READ WHAT MATTERS: QUERY-ADAPTIVE QUANTIZATION FOR KV CACHES

Anonymous authors
Paper under double-blind review

ABSTRACT

KV-cache entries are stored before their future queries are known, but each decoding query needs precision in different places. We study this mismatch using separate budgets for retained bits and bits fetched per query. READKV stores each key and value in a progressive code whose prefixes support different reconstruction precisions. For each query, it allocates key-channel prefixes using the query, computes attention from the reconstructed keys, and then allocates value-token prefixes using that attention. Stored entries remain unchanged. Each stage optimizes a calibrated distortion objective under a fixed budget; we prove exact allocation under diminishing refinement gains and relate these objectives to attention-output error. We also exhibit a finite-dimensional attention family where query-dependent access strictly outperforms every query-independent reader at the same read budget, even with unrestricted competing encoders and decoders.

Across six base models, reading four bits on average from an eight-bit cache increases C4 perplexity by at most 0.66%, using about one quarter of the logical reads and half the retained capacity of a 16-bit cache. It is consistently more accurate than storing and fully reading four bits at the same payload-read budget. Long-context question answering and retrieval on two instruction-tuned models provide additional quality evidence. On the tested 8K-token, batch-one, single-layer workload on an NVIDIA A10G, a restricted eight-bit READKV reader with a two-bit mean payload-read budget has 39% lower latency than the tested TurboQuant codec.

1 INTRODUCTION

Long-context autoregressive language models maintain a key–value (KV) cache to avoid recomputing representations of earlier tokens. During decoding, each new query accesses this cache, and repeated reads can incur substantial memory traffic as the context grows. However, a query may depend primarily on a subset of key dimensions, and its attention may concentrate on a small fraction of value tokens. These observations motivate query-dependent reads that fetch the cache information needed at each decoding step.

We propose READKV, which separates stored bits per scalar, W, from average payload bits read per scalar, R. Its *progressive representation* lets each query choose how much precision to fetch while preserving unread detail for later queries. This design is motivated by the write-once, read-many structure of the KV cache. Retaining additional precision increases storage and insertion cost, but does not require every later query to fetch those extra bits. For an 8,192-token prompt followed by 128 decoding steps, storing eight bits while reading four, rather than storing and reading four, adds write traffic equal to just 0.79% of cumulative reads in the payload-only model of Appendix A. Stored precision and read volume therefore impose distinct costs. Device latency also depends on selection, reconstruction, and temporary memory traffic.

Prior work addresses both stored precision and selective access. KIVI and TurboQuant construct compact low-bit representations (Liu et al., 2024; Zandieh et al., 2026); KIVI's asymmetry concerns the quantization of keys versus values. QAQ, MiKV, and ZipCache use importance estimates to vary the precision retained in the cache (Dong et al., 2024; Yang et al., 2024; He et al., 2024). SparQ and Quest use the current query to select features, tokens, or pages to fetch (Ribar et al., 2024; Tang et al.,

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

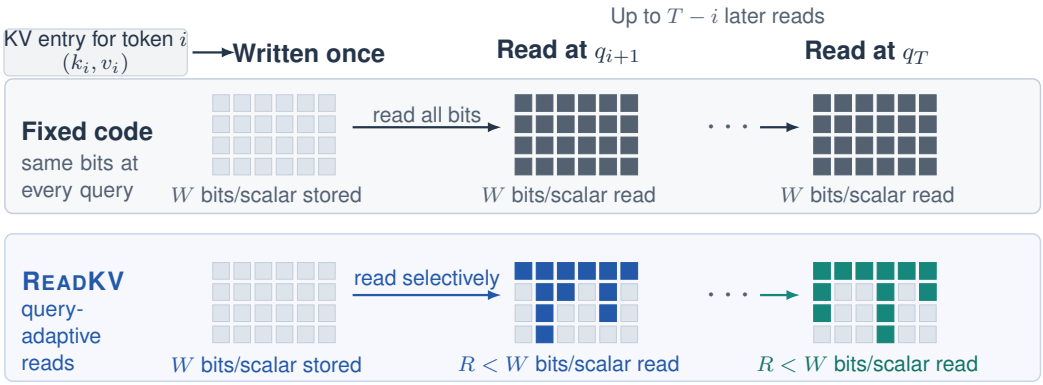


Figure 1: Write-once, read-many KV-cache access. Token i's key and value vectors are stored once and may be read by up to $T - i$ later queries. Fixed-precision quantization fetches the same full W-bit code each time. READKV stores a progressive W-bit code and lets each query fetch different prefixes under an average read budget $R < W$; unread refinements remain available.

2024). READKV makes the precision fetched from a stored scalar a query-time decision: a longer prefix gives a finer reconstruction, while unread refinements remain available to later queries.

The closest precedent is Xie et al. (2025), who combine precision tiers for ranked KV pages with selective retrieval of floating-point bit planes. Their work already separates retained precision from fetched precision. READKV develops this principle with a nested nonuniform scalar code whose every prefix depth is decodable, and an explicit read budget allocated first across key channels using the query, then across value tokens using reconstructed attention. The contribution is this representation and allocation rule, together with an analysis of what query-dependent reading can achieve under separate storage and read constraints. Appendix B gives the broader comparison.

We formalize this distinction through separate storage and read budgets, as illustrated in Figure 1. We focus on the partial-read regime $R < W$, where access decisions determine which information is fetched; $R = W$ permits full-record decoding. Our main contributions are as follows.

- We formulate KV-cache compression with separate storage and read budgets. A cache entry is encoded before its future queries are known, and each query may fetch only a budgeted subset of the stored bits. This distinguishes retained capacity from transferred payload and enables direct comparisons between query-independent and query-dependent access.
- We introduce READKV (Algorithm 1), which stores progressive key and value codes. The current query selects key prefixes; attention computed from their reconstructions selects value prefixes. Different queries can therefore use different precisions from the same stored entries without rewriting them.
- We analyze why query-dependent access can help and how READKV allocates its read budget (Section 4). On two finite-dimensional families, READKV achieves strictly lower worst-case key-score and attention-output MSE, respectively, than every query-independent branching reader with the same read budget. We then analyze READKV's allocation rule, which repeatedly assigns the next prefix bit to the key channel or value token for which it gives the largest predicted error reduction. We show when this rule gives the optimal allocation for each stage and relate the key and value error measures it optimizes to the final attention-output error. Together, these results explain both why query-dependent reading can help and how READKV uses its read budget.
- Across six base models, READKV $(W = 8, R = 4)$ stays within 0.66% of uncompressed perplexity and has lower perplexity than fully reading a four-bit cache at the same payload-read budget. Question answering and retrieval on two instruction-tuned models show further quality–read tradeoffs. In a separate single-layer benchmark on an NVIDIA A10G, READKV $(W = 8, R = 2)$ has 39% lower latency and 62–65% fewer DRAM read bytes than a fused quantized codec.

2 PARTIAL-READ CACHE MODEL

A cached key must be stored before the queries that will later use it are known. At each decoding step, the reader sees a new query and must decide which parts of that stored key to fetch. We ask how

accurately it can recover the query–key score when it may read fewer bits than the encoder retained. The following model separates these two budgets without committing to a particular quantizer.

Let $x \in \mathbb{R}^d$ be the vector to store and $y \in \mathbb{R}^d$ a later query. The target is the inner product $x^\top y$. There are three roles: the *encoder* observes x, but not y, and writes a bit string Z; the *reader* receives y and requests selected bits of Z; and the *decoder* uses y and the returned bits to estimate $x^\top y$. The reader has no other access to x. The encoder, reader, and decoder may be randomized, with all random coins independent of the fixed input pair (x, y).

We assign separate rates to storage and reading. For $W, R \geq 0$, the encoder stores at most $w = \lfloor Wd \rfloor$ bits, and each query may fetch at most $r = \lfloor Rd \rfloor$ bits. For example, $W = 8$ and $R = 2$ allow an $8d$-bit record but only $2d$ fetched bits per query. These are global budgets rather than per-coordinate quotas. The encoder may mix information across coordinates, and the reader may spend more probes on some coordinates than others. In the attention interpretation, x is a cached key, y is the current query, and $x^\top y$ is their unscaled attention score. This single-score model isolates the access question. Section 3 applies the same storage–read distinction to the full KV cache and its attention output.

We count the bits stored for each record and the bits fetched for each query. Computation is unrestricted, as in the one-bit-cell version of the static table-probe model (Yao, 1981). Record-specific metadata counts toward the corresponding budget. Code descriptions and other state fixed independently of (x, y) are shared across records rather than charged once per record. Appendix A accounts separately for their retained memory and reload traffic in the implementation.

To fix the scale of key-score error, we use $\|x\|_2, \|y\|_2 \leq 1$. We measure worst-case expected squared error over the stated input family, averaging over protocol randomness. Appendix C defines the full-unit-ball minimax risk, access hierarchy, and boundary cases.

We distinguish readers by the information they may use to select which stored bits to fetch. An *address* identifies one bit of the record Z, while a *batch* is a set of addresses chosen before any bit in that set is returned. The reader classes below differ in whether these choices may depend on the current query, previously returned bits, or both.

Table 1 orders five reader classes by the information available when addresses are selected. A query-oblivious fixed pattern uses neither the query nor returned bits. A query-independent branching reader may use each returned bit to choose its next address, but its access path and stopping rule cannot use y. Since the returned bits encode x, this path may still depend indirectly on the stored vector. A $\mathsf{staged}(1)$ reader uses y to choose all addresses in one batch. More generally, a $\mathsf{staged}(J)$ reader, for $J \geq 2$, uses at most J batches and may choose each later batch using y and the values returned by earlier batches. A fully branching reader may select each new address after observing every previous return. Every class fetches at most r bits, and its final decoder may use y.

Table 1: Reader classes distinguished by whether address selection may use the current query or previously returned bits. All classes use the same storage and read budgets, and every final decoder may use the query.

Reader Class	Uses Query?	Use of Returned Bits in Address Selection
Query-oblivious fixed pattern	No	None; every address is fixed in advance
Query-independent branching	No	Each new address may use all previously returned bits
$\mathsf{staged}(1)$: one batch	Yes	None; the complete batch is selected before reading
$\mathsf{staged}(J)$, $J \geq 2$: at most J batches	Yes	A later batch may use bits returned by earlier batches
Fully branching	Yes	Each new address may use all previously returned bits

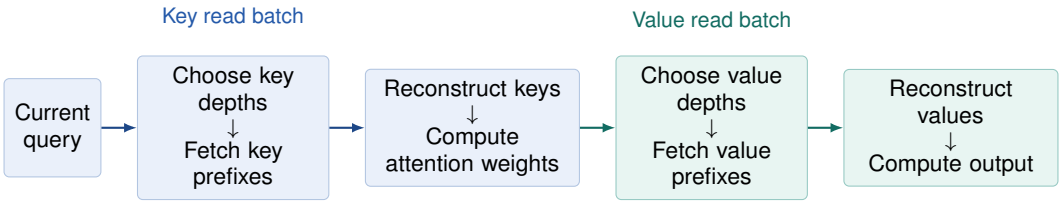
Section 3 develops READKV as a $\mathsf{staged}(2)$ reader: the query selects key prefixes in the first batch, and attention computed from the reconstructed keys selects value prefixes in the second. Section 4 analyzes its allocation rule and gives separations on specified key-score and attention families.

3 QUERY-ADAPTIVE CACHE DESIGN

This section defines the READKV encoder, its $\mathsf{staged}(2)$ reader, and decoder (Algorithm 1). The encoder stores each key and value in a progressive code whose longer prefixes have lower calibrated distortion. At query time, the reader issues two read batches from this unchanged cache. The current query determines the key prefixes fetched in the first batch. An intermediate decoder reconstructs the keys and computes attention, which determines the value prefixes fetched in the second batch.

The final decoder reconstructs the values and returns the attention output. Figure 2 summarizes this query-time flow.

Figure 2: Query-time flow of READKV. The query selects key-prefix depths in the first batch. After the intermediate decoder reconstructs the keys, the resulting attention weights select value-prefix depths in the second batch. Both batches read from the same unchanged progressive W-bit cache, and each batch's addresses are fixed before any of its bits are returned.



3.1 PROGRESSIVE CACHE REPRESENTATION

A conventional fixed-rate scalar code is designed to be decoded as a whole. Instead, READKV stores a path in a binary quantization tree. If the path has eight bits, its first two bits already specify a reconstruction; reading a third bit selects a finer reconstruction, and so on up to eight. Thus a later query can read a different prefix of the same unchanged code. We denote the stored depth by the integer W and use a Gaussian Lloyd tree (Lloyd, 1982); Appendix D gives its construction.

Consider one attention layer with T cached tokens and vector dimension d_h. It has G physical KV heads: each is one cached stream of keys $k_{g,j} \in \mathbb{R}^{d_h}$ and values $v_{g,j} \in \mathbb{R}^{d_h}$, indexed by head g and token j. The layer also has H query heads, with $q_h \in \mathbb{R}^{d_h}$. Several query heads may share the same KV stream. We write $g(h)$ for the KV head used by q_h and $\mathcal{S}(g) = \{h : g(h) = g\}$ for the query heads sharing it. A fetched cache bit is counted once even when several query heads share it, so its importance combines their contributions.

Before encoding, we choose fixed orthogonal Hadamard–sign transforms $U_K, U_V \in \mathbb{R}^{d_h \times d_h}$. The signs are drawn once before encoding, and the same transforms are reused for all cache entries and queries. These transforms mix coordinates before scalar quantization. Applying U_K to both a key and its query preserves their dot product; U_V is inverted after computing the value-weighted output. A channel below means one coordinate in these transformed vectors. For each transformed scalar x', the encoder uses a calibrated channel mean μ and scale σ to form $z = (x' - \mu)/\sigma$, and stores its W-bit tree path. A decoded prefix gives $\widehat{z}$ and hence $\mu + \sigma\widehat{z}$. Reading zero bits gives $\widehat{z} = 0$, the calibrated mean. The transforms and calibration remain fixed as queries change.

