.2 percent of an epoch) take the refined RAPID++ cache from 51.8 to 60.0 percent, while the cartridge reaches only 46 percent after 2,048 steps. The same holds at $106\times$; the $106\times$ curves, the extended questions on Llama-3.2-3B, and the Qwen3-4B training curves are in Appendix I.

We leave the analysis of this mechanism to future work. We hypothesize that a short end-to-end stage aligns the layers with one another, correcting the error that compounds from layer to layer, which independent per-layer training with teacher-forced inputs cannot see. A corrective, on-policy variant of our refinement, in which each layer is trained on the inputs the compressed layers below it produce, might recover part of this gain without additional data.