3.2 TWO-STAGE QUERY-ADAPTIVE READING

Key-Channel Read. A query weights key channels through a dot product. A reconstruction error in a channel with a larger query coefficient has a larger effect on the score. Write $q'_h = U_K q_h$, and let $\sigma^K_{g,c}$ be the calibrated scale of transformed key channel c in KV head g. Squaring the coefficient of a standardized channel error and adding contributions from the sharing query heads gives the importance

$$a^K_{g,c} = \sum_{h \in \mathcal{S}(g)} (q'_{h,c} \sigma^K_{g,c})^2. \tag{1}$$

The reader assigns a prefix depth $t^K_{g,c} \in \{0, \dots, W\}$ to each head–channel pair. That depth is used across all T cached tokens, because the same query coefficient multiplies that channel in every key. All key-prefix addresses are chosen from the query and fixed calibration before any key bits are fetched.

The intermediate decoder reconstructs $\widehat{K}_{g,j} \in \mathbb{R}^{d_h}$ from the returned prefixes. It then forms the attention vector $\widehat{\alpha}_h = (\widehat{\alpha}_{h,1}, \dots, \widehat{\alpha}_{h,T}) \in \mathbb{R}^T$ by applying softmax across tokens to the reconstructed logits $q'^{\top}_h \widehat{K}_{g(h),j}/\sqrt{d_h}$. These probabilities determine the second read batch; neither the decoder nor the reader uses full-precision keys or their unread refinements.

Value-Token Read. The attention output is a weighted sum of values. An error in a value token with larger attention weight therefore matters more to that output. READKV uses the squared reconstructed

probabilities to form

$$a^V_{g,j}(\widehat{\alpha}) = s_g \sum_{h \in \mathcal{S}(g)} \widehat{\alpha}^2_{h,j}. \tag{2}$$

Here $s_g \geq 0$ sets the value scale for a physical head. The reported experiments use $s_g = 1$, measuring value distortion in standardized units; Appendix D describes the physical-scale alternative. The reader chooses one depth $t^V_{g,j} \in \{0, \ldots, W\}$ per value token, shared across its d_h channels. All value-prefix addresses are fixed before this second batch is fetched. The final decoder reconstructs $\widehat{V}_{g,j} \in \mathbb{R}^{d_h}$ from the returned prefixes and returns $U_V^\top \sum_j \widehat{\alpha}_{h,j} \widehat{V}_{g(h),j}$ for query head h. There is one adaptive transition between the batches: returned key bits change attention, which changes the value read.

3.3 Prefix Allocation Under a Read Budget

An importance score alone does not specify a depth: an extra bit can help a coarse reconstruction more than an already precise one. Let $D(t)$ be the stored, calibrated distortion curve predicting standardized scalar mean-squared reconstruction error after t bits. For importance weights a_i, the allocator uses the objective $\sum_i a_i D(t_i)$. Each term combines how much an item's error matters with how much error its chosen prefix leaves. These separable objectives omit cross-terms between reconstruction errors; they do not assume those interactions vanish in real caches.

The two stages have fixed nonnegative integer budgets B_K and B_V. These count group refinements, not individual payload bits: increasing one key-channel depth by one fetches T bits, and increasing one value-token depth by one fetches d_h bits. Within either stage, with budget B, the allocation problem is

$$\min_{\substack{t_i \in \{0,\ldots,W\} \\ \sum_i t_i \leq B}} \sum_i a_i D(t_i).$$

Define $\mathrm{ChooseDepths}(a, D, W, B)$ by starting at $t_i = 0$ and repeatedly increasing an eligible depth $t_i < W$ with the largest positive weighted gain $a_i[D(t_i) - D(t_i + 1)]$. Stop after B increments or when no positive gain remains, breaking ties in a fixed order. Section 4 analyzes this rule, including its exact stagewise optimality under diminishing refinement gains and its relation to attention-output error.

The key stage calls this rule with (a^K, B_K), and the value stage with $(a^V(\widehat{\alpha}), B_V)$. Their total payload is at most $TB_K + d_h B_V$ bits. Writing the budgeted mean depths as $R_K = B_K/(Gd_h)$ and $R_V = B_V/(GT)$ gives the mean payload read budget

$$R = \frac{TB_K + d_h B_V}{2GTd_h} = \frac{1}{2}(R_K + R_V).$$

The K/V split is an input to the allocator; it optimizes the allocation within each stage, not that split. Actual reads may be smaller if no positive refinement gains remain.

Algorithm 1 ReadKV encoder, two-stage reader, and decoder; full pseudocode in Algorithm 2.

1. **Encoder.** Transform and standardize each new key and value, then store every scalar as a depth-W progressive code.
2. **Key Reader.** Compute a^K from the current queries using Equation (1). Use $t^K = \mathrm{ChooseDepths}(a^K, D, W, B_K)$ and fetch the selected key prefixes.
3. **Intermediate Decoder.** Reconstruct the selected keys, then compute the logits and attention probabilities $\widehat{\alpha}$.
4. **Value Reader.** Evaluate $a^V(\widehat{\alpha})$ using Equation (2). Use $t^V = \mathrm{ChooseDepths}(a^V(\widehat{\alpha}), D, W, B_V)$ and fetch the selected value prefixes.
5. **Final Decoder.** Reconstruct the selected values and return $U_V^\top \sum_j \widehat{\alpha}_{h,j} \widehat{V}_{g(h),j}$ for each query head h.

Algorithm 1 summarizes the procedure. With N items, a max-heap implementation uses $O(N)$ memory and $O(N + B \log N)$ time; the key and value stages have $N = Gd_h$ and $N = GT$, respectively. Appendix F.1 specifies integer-budget conversion and tie handling.

4 Performance Guarantees for Partial Reads

In this section, we establish two query-dependent separations under a read budget, then relate ReadKV's allocation objectives to attention error for a fixed cache and query. Appendix E gives formal statements and proofs.

Query-Dependent Separations. Theorem 5 isolates key reading. Its queries have at most k active coordinates in a shared transformed basis. READKV reads those coordinates to the full stored depth. At $(W, R) = (8, 2)$, $d = 128$, and $k = 32$, its worst-case key-score MSE is smaller by a certified factor exceeding 4.85 than that of any query-independent branching reader, even with an arbitrary randomized encoder and decoder. This isolates the value of query-dependent access: the gain does not rely on restricting the competing encoder, decoder, or payload-dependent branching. The proof combines the code's pointwise error with an information limit: after fixing the random coins, r probes yield at most 2^r transcripts.

Theorem 6 gives the main separation for the complete key-then-value procedure. It constructs a two-token cache with independently varying values, where the current query determines which token receives nearly all the attention. The first batch reads one key bit from each token and identifies that token exactly. Because its attention weight is much larger, every refinement of its value has greater weighted gain than the other value's first bit. The second batch therefore reads the selected value to depth eight and leaves the other unread.

A query-independent transcript instead contains at most r bits about both values, yet its decoder must support either query. Averaging the two query choices and applying the same entropy bound on reconstruction error gives the lower bound. At head dimension $m = 128$, READKV stores $W = 8$ bits per scalar and reads $R \approx 2.004$ payload bits per scalar; its worst-case per-coordinate attention-output MSE is smaller by a certified factor exceeding 9.52. This result shows that the advantage survives the complete causal path from key reconstruction through attention to value reconstruction, rather than appearing only in key-score estimation. This is a restricted-family separation from query-independent access, not a full-domain dominance claim.

Allocation and Attention-Error Certificate. Theorem 13 asks what the allocator guarantees beyond the constructed separation families. When the scalar refinement gains are nonnegative and nonincreasing, the marginal-gain rule exactly minimizes the calibrated key objective for its supplied budget. After the reconstructed keys fix $\widehat{\alpha}_{\text{READKV}}$, the same rule exactly minimizes the value objective for those probabilities and its supplied budget. The deployed depth-eight code satisfies this diminishing-gain condition.

When its conversion weights are finite, the theorem bounds the total squared attention-output error across query heads by a weighted sum of these two optimal objective values. The explicit weights account for calibration mismatch, correlations among reconstruction residuals, and the scale of the centered value matrix. This connects the tractable objectives optimized by READKV to its final output and shows when small objective values imply small attention error. The result assumes no independence among queries, keys, values, or residuals.

Our proof strategy has three steps. First, diminishing gains give each key channel or value token a nonincreasing list of improvements from successive prefix bits. A legal read must take a prefix of that list, so selecting the largest available gain gives the optimal integer allocation for each stage. Second, the output error splits into changes in attention weights caused by key reconstruction and value reconstruction error under those weights. Softmax stability bounds the first term. Third, eigenvalue bounds for residual correlation matrices control the cross terms omitted by the calibrated objectives.

The guarantee is stagewise: key allocation is optimized first, then value allocation for the resulting attention. It does not jointly optimize true attention error. Its coefficients require the full-precision cache and reconstructed residuals, so they certify completed reconstructions offline. Section 5 tests model quality in practice.

5 LANGUAGE-MODEL EVALUATION

We test whether READKV reduces KV-cache reads while preserving quality on C4 language modeling (Raffel et al., 2020), LongBench question answering (Bai et al., 2024), and passkey retrieval (Mohtashami & Jaggi, 2023). We use three nominal storage/read settings: $(4, 2)$ uses a smaller cache and low read budget; $(8, 2)$ retains more detail at the same read budget; $(8, 4)$ raises the read budget at fixed capacity. The flexible reader's integer K/V split is selected on development data for each model and rate, then frozen before test. The models in Tables 2 and 3 cover Qwen2.5 (Yang et al., 2025), Yi-1.5 (01.AI et al., 2024), DeepSeek-LLM (DeepSeek-AI et al., 2024), and Mistral (Jiang et al., 2023). Appendix F.1 specifies precision, transform seeds, hardware and calibration cohorts, and

uncertainty estimation. Single-layer GPU latency and physical-traffic measurements use an NVIDIA A10G and the restricted reader described in Section 5; these are separate from the flexible-reader quality experiments.

Baseline Settings. DENSE uses an uncompressed 16-bit KV cache and supplies the quality and resource reference. Our internal FULL READER $(4, 4)$ stores and always reads four bits per scalar from READKV's nested code. KIVI (2-bit K/V) (Liu et al., 2024) uses two payload bits per key and value scalar, plus groupwise scales and offsets and a 128-token uncompressed residual buffer. TurboQuant K4/V4 (Zandieh et al., 2026) uses four-bit key and value codes, with the first and last two layers uncompressed. We give READKV the same exemptions in rows marked †; their (W, R) budgets apply to compressed layers. SparQ $r32/k128$ (Ribar et al., 2024) retains a 16-bit cache, probes the history using $r = 32$ query-selected key channels, and reads at most $k = 128$ selected past positions. Appendix F.2 gives implementation and configuration-selection details.

Reading the Results. For each model and dataset, we report the percentage perplexity increase $\Delta\text{PPL} = 100(\text{PPL}_{\text{method}}/\text{PPL}_{\text{DENSE}} - 1)$ or the F1-point loss $\Delta\text{F1} = \text{F1}_{\text{DENSE}} - \text{F1}_{\text{method}}$. Positive values are worse; negative values are sample improvements. Logical reads and retained KV capacity are percentages of DENSE. Logical reads include payload, metadata, probes, rereads, and dense layers; they exclude model weights, cache writes, and workspace, and are not physical DRAM traffic. Entries are encoded once and read repeatedly: extra retained refinements can improve quality without increasing reads, although their capacity still limits context length and concurrency.

C4 Language Modeling. Each model is evaluated on 32 held-out 2,048-token documents, with separate calibration and development data. Tokenizers differ, so comparisons are within each model.

Table 2: C4: 32 held-out 2,048-token documents per model; three transform seeds where available. READKV $(8, 4)$ stays within 0.7% of DENSE on all six models, using about a quarter of its logical reads and half its KV storage. †: first/last two layers dense. Color marks the observed quality–read frontier: at shown precision, no other method row in the column improves loss or reads without worsening the other. Reads recur per query; retained capacity is shown but not ranked.

Method (W, R)	Qwen2.5 3B	Qwen2.5 7B	Qwen2.5 14B	Yi-1.5 6B	DeepSeek LLM-7B	Mistral 7B-v0.3
DENSE PPL	11.36	10.13	8.74	8.96	8.49	7.62
Top: ΔPPL (%) Color: quality–read frontier						
Bottom: logical reads / retained KV capacity (% of DENSE)						
FULL READER $(4, 4)$	+1.59 25.6 / 25.1	+1.34 25.4 / 25.1	+1.68 25.3 / 25.1	+3.02 25.4 / 25.1	+3.44 25.2 / 25.1	+2.17 25.3 / 25.1
KIVI (2-bit K/V) Liu et al., 2024	+4.84 26.2 / 23.8	+2.79 26.2 / 23.8	+2.21 26.2 / 23.8	+2.82 26.2 / 23.8	+3.54 26.2 / 23.8	+2.11 26.2 / 23.8
SparQ $r32/k128$ Ribar et al., 2024	+1.27 24.7 / 100.0	+1.44 24.7 / 100.0	+0.81 24.7 / 100.0	+92.44 24.7 / 100.0	+0.86 24.7 / 100.0	+53.95 24.7 / 100.0
READKV $(4, 2)$	+4.44 13.1 / 25.1	+3.41 12.9 / 25.1	+3.75 12.8 / 25.1	+12.13 12.9 / 25.1	+9.74 12.7 / 25.1	+11.00 12.8 / 25.1
READKV $(8, 2)$	+3.85 13.1 / 50.1	+2.99 12.9 / 50.1	+2.94 12.8 / 50.1	+10.16 12.9 / 50.1	+6.27 12.7 / 50.1	+11.24 12.8 / 50.1
READKV $(8, 4)$	+0.19 25.6 / 50.1	+0.39 25.4 / 50.1	+0.36 25.3 / 50.1	+0.20 25.4 / 50.1	+0.66 25.2 / 50.1	+0.49 25.3 / 50.1
TurboQuant K4/V4† Zandieh et al., 2026	+1.74 34.4 / 34.4	+1.02 36.7 / 36.7	+0.72 32.3 / 32.3	+0.99 35.4 / 35.4	+0.45 36.0 / 36.0	+0.45 35.4 / 35.4
READKV $(8, 4)^{\dagger}$	+0.14 33.9 / 55.6	+0.09 36.0 / 57.2	+0.13 31.5 / 54.3	+0.22 34.7 / 56.3	+0.11 35.2 / 56.8	+0.11 34.6 / 56.3

With the first and last two layers uncompressed, READKV $(8, 4)^{\dagger}$ achieves lower perplexity and slightly fewer logical reads than TurboQuant K4/V4 on all six models, while retaining more KV storage. Figure 3(a) displays this comparison with the same layer exemptions. Without exemptions, READKV $(8, 4)$ stays within 0.66% of DENSE; it improves perplexity over the evaluated SparQ configuration on all six models and retains roughly half the capacity, with a slightly higher read rate. It also improves on the FULL READER's 1.34–3.44% PPL increase at the same four-bit payload-read budget, illustrating the value of retained refinements.

The $R = 2$ settings reduce reads further but are less consistent across families: $(8, 2)$ stays within 3.9% PPL increase on Qwen, versus 6.27–11.24% on Yi, DeepSeek, and Mistral. The smaller $(4, 2)$ cache retains the FULL READER's quarter-capacity footprint with about 49% fewer logical reads. Query-dependent allocation also matters: in a paired Qwen3B control with the same code and K/V split, calibration-only key allocation raises PPL from 11.789 to 12.011; 31 of 32 documents favor current-query allocation (Appendix F.5). Appendices F.3, F.6, and F.7 give additional baselines, intermediate prefix-depth controls, and the surrogate/error comparison.

LongBench Question Answering. We evaluate Qwen2.5-7B-Instruct and Mistral-7B-Instruct-v0.3 on 100 questions per dataset from Qasper, HotpotQA, and 2WikiMultihopQA, using the same identifiers at 3,500- and 7,500-token prompt limits. All methods use DENSE prompt prefill, which selects the first answer token; compression affects generation from the second token.

Table 3: LongBench QA at a 7,500-token prompt limit; 100 questions per condition. READKV $(8, 2)$ loses at most 1.46 F1 points across the six conditions while using about 13% of DENSE logical reads. †: first/last two layers dense. Color marks the observed quality–read frontier: at shown precision, no other method row in the column improves loss or reads without worsening the other. Reads recur per query; retained capacity is shown but not ranked.

	Qwen2.5-7B-Instruct			Mistral-7B-Instruct-v0.3		
Method (W, R)	Qasper	HotpotQA	2Wiki MultihopQA	Qasper	HotpotQA	2Wiki MultihopQA
DENSE F1	41.13	48.34	45.30	37.41	43.37	42.71
Top: ΔF1 (points) Color: quality–read frontier Bottom: logical reads / retained KV capacity (% of DENSE)						
KIVI (2-bit K/V) Liu et al., 2024	+1.56 20.4 / 19.9	+0.08 20.0 / 19.9	+2.41 20.1 / 19.9	+0.49 20.3 / 19.9	+1.22 20.0 / 19.9	+1.80 20.0 / 19.9
TurboQuant K4/V4† Zandieh et al., 2026	-0.06 36.7 / 36.7	+0.42 36.7 / 36.7	-0.55 36.7 / 36.7	-0.79 35.4 / 35.4	+0.01 35.4 / 35.4	-0.31 35.4 / 35.4
SparQ $r32/k128$ Ribar et al., 2024	+1.40 15.3 / 100.0	-0.44 14.2 / 100.0	+0.69 14.5 / 100.0	+1.26 15.0 / 100.0	+2.65 14.3 / 100.0	+0.47 14.4 / 100.0
READKV $(4, 2)$	+1.68 12.6 / 25.0	-0.60 12.6 / 25.0	+5.43 12.6 / 25.0	+0.69 12.6 / 25.0	-0.42 12.5 / 25.0	+1.08 12.5 / 25.0
READKV $(8, 2)$	+1.46 12.6 / 50.0	-0.35 12.6 / 50.0	+1.09 12.6 / 50.0	-0.66 12.6 / 50.0	-0.23 12.5 / 50.0	+0.52 12.5 / 50.0
READKV $(8, 4)$	+0.98 25.1 / 50.0	+0.25 25.1 / 50.0	+0.29 25.1 / 50.0	+0.98 25.1 / 50.0	-0.44 25.0 / 50.0	+0.00 25.0 / 50.0
READKV $(8, 2)$†	+2.34 25.1 / 57.2	-0.12 25.0 / 57.2	+2.28 25.1 / 57.2	+0.58 23.5 / 56.3	-0.23 23.5 / 56.3	+0.87 23.5 / 56.3
READKV $(8, 4)$†	+1.01 35.8 / 57.2	-0.03 35.8 / 57.2	+0.96 35.8 / 57.2	+0.60 34.4 / 56.3	-0.67 34.4 / 56.3	+0.00 34.4 / 56.3

At the 7,500-token prompt limit, READKV $(8, 2)$ has F1 changes between -0.66 and 1.46 points. It uses 37–38% fewer logical reads than KIVI (2-bit K/V), 65–66% fewer than protected TurboQuant, and 12–18% fewer than SparQ. Figure 3(b) compares it with the two all-layer competitors: KIVI retains a smaller cache, whereas SparQ retains twice its capacity. READKV scores higher than KIVI in all six conditions; against SparQ it scores higher in two conditions and loses at most 0.41 F1 points in the other four. Every condition is shown.

Table 3 also shows how the read and storage budgets affect quality. The all-layer $(8, 4)$ setting has a largest loss of 0.98 F1 points. The quarter-capacity $(4, 2)$ setting stays within two points in five conditions, but loses 5.43 points on Qwen 2WikiMultihopQA; its Qwen hardware and calibration cohort differs from $(8, 2)$ (Appendix F.1). Across both prompt limits, all-layer $(8, 2)$ and $(8, 4)$ losses remain below 1.78 and 1.38 points, respectively. The protected QA rows do not show a uniform quality advantage over TurboQuant. Appendix F.3 gives the paired 3,500-token results and protected-row comparisons, including their cross-cohort qualification.

Passkey Retrieval. Appendix F.4 reports this long-context retention check; several methods recover every tested key, so it supplements the C4 and QA comparisons.

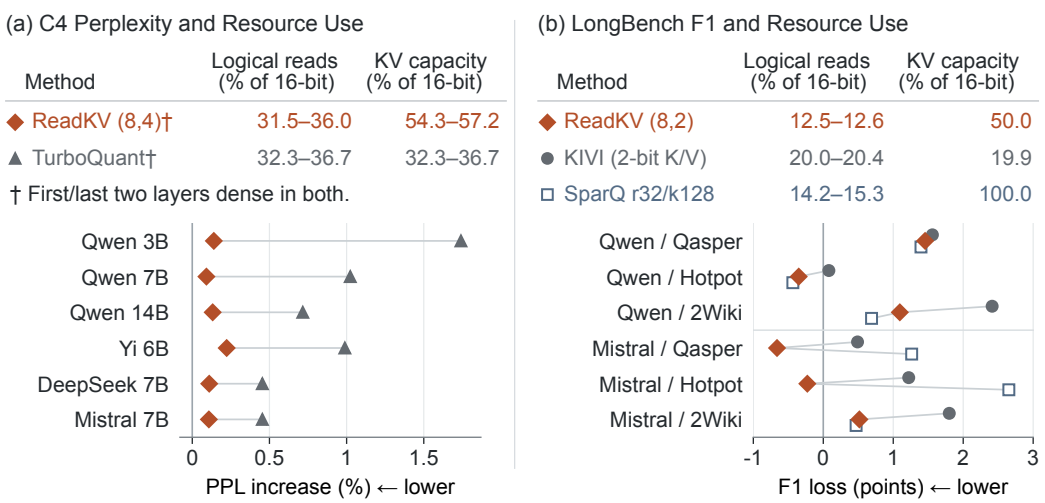


Figure 3: Comparison with representative baselines using Tables 2 and 3. **(a)** On C4, READKV $(8, 4)^{\dagger}$ has lower PPL and slightly fewer logical reads than TurboQuant K4/V4 on all six models when the first and last two layers remain dense, while retaining more KV capacity. **(b)** On LongBench at a 7,500-token prompt limit, READKV $(8, 2)$ uses fewer logical reads than KIVI and SparQ. It has higher F1 than KIVI in all six conditions and mixed results against SparQ. Resource values are percentages of the 16-bit cache; lower quality loss is better.

Logical and Physical Read Costs. To test whether lower logical reads reduce device cost, the timed reader uses the same progressive code with hardware-oriented constraints: per-head budgets, equal key and value rates, minimum prefix depth one, and two-token value packets. Appendix F.8 reports their quality effect and defines the full device policy.

On an NVIDIA A10G with 8,192 cached tokens and batch size one, the optimized READKV $(8, 2)$ kernel takes 0.093 ms under graph replay versus 0.154 ms for the TurboQuant codec and 0.051 ms for DENSE. Under the same launcher and timing boundary, this is 39% lower latency than the tested quantized codec, while DENSE is 45% lower than READKV. Across warm and cache-evicted conditions, profiling shows 62–65% fewer DRAM read bytes and 20–31% less total traffic than TurboQuant. These single-layer measurements start from an existing cache and isolate decode-time codec costs. Appendix G gives the full timing grid, measurement boundaries, and retained-cache, workspace, and physical-traffic accounting.

6 DISCUSSION

KV-cache entries are written once but read repeatedly under changing queries. This asymmetry motivates separating retained from fetched precision: a cache can preserve detail for future queries without moving all of it on every decode step. READKV implements this idea with a progressive code and a causal two-stage reader. The query allocates key precision, and reconstructed attention allocates value precision. We prove finite settings where this access strictly outperforms query-independent reading, characterize the stagewise allocator, and relate its objectives to attention-output error. Across six base models, READKV $(W = 8, R = 4)$ remains within 0.66% of uncompressed C4 perplexity and improves over fully reading a four-bit cache at the same payload-read budget. Question answering and retrieval provide further evidence of useful quality–read tradeoffs.

The theoretical separations concern explicit input families, and the attention certificate depends on calibration and residual-correlation factors. Finally, while the NVIDIA A10G study shows lower single-layer latency and DRAM traffic than the tested fused quantized codec, it does not yet establish an end-to-end serving advantage. Fusing allocation, reconstruction, and attention, while supporting more flexible K/V budgets, is the main path toward reducing this gap. Complete-model evaluation under long contexts and memory-limited batching can then measure the resulting serving benefit.

AI USE STATEMENT

Generative AI tools (ChatGPT/Codex) assisted with theoretical analysis and proof checking, experiment design and implementation, result interpretation, literature searches, and manuscript preparation. This included resource accounting, table generation and consistency checks. Automated checks compared reported summaries with saved experimental records. The authors are responsible for the final text, claims, code and other artifacts.

REPRODUCIBILITY STATEMENT

We provide separate reproduction paths for the theoretical and empirical results. Appendices C, D and E state the formal protocol, full READKV pseudocode, deployed codec tables, assumptions and complete proofs. Appendix A gives the storage and logical-read accounting used in the experiments. Appendix F.1 specifies the data splits, calibration and development procedure, decoding settings, random seeds, numerical precision, metrics, uncertainty estimates and hardware provenance. Tables 2 and 3 report the principal comparisons; Appendix F adds controls and further evaluation conditions. Appendix G defines the device reader, benchmark scope, timing boundary, cache conditions, workspace and DRAM-counter measurements.

REFERENCES

01.AI, Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Heng Li, Jiangcheng Zhu, Jianqun Chen, Jing Chang, et al. Yi: Open foundation models by 01.AI. *arXiv preprint arXiv:2403.04652*, 2024. URL https://arxiv.org/abs/2403.04652.

Noga Alon and Bo'az Klartag. Optimal compression of approximate inner products and dimension reduction. *arXiv preprint arXiv:1610.00239*, 2016. URL https://arxiv.org/abs/1610.00239.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://arxiv.org/abs/2308.14508.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 2 edition, 2006. doi: 10.1002/047174882X. URL https://doi.org/10.1002/047174882X.

Ronald de Wolf. Error-correcting data structures. *arXiv preprint arXiv:0802.1471*, 2008. URL https://arxiv.org/abs/0802.1471.

DeepSeek-AI, Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. DeepSeek LLM: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024. URL https://arxiv.org/abs/2401.02954.

Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. QAQ: Quality adaptive quantization for LLM KV cache. *arXiv preprint arXiv:2403.04643*, 2024. URL https://arxiv.org/abs/2403.04643.

William H. R. Equitz and Thomas M. Cover. Successive refinement of information. *IEEE Transactions on Information Theory*, 37(2):269–275, 1991.

Bennett Fox. Discrete optimization via marginal analysis. *Management Science*, 13(3):210–216, 1966. doi: 10.1287/mnsc.13.3.210. URL https://pubsonline.informs.org/doi/10.1287/mnsc.13.3.210.

Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44 (6):2325–2383, 1998.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. ZipCache: Accurate and efficient KV cache quantization with salient token identification. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2405.14256.

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. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. URL https://arxiv.org/abs/2310.06825.

Tengxuan Liu, Shiyao Li, Jiayi Yang, Tianchen Zhao, Feng Zhou, Xiaohui Song, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. PM-KVQ: Progressive mixed-precision KV cache quantization for long-CoT LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2505.18610.

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. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 32332–32344. PMLR, 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Ali Makhdoumi, Shao-Lun Huang, Muriel Médard, and Yury Polyanskiy. On locally decodable source coding. *arXiv preprint arXiv:1308.5239*, 2013. URL https://arxiv.org/abs/1308.5239.

Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2305.16300.

Pranav Ajit Nair, Puranjay Datta, Jeff Dean, Prateek Jain, and Aditya Kusupati. Matryoshka quantization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 45484–45506. PMLR, 2025. URL https://proceedings.mlr.press/v267/nair25a.html.

Pravin Nair. Softmax is 1/2-lipschitz: A tight bound across all ℓ_p norms. *arXiv preprint arXiv:2510.23012*, 2025. URL https://arxiv.org/abs/2510.23012.

Yeonhong Park, Jake Hyun, Sanglyul Cho, Bonggeun Sim, and Jae W. Lee. Any-precision LLM: Low-cost deployment of multiple, different-sized LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 39682–39701. PMLR, 2024. URL https://proceedings.mlr.press/v235/park24e.html.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL https://jmlr.org/papers/v21/20-074.html.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ attention: Bandwidth-efficient LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 42558–42583. PMLR, 2024. URL https://proceedings.mlr.press/v235/ribar24a.html.

Bixio Rimoldi. Successive refinement of information: Characterization of the achievable rates. *IEEE Transactions on Information Theory*, 40(1):253–259, 1994. doi: 10.1109/18.272493. URL https://doi.org/10.1109/18.272493.

A. Segall. Bit allocation and encoding for vector sources. *IEEE Transactions on Information Theory*, 22(2):162–169, 1976. doi: 10.1109/TIT.1976.1055533. URL https://doi.org/10.1109/TIT.1976.1055533.

Y. Shoham and A. Gersho. Efficient bit allocation for an arbitrary set of quantizers. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(9):1445–1453, 1988. doi: 10.1109/29.90373. URL https://doi.org/10.1109/29.90373.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2406.02542.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2406.10774.

Rui Xie, Asad Ul Haq, Linsen Ma, Yunhua Fang, Zirak Burzin Engineer, Liu Liu, and Tong Zhang. Reimagining memory access for LLM inference: Compression-aware memory controller design. *arXiv preprint arXiv:2503.18869*, 2025. URL https://arxiv.org/abs/2503.18869.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. ThinK: Thinner key cache by query-driven pruning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/8edb116d5b288b6a9bba4c16ab647702-Abstract-Conference.html.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2025. URL https://arxiv.org/abs/2412.15115.

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. *arXiv preprint arXiv:2402.18096*, 2024. URL https://arxiv.org/abs/2402.18096.

Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, July 1981. doi: 10.1145/322261.322274.

Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. Turboquant: Online vector quantization with near-optimal distortion rate. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=tO3ASKZlok.

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

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. MixKVQ: Query-aware mixed-precision KV cache quantization for long-context reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7189–7204. Association for Computational Linguistics, July 2026b. doi: 10.18653/v1/2026.acl-long.326. URL https://aclanthology.org/2026.acl-long.326/.

APPENDIX ORGANIZATION

The appendices provide definitions, proofs, reproduction details, and additional experiments. The table maps each main technical topic to its supporting material.

Topic	Location
Storage and read accounting	Appendix A
Cache encoding and reading	Appendix D, especially D.1
Provable query-dependent advantage	Appendix E.1
Integer prefix-allocation optimality	Appendix E.2
Allocation-based attention-error bounds	Appendix E.4
Quality evaluation and mechanism controls	Appendix F
Additional C4 controls and shorter-context QA	Tables 5 and 6
GPU timing and traffic measurements	Appendix G

A STORAGE AND TRAFFIC ACCOUNTING

A cache entry occupies capacity while resident, incurs a write on insertion, and incurs reads when reused. Reuse can amortize the insertion cost, but does not reduce the retained capacity. We account for these costs separately.

A.1 TASK-TABLE COST ACCOUNTING

The quality tables keep the algorithm settings W, R in the method name. Their resource fields instead report percentages of DENSE, the matched uncompressed 16-bit KV representation, including each method's required auxiliary information and any full-precision portions. Figure 3 uses these same resource quantities. Weight precision and attention arithmetic are separate protocol choices; an FP32 attention reference does not imply a 32-bit accounted cache.

At a declared cache state t, let C_t be the method's accounted persistent KV-state bytes and C_t^0 those of DENSE. Then

$$\text{Cache}\%(t) = 100\frac{C_t}{C_t^0}.$$

The numerator includes required retained supporting information and copies; any device-only subtotal is identified separately from host state. It excludes model weights and temporary workspace: "modeled cache" means the persistent KV representation, not a model's total allocated memory. For matched calls, let I_t and I_t^0 be the logical cache information requested, in bits. Reading cost is

$$\text{KV reads}\% = 100\frac{\sum_t I_t}{\sum_t I_t^0}.$$

Logical reads count payload, probes, rereads and fetched supporting state; the denominator covers all original KV scalars that DENSE would inspect at those states, including entries skipped by a selective method. A physical KV head is counted once in the underlying cache even when several query heads share it.

The task tables report capacity at a fixed reference state: 2,048 cached tokens for prediction, 3,500 or 7,500 for the corresponding QA panel, and 4,096 or 8,192 for passkey retrieval. The QA states are the prompt caps, not claims that every example has that length. This choice keeps cache capacity independent of answer length. Endpoint and trace-weighted occupancy views remain in the machine-readable accounting record; neither is a cumulative write count.

Logical reads use each method's recorded attention calls and normalize against DENSE at those same states. Under the dense-prefill generation protocol, a g-token answer after an n_0-token prompt causes $g-1$ compressed calls at cache lengths $n_0+1, \ldots, n_0+g-1$: DENSE prefill already chose the first answer token, and the final emitted token is not forwarded. These percentages compare information per recorded attention work; differing stopping lengths do not establish equal-work total byte savings.

Token-associated payload and side information are supplemented by an explicit conservative allowance for shared decoder data. Each layer call is charged one load of that layer's calibration arrays and the retained decoder table needed by the method. Tables shared across layers are counted once in persistent capacity but once per compressed-layer call in this read allowance. The accounting record separates these components. This full-load convention avoids assuming free hardware residency; it is an analytic allowance, not a measurement of device traffic or instruction-level loads. It is applied to the competing codecs as well as to ReadKV.

Per-token or per-layer metadata, residual buffers and protected layers are charged at their declared scope. Layout duplication is reported for the corresponding representation; packet-tail padding is included in logical reads where it is exported in the read counters. Shared calibration and codebooks are charged once per retained instance. A quality simulation's analytical count is not a measurement of GPU allocation or physical traffic. Hardware transaction rounding belongs to physical traffic, whereas complete packets requested by the algorithm belong to its logical count.

Neither task-table field includes temporary selection, sorting, reconstruction or attention buffers, even when scratch space is preallocated and reused. Their simultaneously live storage belongs in peak workspace; their accesses belong in physical traffic when they cross the measured memory boundary. Allocation schedules are produced by the reader rather than fetched from the retained cache, so they receive the same treatment. The task-table costs include fetched calibration and decoder data but exclude generated schedules. Including workspace accesses is necessary for a device-traffic or latency claim, but adding them to retained KV reads would change the quantity being compared. Actual peak memory must count simultaneously live KV state, workspace, weights and other allocations, rather than sum unrelated individual peaks.

A.2 Payload-Only Cache-Reuse Model

The following model explains the write/read example in the introduction. It counts an idealized sequence of decode calls; the preceding subsection gives the fuller accounting and dense-prefill boundary used by the task tables.

We count logical payload bits for one cached key or value scalar. The whole-cache count is obtained by multiplying by the number of key and value scalars stored per token. A physical KV head is counted once even when several query heads share it. Unlike the task-table accounting above, this model omits metadata and implementation overhead to isolate the cost of reuse.

Let $N \geq 0$ be the prompt length before decoding begins, and let $L \geq 1$ be the number of decode steps. At step $t \in \{0, \ldots, L-1\}$, the historical cache contains $N+t$ tokens. We charge one insertion for each prompt token and each generated token. This includes the token produced at the final decode step, even though it is not read again within the accounting horizon. The read count includes only accesses to historical cache entries during decoding. It excludes prefill attention, current-token self-attention, metadata, and other memory operations.

The total number of historical scalar reads is

$$S(N, L) = \sum_{t=0}^{L-1} (N + t) = LN + \frac{L(L-1)}{2}.$$

For a fixed W_0-bit codec and a progressive codec that stores W bits and reads R bits, the corresponding logical payload traffic is

$$T_{W_0}(N, L) = W_0(N + L) + W_0 S(N, L),$$
$$T_{W,R}(N, L) = W(N + L) + RS(N, L).$$

The first term in each expression is write traffic, and the second is read traffic. If keys and values use different depths, the formula is applied to each and the two counts are added. If an implementation omits the final insertion, $N + L$ is replaced by $N + L - 1$ in both write terms.

A.3 Equal Read Budgets

Suppose $R = W_0 < W$. The two codecs then fetch exactly the same number of historical payload bits. The progressive codec additionally writes $(W - W_0)(N + L)$ bits as entries are inserted. It

cannot reduce total logical traffic in this equal-read comparison, but the extra write traffic may be small relative to the reads accumulated over the sequence. When $S(N, L) > 0$, that fraction is

$$\eta(N, L) = \frac{(W - W_0)(N + L)}{W_0[LN + L(L-1)/2]}.$$

For $W = 8$, $W_0 = R = 4$, $N = 8192$, and $L = 128$, we obtain $\eta(N, L) = 0.0078735$. Thus, the added write traffic is approximately 0.79% of the common payload-read traffic. More generally, the extra-write fraction is at most $\varepsilon > 0$ exactly when

$$(W - W_0)(N + L) \le \varepsilon W_0[LN + L(L-1)/2].$$

For $N \gg L$, this fraction approaches $(W - W_0)/(W_0 L)$. If the sequence starts from an empty cache, so that $N = 0$ and $L > 1$, it equals $2(W - W_0)/[W_0(L-1)]$. These expressions show that repeated reuse, rather than prompt length by itself, amortizes the additional writes. For example, when $W = 8$ and $W_0 = 4$, 201 decode steps make the fraction at most one percent for every $N \ge 0$. This statement is uniform over the prompt length. For fixed L, the fraction decreases with N because the derivative of $(N + L)/[LN + L(L-1)/2]$ has numerator $-L(L+1)/2$. Its largest value therefore occurs at $N = 0$, where $2/(L-1) \le 0.01$ exactly when $L \ge 201$.

A.4 Reduced Read Budgets

Suppose $W > W_0$ and $R < W_0$. The savings from repeated reads can then offset the additional write traffic. Let $a = W - W_0 > 0$ be the extra stored depth and $\delta = W_0 - R > 0$ the reduction in read depth. The progressive codec saves $\delta S(N, L)$ read bits and writes $a(N + L)$ additional bits. Its total logical payload traffic is therefore smaller exactly when

$$\delta \left[LN + \frac{L(L-1)}{2} \right] > a(N + L).$$

To express the crossover in terms of the decode length, define $c = \delta(N - \frac{1}{2}) - a$. The read saving minus the additional write cost is $\frac{\delta}{2}L^2 + cL - aN$. This quadratic has a positive leading coefficient and is nonpositive at $L = 0$. Hence, for $L > 0$, the progressive codec has lower traffic exactly above the nonnegative root

$$L > \frac{-c + \sqrt{c^2 + 2\delta a N}}{\delta}.$$

For a sequence grown from an empty cache, this condition simplifies to $L > 1 + 2(W - W_0)/(W_0 - R)$. The derivation sums the actual reuse count of every entry and does not assume that all entries have the same age.

A.5 Equal Storage

When $W = W_0$, the two codecs have the same payload capacity and write traffic. Reducing R then lowers only the repeated-read term. This comparison isolates the quality obtained from query-dependent access at fixed storage. Whether the resulting approximation is acceptable must be determined from the downstream task; the allocation theorem does not claim that a partial prefix reproduces the full-depth reconstruction.

A.6 Capacity and Physical Traffic

Amortizing write traffic does not reduce the resident cache size. At the end of the sequence, the ratio of payload capacities is W/W_0, independently of L. A cache with $W = 8$ therefore occupies twice the payload memory of a four-bit cache. This larger footprint can limit context length or the number of concurrent requests, so capacity must be reported separately from traffic.

Logical payload counts also differ from physical device traffic. Actual byte movement includes packet rounding, norms, scales, residual corrections, descriptors, sorting buffers, and intermediate attention arrays. Cache hits may reduce DRAM traffic, whereas irregular accesses and additional passes may increase it. Neither the equations above nor the allocation objective alone determines device latency.

The two principal comparisons should therefore be interpreted differently. For READKV $(8, 4)$, a fixed four-bit codec reads the same logical payload, while READKV retains twice as many payload bits. For READKV $(4, 2)$, both methods retain four bits, but READKV halves the logical read budget. We report storage capacity and logical payload traffic separately. The systems evaluation also measures physical traffic and the implementation overheads omitted by this scalar model.

B RELATED KV-CACHE METHODS

Prior work reaches our setting through several lines of research. Progressive coding studies representations that remain decodable at several precisions. KV-cache quantization methods vary the precision retained for different cache entries, while query-adaptive retrieval methods vary what is fetched for the current query. These choices are related but not equivalent. Changing the stored cache affects every later query, whereas selective reading leaves unread information available for future use. The following sections place READKV within each line of work.

B.1 NESTED CODES

Progressive descriptions are classical in source coding (Equitz & Cover, 1991; Rimoldi, 1994). Successive-refinement theory asks when one nested representation can match the rate–distortion performance of separate codes designed for each rate. A scalar tree ensures that every prefix is decodable, but it does not by itself establish this stronger optimality property. Requiring a common tree can also increase distortion relative to fitting an independent codebook at each depth (Gray & Neuhoff, 1998).

Any-Precision LLM is a close representational precursor for neural-network compression. It recursively splits nonuniform weight centroids and loads only the requested bit planes (Park et al., 2024). MatQuant instead trains one weight representation to remain useful under several precision slices (Nair et al., 2025). Both demonstrate that a single stored code can support multiple reconstruction depths. READKV uses this property for KV entries and allocates a finite query-time read budget across key channels and value tokens.

More abstractly, the separation between preprocessing and later probes connects our problem to static data structures and locally decodable source coding (de Wolf, 2008; Makhdoumi et al., 2013). Approximate inner-product sketches instead bound the representation size needed to support geometric queries (Alon & Klartag, 2016). Our formulation jointly constrains the stored real-vector representation and the portion inspected for each future query.

B.2 ATTENTION-GUIDED PRECISION ALLOCATION

Several KV-cache methods use attention history to decide what precision to retain. QAQ, MiKV, and ZipCache all allocate precision using attention-based importance (Dong et al., 2024; Yang et al., 2024; He et al., 2024). These decisions need not remain fixed after prefill. QAQ updates precision from recent attention windows, while ZipCache periodically recompresses the cache during decoding. QAQ also retains an unquantized CPU copy. When a token later needs greater precision, that copy is transferred back and requantized (Dong et al., 2024, Section 4.2).

A separate group also adapts the precision retained in the cache. PM-KVQ starts at higher precision and reduces stored precision as memory fills. It uses an integer program to allocate precision across transformer blocks (Liu et al., 2026). MixKVQ combines query importance with quantization sensitivity to assign key-channel precision. Its lazy update rule uses accumulated query information when quantizing blocks outside a recent-token buffer (Zhang et al., 2026b). RDKV computes attention-sensitive weights after prefill and applies reverse water-filling to allocate stored bits, including zero-bit eviction (Zhang et al., 2026a).

These methods adapt the precision resident in the cache. READKV instead keeps one progressive code unchanged and adapts only what is read from it for each query. Additional precision is obtained by reading a longer prefix, rather than by recovering and requantizing a separate copy. The current query determines the key-prefix depths, and attention computed from the reconstructed keys then determines the value-prefix depths. The calibration-only reader and the fixed-split key-stage control

in Appendix F.5 test the benefit of this current-query information over importance fixed before evaluation.

B.3 Query-Adaptive Cache Access

Other methods adapt the access pattern to the current query. SparQ first uses query-dependent feature selection to approximate attention, then retrieves the most important value tokens (Ribar et al., 2024). Quest ranks cache pages from the current query and compact page summaries (Tang et al., 2024). Loki uses low-dimensional key information to select tokens (Singhania et al., 2024). These methods establish that the data retained in the cache need not coincide with the data fetched for one query. ThinK is related but changes the retained cache: it uses an observation window of queries to select key channels and keeps the resulting pruned representation (Xu et al., 2025).

Xie et al. (2025) provide the closest precedent for precision-selective KV access. They evaluate several precision tiers for ranked KV pages and co-design retrieval of floating-point bit planes with a compression-aware memory controller. Their work already contains the principle of retaining more precision than a query fetches. ReadKV uses a nested nonuniform scalar code with every prefix depth available to the reader. Under diminishing refinement gains, it exactly optimizes its stated separable objective within an explicit read budget, first across key channels using the current query and then across value tokens using reconstructed attention. The formulation in Section 2 separates storage, read budget, and access stages as constraints on one finite record. Our separation results address a further question: whether query-dependent addressing can outperform query-independent readers under a common read budget, even when the competing encoders and decoders are unrestricted.

B.4 Fixed-Rate Quantization

Fixed-rate quantizers provide the main compact-cache baselines. KIVI uses per-channel quantization for keys and per-token quantization for values, together with a recent full-precision residual buffer (Liu et al., 2024). KVQuant combines calibrated nonuniform levels with per-channel and pre-RoPE key quantization, as well as sparse outlier treatment (Hooper et al., 2024). A generic uniform quantizer does not reproduce either method, and their auxiliary state must be included in storage and read accounting.

TurboQuant develops a rotation-based scalar quantizer designed for vector error. A second variant adds a one-bit randomized sketch of the quantization residual, called QJL, which yields an unbiased inner-product estimator (Zandieh et al., 2026). Comparisons must therefore identify the TurboQuant variant and count its auxiliary data. Our direct Lloyd control includes norm correction but does not reproduce either complete TurboQuant algorithm.

These baselines support two complementary comparisons. Matching the read budget tests whether retaining additional refinements improves quality per fetched bit. Matching storage tests whether query-dependent reading helps at the same cache capacity. Neither comparison subsumes the other.

B.5 Bit-Allocation Methods

The optimizer used by ReadKV also has classical foundations. Selecting the largest available marginal gain under diminishing returns is a standard discrete allocation rule (Fox, 1966; Shoham & Gersho, 1988). Continuous bit allocation and reverse water-filling have a similarly long history (Segall, 1976; Cover & Thomas, 2006).

We use these tools to obtain an executable schedule that respects the finite prefix depth of the stored code. The problem-specific content lies in what is allocated, namely key channels and value tokens, how the current query defines their weights, and which finite prefixes may be fetched. The greedy rule and water-filling are tools used in the analysis, not standalone novelty claims.

C Notation and Access Model

C.1 Notation Reference

Table 4 summarizes notation used across multiple sections. Symbols introduced only within a single proof are defined locally and omitted from the table.

C.2 Formal Access Model

The record is a vector $Z \in \{0,1\}^w$ indexed by the public address set $[w] = \{1, \dots, w\}$. An encoder using fewer than w cells may pad the remaining locations. Public state θ describes the code and layout but is fixed independently of the input pair (x, y). Any object-specific information needed to interpret a cell belongs to the charged record. This is a one-bit-cell probe model. Only stored cells and probes are charged. Encoder preprocessing, address computation, transcript processing, decoder computation, and arithmetic precision are free. The systems evaluation measures physical traffic and execution cost separately.

Let Ω_0 be public randomness, Ω_e encoder-local randomness, and Ω_r reader-local randomness. These variables are mutually independent, and each is independent of the fixed pair (x, y). The encoder writes $Z = \mathrm{Enc}(x; \theta, \Omega_0, \Omega_e)$. The reader can access encoder-local randomness only through Z.

A query-oblivious reader chooses an ordered address list of length at most r from $(\theta, \Omega_0, \Omega_r)$ alone. A fully branching reader may choose its jth address as

$$A_j = A_j(y, \theta, \Omega_0, \Omega_r, A_1, Z_{A_1}, \dots, A_{j-1}, Z_{A_{j-1}}),$$

and may stop early. The total number of probes is at most r on every execution, not only in expectation. Repeated probes carry no new information and may be replaced by retaining the earlier response.

The separation theorems use a stronger comparison class that we call *query-independent branching*. Its address recursion is the same as the fully branching recursion above, except that y is omitted from every address and stopping rule. Thus, later addresses may depend on earlier payloads, but the complete access path cannot depend on the query. This class contains the query-oblivious readers defined above.

A $\mathsf{staged}(J)$ reader makes at most J requests, each containing a batch of addresses. The first batch may use $(y, \theta, \Omega_0, \Omega_r)$, and each later batch may also use the transcript of earlier batches. A batch is chosen before any of its values are returned. The total number of requested addresses is at most r. Empty batches may be omitted. In every class, the final answer may be an arbitrary measurable function of y, θ, (Ω_0, Ω_r), and the observed address–payload transcript.

For an access class A, let Π range over complete encoder–reader–decoder protocols satisfying its rules and budgets, and let $\widehat{s}_\Pi(x, y)$ denote the resulting estimate of $x^\top y$. On the Euclidean unit ball $\mathbb{B}_2^d$, the minimax worst-case mean-squared error is

$$\mathcal{D}^{\mathrm{wc}}_{d,\mathsf{A}}(W, R) = \inf_{\Pi \in \mathsf{A}} \sup_{x,y \in \mathbb{B}_2^d} \mathbb{E}_\Pi \left[\left(x^\top y - \widehat{s}_\Pi(x, y) \right)^2 \right].$$

The input pair (x, y) is fixed before the random coins are drawn, and the expectation is over protocol randomness.

Proposition 1 (Access Hierarchy). *Fix w, r, the input domain, the loss, and the randomization convention. For every integer $J \geq 1$,*

$$\mathsf{oblivious} \subseteq \mathsf{staged}(1) \subseteq \mathsf{staged}(J) \subseteq \mathsf{staged}(J+1) \subseteq \mathsf{branching}.$$

The corresponding minimax risks are ordered in the reverse direction. If $r \geq 1$, then $\mathsf{staged}(J)$ and branching have the same optimal risk for $J \geq r$. If $r = 0$, all access classes coincide.

Proof. A $\mathsf{staged}(1)$ reader can ignore y, so every query-oblivious protocol is $\mathsf{staged}(1)$. An at-most-J protocol is also an at-most-$(J+1)$ protocol because it may leave the last batch unused.

To simulate a staged protocol, a branching reader computes the next batch from the transcript of earlier batches. It probes those addresses one at a time. The simulator withholds their payloads from

the staged policy until the batch is complete. This produces the same transcript with the same encoder and probe count. Conversely, a branching protocol with at most r probes is a staged protocol with at most r singleton batches. Hence $\mathsf{staged}(J)$ and branching have the same achievable risks once $J \geq r$.

Each relation is an inclusion of complete protocol classes under the same loss and budgets. Taking the infimum of the same worst-case expected loss therefore reverses the inclusions. If $r = 0$, no payload is observed, so all access classes contain the same decoders. □

Proposition 2 (Zero-Read Limit)**.** *For $d \geq 1$ and every access class A, the risk is one whenever $\lfloor Rd \rfloor = 0$.*

Proof. The constant answer zero has squared error at most one because $|x^\top y| \leq 1$. Conversely, fix any unit vector y. With no reads, the random answer A is a function only of y, public state, and random coins independent of x. It therefore has the same distribution under $x = y$ and $x = -y$. If $\mathbb{E}[A^2] = \infty$, the lower bound is immediate. Otherwise, the larger of the two expected squared errors is at least their mean,

$$\tfrac{1}{2}\mathbb{E}[(1 - A)^2 + (-1 - A)^2] = 1 + \mathbb{E}[A^2] \geq 1.$$

□

C.3 Full-Read Limit

Let $\mathcal{D}_d^{\mathrm{full}}(W)$ be the worst-case risk defined above, except that the decoder receives the entire $w = \lfloor Wd \rfloor$-bit message. The infimum uses the same encoders and randomness conventions. In particular, the full-message decoder receives any public randomness and may generate the same independent reader-local coins as an access protocol. The scalar decoder may depend on the query and the whole message. It need not first reconstruct a vector.

Proposition 3 (Full-Read Equivalence)**.** *If $r \geq w$, every access class has risk $\mathcal{D}_d^{\mathrm{full}}(W)$. In particular, for $J \geq 1$,*

$$\mathcal{D}^{\mathrm{wc}}_{d,\mathsf{oblivious}}(W, W) = \mathcal{D}^{\mathrm{wc}}_{d,\mathsf{staged}(J)}(W, W) = \mathcal{D}^{\mathrm{wc}}_{d,\mathsf{branching}}(W, W) = \mathcal{D}_d^{\mathrm{full}}(W).$$

This equality applies to randomized as well as deterministic protocols.

Proof. Given the full record Z, a decoder can simulate any legal access policy: whenever the policy requests address A_j, the simulator returns Z_{A_j}. Using the same random coins preserves the complete transcript and output distribution for every fixed (x, y). Thus, the full-message risk is no larger than the risk of any access class.

For the reverse inequality, when $r \geq w$, a query-oblivious reader requests the fixed list $(1, \ldots, w)$ in one batch and applies any full-message decoder. This simulation preserves the encoder and output distribution. Taking worst cases and then infima in both directions proves equality. □

There is therefore no further information to gain by allowing $R > W$ in this matched cell model. The access constraints disappear at $R = W$. The remaining problem is how best to encode a vector for full-message inner-product queries.

D Progressive Codec and Distortion Model

This section gives the full ReadKV pseudocode and records the finite properties of the progressive scalar codec used in the analysis and experiments.

D.1 Full Pseudocode

Algorithms 1 and 2 describe the same ReadKV encoder, two-stage reader, and decoder. The appendix version makes the public inputs, allocation loop, stopping rule, tie-breaking rule, and shared-head reuse explicit.

Algorithm 2 READKV encoder, two-stage reader, and decoder.

Public Inputs. Stored depth W; transforms U_K, U_V; calibrated means and scales; distortion table $D(t)$, which predicts squared scalar reconstruction error after t prefix bits; and integer budgets B_K, B_V, each measured in one-level item refinements.

Per-Token Encoder. Transform and standardize the token's keys and values. Encode each scalar as a W-bit path through the nested quantization tree, and append those bits to the cache bit planes.

Shared Prefix Allocator $\mathrm{ChooseDepths}(a, D, W, B)$. An item is a transformed key channel in Stage 1 or a cached value token in Stage 2. Its score a_i measures its importance to the current query, and t_i is its planned prefix depth.

1. Set every depth to zero.
2. For each of at most B allocation steps, compute $g_i = a_i[D(t_i) - D(t_i + 1)]$ for every item with $t_i < W$. This is the predicted reduction in weighted error from extending item i by one level.
3. If every available gain is nonpositive, stop. Otherwise choose an item with the largest gain, set $t_i \leftarrow t_i + 1$, and continue. Break ties using a fixed public item order.
4. Return the selected depth vector t.

Key Reader (Stage 1).

1. Transform each query as $q'_h = U_K q_h$ and compute the importance of each key-channel item using Equation (1).
2. Set $t^K = \mathrm{ChooseDepths}(a^K, D, W, B_K)$. Each entry of t^K specifies the prefix depth used for every scalar in that key-channel item.
3. Fetch these prefixes once per physical head.

Intermediate Decoder. Reconstruct $\widehat{K}$, reuse each reconstruction across the query heads sharing that physical head, compute the masked logits, and apply softmax to obtain $\widehat{\alpha}$.

Value Reader (Stage 2).

4. Use $\widehat{\alpha}$ to compute the importance of each value-token item using Equation (2).
5. Set $t^V = \mathrm{ChooseDepths}(a^V, D, W, B_V)$. Each entry of t^V specifies the prefix depth used for every scalar in that value-token item.
6. Fetch these prefixes.

Final Decoder. Reconstruct the transformed values $\widehat{V}$ and return $U_V^\top \sum_j \widehat{\alpha}_{h,j} \widehat{V}_{g(h),j}$ for each query head h.

D.2 PROGRESSIVE SCALAR CODEC

The scalar code is a binary tree fitted to $Z \sim \mathcal{N}(0, 1)$. Every internal node partitions its interval into two children to reduce probability-weighted squared reconstruction error; each child stores its conditional mean. A depth-t prefix identifies one of the resulting intervals. The maximum-depth leaf index therefore supplies all coarser reconstructions. Their centroids need not be stored separately for each vector. The shared centroid tables are charged as shared model state.

The implementation polishes each split to Lloyd stationarity, so every boundary is the midpoint of its two child centroids. The finite properties needed by the theory are verified directly from the deployed table in Lemma 4.

The approximate standardized distortion curve is

t	0	1	2	3	4	5	6	7	8
$D(t)$	1	.3634	.1175	.03484	.009796	.002663	.000708	.000185	.00004795.

The allocator uses a calibrated distortion curve. For the deployed float32 gain table, define the effective curve by $D(0) = 1$ and $D(t) = 1 - \sum_{s=1}^{t} \Delta_s$, where Δ_s is the stored gain at level s. This curve approximates standardized Gaussian reconstruction MSE. It need not equal the exact Gaussian expectation because the deployed centroids are stored at finite precision. The following lemma verifies the required gain inequalities directly from the finite table and also bounds the codec's pointwise reconstruction error.

Lemma 4 (Finite Bounds for the Deployed Codec)**.** *Let Δ_t be the stored float32 refinement gains, equivalently $D(t-1) - D(t)$ for the effective curve above. The key and value gain tables coincide*

and satisfy

t	*lower bound for* Δ_t	*upper bound for* Δ_t
1	0.63661974	0.63661975
2	0.24589838	0.24589839
3	0.08264372	0.08264373
4	0.02504228	0.02504229
5	0.00713273	0.00713274
6	0.00195518	0.00195519
7	0.00052264	0.00052265
8	0.0001373218	0.0001373220

In particular, $\Delta_1 > \cdots > \Delta_8 > 0$. *The depth-eight pointwise error on the standardized interval satisfies*

$$0.0096543431 < \epsilon_8 := \sup_{|z|\le 1} |z - Q_8(z)| < 0.0096543432 < 0.0097.$$

The positive depth-one centroid and the depth-two endpoint reconstruction satisfy

$$0.7978845834 < c_1 < 0.7978845835, \qquad 1.5104175806 < Q_2(1) < 1.5104175807,$$

with $Q_2(-1) = -Q_2(1)$. *Consequently,*

$$0.26052610 < (Q_2(1) - 1)^2 < 0.26052612.$$

Proof. The proof consists of three finite checks. First, the pointwise error of the piecewise-constant depth-eight quantizer is maximized at a cell endpoint. Second, direct comparison of the eight stored gains proves positivity and strict decrease. Third, the depth-one and depth-two claims follow from their stored centroid entries. Each float32 entry has an exact rational value, and the displayed decimal intervals are outward bounds on those values.

At depth eight, write the ordered cells as C_j and their reconstruction values as μ_j, for $j = 1, \ldots, 256$. For every cell that intersects $[-1, 1]$, Q_8 is constant and

$$\sup_{z \in C_j \cap [-1,1]} |z - Q_8(z)| = \max_{u \in \partial(C_j \cap [-1,1])} |u - \mu_j|.$$

Thus ϵ_8 is the maximum of the endpoint deviations from the 256 deployed cells. Exact enumeration gives its maximum at the endpoint $-0.981598794460\ldots$ adjacent to centroid $-0.991253137588\ldots$. Their difference is $0.009654343128\ldots$, which lies in the displayed interval. Symmetry gives the corresponding positive endpoint.

The depth-one and depth-two bounds follow directly from their table entries. The eight gain entries of the same finite table lie in the displayed intervals. These intervals are disjoint and ordered, which proves $\Delta_1 > \cdots > \Delta_8 > 0$. Squaring the positive interval for $Q_2(1) - 1$ gives the final bound. □

The gain inequalities show that the implemented curve satisfies the hypotheses of Proposition 7. Physical scales appear in a_i; they are not multiplied into $D(t)$ a second time. The evaluated centroid code is deterministic and is not claimed to be pointwise unbiased.

Calibration uses held-out training documents to estimate means and scales per layer, physical head, and transformed channel. Keys and queries share their key transform, and values and reconstructed outputs share a separate value transform. The scalar Gaussian fit and the calibration procedure are design tools. Neither imposes the independence assumption excluded by the main formulation.

D.3 Diagonal Surrogate

For a residual $e = x - \widehat{x}$ at a realized query q, the identity $\mathbb{E}[(q^\top e)^2 \mid Q = q] = q^\top \mathbb{E}[ee^\top \mid Q = q]q$ holds without assuming zero-mean errors. The implemented key surrogate replaces the conditional second-moment matrix by a depth-separable diagonal approximation estimated from calibration. The true matrix can depend on q and may have off-diagonal entries. Summing the resulting diagonal expression over sibling heads yields Equation (1) exactly within that approximation.

For values, the full squared residual contains cross-token terms $\widehat{\alpha}_{h,j}\widehat{\alpha}_{h,k}e_j^\top e_k$. Neglecting these terms gives the physical-scale choice $s_g = \sum_c (\sigma^V_{g,c})^2$ in Equation (2). The language-model experiments

use $s_g = 1$ instead and optimize in standardized value units. This distinction matters when one allocation budget is shared across physical heads. Reconstructed attention probabilities add another source of output error. The allocation theorem therefore optimizes the declared surrogate, while the complete causal experiment tests how that surrogate behaves in the language model.

E PROOFS OF THE READKV PERFORMANCE GUARANTEES

This section supplies the complete argument behind Section 4. The dependency chain has four parts. Appendix E.1 proves the two finite-dimensional separations directly. Proposition 7 then proves exact stagewise prefix allocation under diminishing gains. Lemma 10 and Proposition 11 convert the resulting key and value objectives into an attention-output bound. Finally, Theorem 13 combines the allocation and attention-error results for the complete READKV procedure. The continuous allocation and integer-rounding results are auxiliary comparisons and are not used in the proofs of the three main guarantees.

E.1 FINITE-DIMENSIONAL SEPARATIONS

All logarithms in this section are natural. Both converse proofs use the same entropy–MMSE inequality. We write I for mutual information, H for discrete entropy, and h for differential entropy.

Cube Entropy–MMSE Bound. Let $n \geq 1$ and $b_{\text{info}} \geq 0$. Let Z be uniform on $[-1,1]^n$, let O be any observation satisfying $I(\mathsf{Z}; O) \leq b_{\text{info}} \log 2$, and define $\mathcal{M}_n = \mathbb{E}\|\mathsf{Z} - \mathbb{E}[\mathsf{Z} \mid O]\|_2^2$. Then

$$\frac{\mathcal{M}_n}{n} \geq \frac{2^{\,1-2b_{\text{info}}/n}}{\pi e}. \tag{3}$$

Indeed, $h(\mathsf{Z}) = n \log 2$, so the information bound gives $h(\mathsf{Z} \mid O) \geq (n - b_{\text{info}}) \log 2$. For almost every observation $O = o$, Gaussian maximum entropy and the determinant–trace inequality give

$$h(\mathsf{Z} \mid O = o) \leq \frac{n}{2} \log\left(\frac{2\pi e}{n} \mathbb{E}\big[\|\mathsf{Z} - \mathbb{E}[\mathsf{Z} \mid O]\|_2^2 \,\big|\, O = o\big] \right).$$

Taking expectations and applying Jensen's inequality bounds the conditional entropy by $\frac{n}{2} \log(2\pi e\, \mathcal{M}_n / n)$. Rearranging proves Equation (3). These standard entropy inequalities are reviewed by Cover & Thomas (2006).

E.1.1 KEY-STAGE SEPARATION

We first isolate the key-reading stage on a transformed-sparse family. Fix a public orthogonal transform $U \in \mathbb{R}^{d \times d}$ and an integer $k \geq 1$. Define

$$\mathcal{X}_U = \{U^\top z/\sqrt{d} : z \in [-1,1]^d\}, \qquad \mathcal{Y}_{U,k} = \{U^\top q : \|q\|_2 \leq 1,\ \|q\|_0 \leq k\}.$$

Both sets lie in $\mathbb{B}_2^d$. The query sparsity is imposed in the public transformed basis. In particular, when U is a Hadamard–sign transform, a sparse transformed query can be dense in the original coordinates.

Let $D(t)$ be the scalar distortion at prefix depth t, and let $\Delta_t = D(t-1) - D(t)$. For an integer $L \geq 1$, let Q_L be the depth-L reconstruction of a nested binary scalar code with $\Delta_t > 0$ for $t = 1, \ldots, L$. Define $\epsilon_L = \sup_{|u| \leq 1} |u - Q_L(u)|$.

For $x \in \mathcal{X}_U$, write $z = \sqrt{d}\, Ux \in [-1,1]^d$. The encoder stores the L-bit code of every coordinate of z. Given y, write $q = Uy \in \mathbb{R}^d$ and assign coordinate i weight $a_i = q_i^2/d$. This is the single-vector specialization of the key stage in Algorithm 1, with $G = H = T = 1$, transform $U_K = U$, zero calibration mean, scale $1/\sqrt{d}$, and key-refinement budget $B_K = Lk$. Under this specialization, one allocation item is one transformed coordinate. The reader fetches all L bits of every coordinate in $\operatorname{supp}(q)$ and nothing elsewhere. The reads form one batch because their addresses depend only on the query and public code tables. The resulting estimate is $\widehat{s}_{\text{READKV}}(x, y) = d^{-1/2} \sum_i q_i Q_L(z_i)$.

Theorem 5 (Finite READKV Separation for Transformed-Sparse Queries)**.** *Let $d \geq 1$ and $1 \leq k \leq d$. Let L, r, w be positive integers satisfying $Ld \leq w$ and $Lk \leq r$. The construction above stores Ld*

informative bits in a w-bit record, padding unused cells if necessary, and uses at most Lk probes. Its worst-case error satisfies

$$\sup_{x\in\mathcal{X}_U,\ y\in\mathcal{Y}_{U,k}} \left(x^\top y - \widehat{s}_{\textsc{ReadKV}}(x,y)\right)^2 \le \frac{k\epsilon_L^2}{d}.$$

In contrast, every randomized protocol Π whose reader is query-independent branching and uses at most r probes satisfies

$$\sup_{x\in\mathcal{X}_U,\ y\in\mathcal{Y}_{U,k}} \mathbb{E}_\Pi\left[\left(x^\top y - \widehat{s}_\Pi(x,y)\right)^2\right] \ge \frac{2^{\,1-2r/d}}{\pi e\, d}.$$

The competing protocol may use an arbitrary randomized encoder and decoder. The lower bound imposes no storage restriction beyond requiring a finite binary record. Therefore, the ReadKV *construction has strictly smaller worst-case MSE whenever $k\epsilon_L^2 < 2^{\,1-2r/d}/(\pi e)$.*

The theorem applies to any finite nested scalar code with positive prefix gains. Its numerical instance and Theorem 6 use the deployed depth-eight table. Both lower bounds allow the reader to branch on returned payloads, but neither an address nor the stopping decision may depend on the query. The decoder may still use the query and the complete transcript. All of these permissions are included in the lower bounds.

Theorem 5 can be instantiated at any admissible code depth. If a nested scalar code has positive gains through depth $L_{\max}$, then for $w \ge d$ and $r \ge k$ one may choose any integer

$$1 \le L \le \min\left\{L_{\max}, \left\lfloor \frac{w}{d} \right\rfloor, \left\lfloor \frac{r}{k} \right\rfloor\right\}.$$

This choice uses Ld stored bits and at most Lk probes. The depth-eight numerical instance takes $L = 8$, $w = 8d$, and $r = 2d$.

The proof of Theorem 5 establishes three subclaims. First, the ReadKV reader recovers every active transformed coordinate to depth L, which gives the pointwise estimate $|x^\top y - \widehat{s}| \le \epsilon_L\sqrt{k/d}$. Second, after conditioning on all reader-visible randomness, a query-independent r-probe transcript is a binary decision-tree leaf. It has at most 2^r possible values and therefore reveals at most $r\log 2$ nats about the source. Third, a uniformly random signed coordinate query turns inner-product prediction into source estimation: its Bayes risk is the source MMSE divided by d^2. The entropy–MMSE calculation lower-bounds that source MMSE by $d\,2^{1-2r/d}/(\pi e)$.

Proof of Theorem 5. We first verify the domain and the ReadKV upper bound. For $z \in [-1,1]^d$, $\|U^\top z/\sqrt{d}\|_2^2 = \|z\|_2^2/d \le 1$. For every q in the definition of $\mathcal{Y}_{U,k}$, $\|U^\top q\|_2 = \|q\|_2 \le 1$. Thus $\mathcal{X}_U, \mathcal{Y}_{U,k} \subseteq \mathbb{B}_2^d$.

For $x \in \mathcal{X}_U$, write $z = \sqrt{d}\,Ux \in [-1,1]^d$. If $y = U^\top q \in \mathcal{Y}_{U,k}$, then $Uy = q$. Let $S = \{i : q_i \ne 0\}$, so $|S| \le k$. The encoder writes the depth-L leaf index of each standardized coordinate, using exactly Ld informative bits, and pads to w cells if needed. The centroid table is public state. With shared scale $1/\sqrt{d}$, the depth-L reconstruction of coordinate $(Ux)_i$ is $Q_L(z_i)/\sqrt{d}$. The key importance of coordinate i is

$$a_i = \left((Uy)_i/\sqrt{d}\right)^2 = \frac{q_i^2}{d}.$$

Every one of the first L marginal gains is strictly positive. Hence, each coordinate in S has L positive prefix edges, while every coordinate outside S has zero weighted gains. Until a coordinate in S reaches depth L, its next edge remains in the priority queue. There are at most $L|S| \le Lk \le r$ positive edges in total. The greedy reader therefore fetches every active coordinate to depth L. It then stops because no positive marginal gain remains. The weights, gain table, and tie-breaking order determine the full address list from y and public state, so all probes can be issued in one batch.

The target and the ReadKV answer are

$$x^\top y = \frac{1}{\sqrt{d}}\sum_{i\in S} q_i z_i, \qquad \widehat{s}_{\textsc{ReadKV}}(x,y) = \frac{1}{\sqrt{d}}\sum_{i\in S} q_i Q_L(z_i).$$

Therefore,

$$|x^\top y - \widehat{s}_{\textsc{ReadKV}}(x,y)| \le \frac{\epsilon_L}{\sqrt{d}} \sum_{i\in S} |q_i| \le \epsilon_L \sqrt{\frac{|S|}{d}}\, \|q\|_2 \le \epsilon_L \sqrt{\frac{k}{d}}.$$

The second inequality is Cauchy–Schwarz. Squaring proves, pointwise over $\mathcal{X}_U \times \mathcal{Y}_{U,k}$,

$$\big(x^\top y - \widehat{s}_{\textsc{ReadKV}}(x,y)\big)^2 \le \frac{k\epsilon_L^2}{d}.$$

We next lower-bound an arbitrary query-independent branching protocol. This includes arbitrary randomized encoders and decoders. Draw Ξ uniformly from $[-1,1]^d$, draw K uniformly from $[d]$, and draw an independent uniform sign η. Set

$$X = U^\top \Xi/\sqrt{d}, \qquad Y = \eta U^\top e_K.$$

This prior is supported on $\mathcal{X}_U \times \mathcal{Y}_{U,k}$ because $k \ge 1$. It also satisfies $\mathbb{E}[YY^\top] = I_d/d$.

Let $C = (\theta, \Omega_0, \Omega_r)$ contain the public state and all reader-visible random coins. Conditional on $C = c$, a query-independent reader is a binary decision tree of depth at most r. Its leaf Λ determines every returned bit, every address selected from earlier bits, and the stopping decision. The tree has at most 2^r leaves. Encoder-local randomness may remain unconditioned. It changes the distribution of Λ given (Ξ, C) but not the number of leaves. Since C is independent of Ξ,

$$I(\Xi; C, \Lambda) = I(\Xi; \Lambda \mid C) \le H(\Lambda \mid C) \le r \log 2.$$

Thus, this bound includes payload-dependent branching and early stopping. It does not use the write budget, so it remains valid for every fixed finite binary record length under the same at-most-r probe limit.

Put $O = (C, \Lambda)$ and $\bar{X}(O) = \mathbb{E}[X \mid O]$. The query is independent of (X, O) because (K, η) is independent of the source and all protocol coins, and the access path does not use (K, η). Hence

$$\mathbb{E}[X^\top Y \mid O, Y] = Y^\top \mathbb{E}[X \mid O, Y] = Y^\top \bar{X}(O).$$

This is the Bayes estimator given everything available to the decoder. For any possibly randomized decoder, conditional least squares therefore gives

$$\mathbb{E}\big[(X^\top Y - \widehat{s}_\Pi)^2\big] \ge \mathbb{E}\Big[\big(X^\top Y - \mathbb{E}[X^\top Y \mid O, Y]\big)^2\Big].$$

Substituting the conditional mean and averaging over the random signed coordinate query yields

$$\begin{aligned}\mathbb{E}\Big[\big(X^\top Y - \mathbb{E}[X^\top Y \mid O, Y]\big)^2\Big] &= \mathbb{E}\big[(Y^\top (X - \bar{X}(O)))^2\big] \\ &= \frac{1}{d}\,\mathbb{E}\big[\|X - \bar{X}(O)\|_2^2\big] \\ &= \frac{1}{d^2}\,\mathbb{E}\big[\|\Xi - \mathbb{E}[\Xi \mid O]\|_2^2\big].\end{aligned}$$

The middle equality uses $\mathbb{E}[YY^\top] = I_d/d$. It does not require the decoder to reconstruct a vector. The last equality uses $\bar{X}(O) = U^\top \mathbb{E}[\Xi \mid O]/\sqrt{d}$ and orthogonality of U.

Let $\mathcal{M} = \mathbb{E}\|\Xi - \mathbb{E}[\Xi \mid O]\|_2^2$. Equation (3), applied with $n = d$ and $b_{\text{info}} = r$, gives $\mathcal{M}/d \ge 2^{1-2r/d}/(\pi e)$. The preceding Bayes-risk inequality is therefore at least $2^{1-2r/d}/(\pi e\, d)$. A worst-case risk is no smaller than its average under a prior supported on the same domain. Comparing this lower bound with the READKV upper bound gives the strict-separation condition in the theorem.

For the numerical instance, set $L = 8$ and $r = 2d$. Lemma 4 gives $\epsilon_8 < 0.0097$. Also, $2^{1-2r/d}/(\pi e) = 1/(8\pi e) > 0.01463$, while $(0.0097)^2 = 0.00009409$. Their ratio exceeds 155. More precisely, using $\pi < 3.142$ and $e < 2.719$, the ratio between the lower coefficient and the displayed READKV upper coefficient exceeds $155.507/k$. At $d = 128$ and $k = 32$, it exceeds 4.85. □

E.1.2 TWO-STAGE ATTENTION SEPARATION

We give the full construction and quantitative bounds behind Theorem 6. Both cached values are arbitrary inputs, and the competing reader cannot use the query to decide which stored cells to inspect.

For an integer $m \geq 1$, let $e_1 \in \mathbb{R}^m$ be the first standard basis vector. Let Q_8 denote the depth-eight reconstruction of the deployed scalar code, and let c_1 be its positive reconstruction value at depth one. Choose $b \in \{-1, 1\}$ and values $v_1, v_2 \in [-1, 1]^m$. Set $k_1 = bc_1e_1 \in \mathbb{R}^m$ and $k_2 = -bc_1e_1 \in \mathbb{R}^m$. For a query sign $\eta \in \{-1, 1\}$, let $q_\eta = \eta(5\sqrt{m}/c_1)e_1 \in \mathbb{R}^m$. Under the standard $1/\sqrt{m}$ attention normalization, the two logits are $(5\eta b, -5\eta b)$. Define $p = (1+e^{-10})^{-1}$ and $\rho = 1-p = (1+e^{10})^{-1}$. Let $j_\star = 1$ when $\eta b = 1$ and $j_\star = 2$ otherwise, so $j_\star$ is the high-attention token. This is a separate attention family with the query norm specified above, not the unit-ball query domain. The exact output is $o(b, v_1, v_2, \eta) = pv_{j_\star} + \rho v_{3-j_\star} \in \mathbb{R}^m$. We write o for this vector when its arguments are clear. The cache encoder observes (b, v_1, v_2) before the query sign η is known.

Use identity transforms, zero calibration means, and unit scales, so a depth-zero value reconstructs to zero. The encoder stores all four vectors with the depth-eight progressive code. The key-refinement budget $B_K = 1$ extends one key-channel item by one level and therefore reads one bit from each of the two keys. The value-refinement budget $B_V = 8$ extends one value-token item through eight levels and therefore reads $8m$ value bits. The first batch reconstructs both keys exactly and identifies $j_\star$. The value allocator then compares gains weighted by p^2 for $j_\star$ and by ρ^2 for the other token. Writing $\Delta_t = D(t-1) - D(t)$ for the refinement gain at depth t, the deployed gain table satisfies $p^2\Delta_8 > \rho^2\Delta_1$ by Lemma 4 and the calculation below. Since $\Delta_1 \geq \cdots \geq \Delta_8 > 0$, every available refinement of $v_{j_\star}$ has larger weighted gain than the first refinement of $v_{3-j_\star}$. With $B_V = 8$, the allocator therefore reads all eight prefix levels of every coordinate of $v_{j_\star}$ and leaves $v_{3-j_\star}$ unread. The decoder therefore returns $\widehat{o}_{\text{READKV}} = pQ_8(v_{j_\star})$, with Q_8 applied coordinatewise. For any competing protocol Π, let $\widehat{o}_\Pi$ denote the output returned by its decoder.

Theorem 6 (Finite Two-Stage READKV Separation)**.** *For every $m \geq 1$, the construction above is a* staged(2) READKV *protocol. It stores $w = 32m$ bits and reads $r = 8m + 2$ bits. Let $\epsilon_8 = \sup_{|u|\leq 1} |u - Q_8(u)| < 0.0097$. Here $o = pv_{j_\star} + \rho v_{3-j_\star} \in \mathbb{R}^m$ is the exact attention output. Its worst-case per-coordinate attention-output error satisfies*

$$\sup_{b,v_1,v_2,\eta} \frac{1}{m}\|o - \widehat{o}_{\text{READKV}}\|_2^2 \leq (p\epsilon_8 + \rho)^2 < 9.50 \times 10^{-5},$$

where the supremum is over $b, \eta \in \{-1, 1\}$ and $v_1, v_2 \in [-1, 1]^m$. Every randomized protocol Π whose encoder is fixed before η is revealed and whose reader is query-independent branching with respect to η, using at most r probes, satisfies

$$\sup_{b,v_1,v_2,\eta} \frac{1}{m}\mathbb{E}_\Pi\big[\|o - \widehat{o}_\Pi\|_2^2\big] \geq \frac{\tanh^2(5)}{\pi e}\, 2^{-7-2/m} > 2.285 \times 10^{-4}.$$

The competing protocol may use an arbitrary randomized encoder and decoder and a finite binary record of arbitrary length. Thus, the certified separation factor exceeds 2.40 for every $m \geq 1$ and exceeds 9.52 at $m = 128$. The latter setting has $W = 8$ stored bits per cached scalar and realized read rate $R = (8m + 2)/(4m) = 2 + 1/(2m) \approx 2.004$.

The proof of Theorem 6 follows the causal order of the reader and establishes three subclaims. First, the two key probes recover the exact logits and identify the high-attention token. Second, $p^2\Delta_8 > \rho^2\Delta_1$ forces the value schedule $(8, 0)$ or $(0, 8)$, which gives the pointwise bound $m^{-1}\|o - \widehat{o}_{\text{READKV}}\|_2^2 \leq (p\epsilon_8 + \rho)^2$. Third, a query-independent transcript contains at most r bits about the pair of value vectors. Averaging the two query signs produces a 2×2 coefficient matrix with smallest eigenvalue $\tanh^2(5)/2$. An entropy–MMSE bound for the $2m$-dimensional source then gives the converse.

Proof of Theorem 6. Let $\Delta_t = D(t-1) - D(t)$ be the deployed scalar-code gain at depth t. Lemma 4 gives

$$0.63661974 < \Delta_1 < 0.63661975, \qquad 0.0001373218 < \Delta_8 < 0.0001373220,$$

and all eight gains are positive and decreasing.

The key score is positive only in the first transformed channel. One key refinement therefore reads the first prefix bit of that channel for both tokens, using two bit probes. Since the two nonzero key coordinates are the depth-one centroids $\pm c_1$, these prefixes reconstruct both keys exactly. The reconstructed logits equal the exact logits $(5\eta b, -5\eta b)$. Consequently, the high-attention token has probability $p = (1 + e^{-10})^{-1}$ and the other has probability $\rho = (1 + e^{10})^{-1}$.

The value planner compares marginal gains weighted by p^2 and ρ^2. The smallest weighted gain among the eight refinements of the high token is larger than the largest weighted gain of the low token because

$$\frac{p^2\Delta_8}{\rho^2\Delta_1} = e^{20}\frac{\Delta_8}{\Delta_1} > \frac{(4.84 \times 10^8)(0.0001373218)}{0.63661975} > 104400 > 1.$$

Here we used $e^{10} > 22000$, hence $e^{20} > 4.84 \times 10^8$. The greedy rule therefore assigns all eight value levels to the high token before selecting any level of the other token. Each value level contains one bit from each of its m coordinates. The second batch consequently uses $8m$ probes, for a total of $r = 8m + 2$. Storing two m-dimensional keys and two m-dimensional values at depth eight uses $w = 32m$ bits.

The zero-depth reconstruction is zero. Applying Q_8 coordinatewise, the two-stage output is $\widehat{o}_{\textsc{ReadKV}} = pQ_8(v_{j_\star})$. For every coordinate ℓ,

$$\begin{aligned}|o_\ell - \widehat{o}_{\textsc{ReadKV},\ell}| &= |p(v_{j_\star,\ell} - Q_8(v_{j_\star,\ell})) + \rho v_{3-j_\star,\ell}| \\ &\le p\epsilon_8 + \rho.\end{aligned}$$

Summing the squared coordinate bounds proves

$$\frac{1}{m}\|o - \widehat{o}_{\textsc{ReadKV}}\|_2^2 \le (p\epsilon_8 + \rho)^2.$$

We now lower-bound an arbitrary query-independent access path with at most $r = 8m + 2$ probes. Revealing b to the entire competing protocol before any probe can only strengthen it, so we grant this information for free. Draw V_1, V_2 independently and uniformly from $[-1, 1]^m$, draw B and η independently and uniformly from $\{-1, 1\}$, and use the attention instance in the theorem.

Let $C = (\theta, \Omega_0, \Omega_r)$ contain public state and reader-visible random coins. Conditional on (B, C), the query-independent branching reader is a binary decision tree of depth at most r. Its leaf Λ includes all returned bits, selected addresses, and the stopping decision. Encoder-local randomness may change the distribution over leaves, but there remain at most 2^r leaves. With $\mathsf{V} = (V_1, V_2) \in \mathbb{R}^{2m}$ and $O = (C, \Lambda)$,

$$I(\mathsf{V}; O \mid \mathsf{B}) = I(\mathsf{V}; \Lambda \mid \mathsf{B}, C) \le H(\Lambda \mid \mathsf{B}, C) \le r \log 2.$$

The equality uses the independence of C from (V, B).

Conditional on B, the sign $S = \eta\mathsf{B}$ is uniform and independent of (V, O). Define $a_+ = (p, \rho)^\top$ and $a_- = (\rho, p)^\top$. The Bayes estimator of the attention output given (O, B, η) is obtained by replacing V with $\mathbb{E}[\mathsf{V} \mid O, \mathsf{B}]$. Indeed, η is independent of (V, O) conditional on B, so conditioning additionally on η does not change this posterior mean. Put $E = \mathsf{V} - \mathbb{E}[\mathsf{V} \mid O, \mathsf{B}]$. Conditional least squares implies that every possibly randomized decoder has risk at least the risk of this posterior-mean estimator. Averaging that lower bound over the two values of S gives

$$\mathbb{E}\big[\|o - \widehat{o}_\Pi\|_2^2\big] \ge \frac{1}{2}\mathbb{E}\big[\|(a_+^\top \otimes I_m)E\|_2^2 + \|(a_-^\top \otimes I_m)E\|_2^2\big] = \mathbb{E}\big[E^\top (M \otimes I_m)E\big],$$

where I_m is the $m \times m$ identity matrix, $\otimes$ denotes the Kronecker product, and

$$M = \frac{1}{2}(a_+ a_+^\top + a_- a_-^\top) = \frac{1}{2}\begin{pmatrix} p^2 + \rho^2 & 2p\rho \\ 2p\rho & p^2 + \rho^2 \end{pmatrix}.$$

The eigenvectors $(1, 1)^\top$ and $(1, -1)^\top$ have eigenvalues $1/2$ and $(p - \rho)^2/2 = \tanh^2(5)/2$, respectively. Therefore, with $\mathcal{E} = \mathbb{E}\|E\|_2^2$,

$$\mathbb{E}\big[\|o - \widehat{o}_\Pi\|_2^2\big] \ge \frac{\tanh^2(5)}{2}\,\mathcal{E}.$$

It remains to lower-bound $\mathcal{E}$. The vector V is uniform on $[-1,1]^{2m}$ and independent of B. Hence $I(\mathsf{V};O,\mathsf{B}) = I(\mathsf{V};O \mid \mathsf{B}) \le r\log 2$. Equation (3), applied with $n = 2m$, $b_{\mathrm{info}} = r$, and observation (O,B), gives

$$\frac{\mathcal{E}}{m} \ge \frac{2^{\,2-r/m}}{\pi e} = \frac{2^{-6-2/m}}{\pi e}.$$

The Bayes-risk inequality therefore becomes

$$\frac{1}{m}\mathbb{E}\big[\|o-\widehat{o}_{\Pi}\|_2^2\big] \ge \frac{\tanh^2(5)}{\pi e}\,2^{-7-2/m}.$$

The prior is supported on the stated family, so the worst-case risk is at least this average risk.

For the numerical bounds, $e^{10} > 22000$ implies $\rho < 1/22001$ and $\tanh(5) > 21999/22001 > 0.9999$. Together with $\pi < 3.142$, $e < 2.719$, and $\epsilon_8 < 0.0097$, these bounds give

$$(p\epsilon_8+\rho)^2 < \left(0.0097 + \frac{1}{22001}\right)^2 < 9.50\times 10^{-5}.$$

Since $2^{-7-2/m} \ge 2^{-9}$ for $m \ge 1$, the lower bound exceeds 2.285×10^{-4}, and the ratio exceeds 2.40. At $m = 128$, the ratio is bounded below by

$$\frac{0.9999^2\,2^{-7-2/128}}{(3.142)(2.719)\,(0.0097+1/22001)^2} > 9.52.$$

□

E.1.3 Full-Unit-Ball Obstruction

The transformed-sparsity restriction in Theorem 5 cannot simply be removed for the deployed deterministic $(8,2)$ construction. The following example shows that this construction does not dominate query-oblivious protocols on the full unit ball. Consider a power-of-two dimension and a normalized Hadamard–sign transform. Set $x = y = e_j$. The source x still belongs to $\mathcal{X}_U$, but Uy has all d coordinates nonzero, so this query lies outside $\mathcal{Y}_{U,k}$ when $k < d$. Every entry of Uy has magnitude $1/\sqrt{d}$, and the shared scale is $1/\sqrt{d}$. Every key weight is therefore $1/d^2$. Because the eight gains are strictly decreasing, a budget of $2d$ assigns the first level to every coordinate and then the second level to every coordinate. Thus, all depths equal two.

The standardized entries of Ux are in $\{-1,1\}$. Lemma 4 gives $1.5104175806 < Q_2(1) < 1.5104175807$ and $Q_2(-1) = -Q_2(1)$. Thus, the reconstructed transformed vector is $Q_2(1)Ux$. Orthogonality gives the inner-product estimate $Q_2(1)$, whose squared error exceeds 0.26052610, independently of d.

This obstruction is larger than the worst-case error of a simple query-oblivious protocol. Quantize each original coordinate independently to the four levels

$$-1, \quad -\frac{1}{3}, \quad \frac{1}{3}, \quad 1$$

by stochastic rounding between its two neighboring levels. Denote the quantized vector by $\widetilde{X}$. Each coordinate is unbiased, the coordinate errors are independent, and adjacent levels are separated by $2/3$. Therefore

$$\mathrm{Var}(\widetilde{X}_i - X_i) \le \frac{1}{4}\left(\frac{2}{3}\right)^2 = \frac{1}{9}.$$

The encoder stores the resulting two-bit index for every coordinate, pads the remaining $6d$ cells arbitrarily, and the query-oblivious reader always fetches the first $2d$ cells. For every fixed $x, y \in \mathbb{B}_2^d$,

$$\begin{aligned}\mathbb{E}\Big[\big(y^\top\widetilde{X} - y^\top x\big)^2\Big] &= \sum_{i=1}^{d} y_i^2\,\mathrm{Var}(\widetilde{X}_i - X_i)\\ &\le \frac{1}{9}\|y\|_2^2 \le \frac{1}{9}.\end{aligned}$$

Since $1/9 < 0.260526106590\ldots$, the deployed deterministic ReadKV $(8,2)$ instantiation cannot strictly dominate all query-oblivious protocols on the full unit ball.

E.2 PREFIX-ALLOCATION RESULTS

Proposition 7 is the allocation result used by Theorem 13. The later continuous and rounding results concern the idealized geometric distortion curve. They are auxiliary comparisons and are not assumptions in any READKV guarantee.

E.2.1 EXACT INTEGER ALLOCATION

Consider N items. Item i has nonnegative weight a_i and distortion $D_i(t)$ after reading a prefix of depth $t \in \{0, \dots, W\}$. Its refinement at level s has weighted gain $a_i[D_i(s-1) - D_i(s)]$.

The proof uses two subclaims. A depth vector is equivalent to a prefix-closed set of refinements, and its objective equals the depth-zero objective minus the selected gains. Moreover, among all sets of a fixed size, a maximum-gain set can be chosen prefix-closed: whenever it contains a refinement but omits a predecessor from the same item, replacing the refinement by that predecessor cannot decrease total gain. The constrained problem therefore reduces to merging nonincreasing gain lists.

The result below is an instance of the classical marginal-allocation principle (Fox, 1966; Segall, 1976; Shoham & Gersho, 1988). That principle allocates equal-cost resources by repeatedly taking the largest available marginal return when each item's returns decrease with additional allocation. Proposition 7 states this rule in the prefix-depth notation used by READKV.

Proposition 7 (Exact Integer Allocation)**.** *Let $a_1, \dots, a_N \geq 0$, and let W and B be nonnegative integers. Suppose refinements have unit cost and, for every item i, the gains $D_i(s-1) - D_i(s)$ are nonnegative and nonincreasing in $s = 1, \dots, W$. Repeatedly selecting the available next refinement with largest weighted gain globally minimizes*

$$\sum_{i=1}^{N} a_i D_i(t_i) \quad \textit{over} \quad t_i \in \{0, \dots, W\}, \qquad \sum_{i=1}^{N} t_i \leq B.$$

The procedure stops when the budget is exhausted, all refinements have been selected, or no positive gain remains.

Proof. For each item and level, write $g_{i,s} = a_i[D_i(s-1) - D_i(s)]$ for $s = 1, \dots, W$. A feasible depth vector t selects the prefix-closed set $S(t) = \{(i, s) : 1 \leq s \leq t_i\}$. Its objective telescopes as

$$\sum_i a_i D_i(t_i) = \sum_i a_i D_i(0) - \sum_{(i,s) \in S(t)} g_{i,s}.$$

It is therefore enough to maximize selected gain over prefix-closed sets of cardinality at most B.

Fix a cardinality k. Start from a set of k refinements with maximum total gain, without imposing prefix closure. If it contains (i, s) but omits a predecessor (i, u) with $u < s$, replace (i, s) by the smallest missing predecessor. Since the gain list for item i is nonincreasing, this replacement does not reduce total gain. Repeating the operation terminates with a prefix-closed maximum-gain set. Thus prefix closure does not reduce the best total gain at any fixed cardinality.

Now suppose the greedy algorithm has selected depths $t_1, \dots, t_N$. For an item with $t_i < W$, its next available gain is g_{i,t_i+1}. Every deeper unselected gain from that item is no larger. Consequently, a largest available gain is also a largest gain among all refinements not yet selected. Induction on the number of selections shows that after k steps, greedy has the maximum total gain among all sets of k refinements, and hence among all feasible prefix-closed sets of that size.

All gains are nonnegative. Therefore, an optimum with at most B refinements either uses the full available budget or leaves only zero gains unselected. The greedy procedure is optimal both when it exhausts the budget and when it stops because no positive gain remains. □

The nonincreasing-gain and unit-cost assumptions cannot be omitted from this guarantee. If a large gain can be hidden behind a smaller predecessor, or if refinements have unequal costs, selecting the largest currently available gain need not be optimal. For example, with budget two, gain lists $(0, 100)$ and $(1, 1)$ make the current-gain rule obtain total gain two, whereas the optimum is 100. With unequal costs, a rule based only on gain would select one cost-two refinement of gain three, although two cost-one refinements of gain two each have larger total gain.

E.2.2 CONTINUOUS GEOMETRIC ALLOCATION

The deployed reader uses the integer rule above. To compare integer and fractional depth budgets, the following relaxation uses the idealized geometric curve $D(t) = 4^{-t}$. Define

$$C_W(B) = \min_{\substack{0 \le t_i \le W \\ \sum_i t_i \le B}} \sum_{i=1}^{N} a_i 4^{-t_i}.$$

The next proposition is the finite-cap form of reverse water-filling (Cover & Thomas, 2006). Standard reverse water-filling equalizes the remaining weighted distortion across active items. The finite depth constraint clips this allocation: sufficiently small weights remain unread, while sufficiently large weights receive the full depth W.

Proposition 8 (Finite-Cap Continuous Allocation)**.** *Let $a_1, \dots, a_N \ge 0$, $W > 0$, $B \ge 0$, and $P = \{i : a_i > 0\}$. An optimum assigns zero depth to $i \notin P$. If $0 < B < W|P|$, the optimal depths on P are unique and satisfy*

$$t_i^\star = \min\{W, \max\{0, \log_4(a_i/\tau)\}\}, \qquad \sum_i t_i^\star = B,$$

for some $\tau > 0$, which need not be unique. If $B = 0$, all depths are zero. If $B \ge W|P|$, every positive-weight item has depth W.

Proof. Write $P = \{i : a_i > 0\}$. A zero-weight item does not affect the objective, so we choose the canonical optimum that assigns it depth zero. If P is empty, this already proves the proposition. If $B = 0$, feasibility fixes every depth at zero. If $B \ge W|P|$, assigning depth W to every positive-weight item minimizes each term separately. These observations cover the boundary cases in Proposition 8.

Suppose now that $0 < B < W|P|$. The feasible set is compact, so an optimum exists. Increasing an unsaturated positive-weight depth strictly decreases the objective. Not all positive-weight items can be saturated, so the budget constraint binds. On the coordinates in P, the objective is strictly convex because

$$\frac{\mathrm{d}^2}{\mathrm{d}t^2}(a_i 4^{-t}) = (\ln 4)^2 a_i 4^{-t} > 0.$$

Consequently, the positive-weight part of the optimum is unique.

The problem is convex and has a point strictly inside the box constraints. The Karush–Kuhn–Tucker conditions are therefore sufficient. Attach a multiplier $\lambda \ge 0$ to the total budget. For an interior item, stationarity gives

$$-\ln(4) a_i 4^{-t_i} + \lambda = 0.$$

The budget binds, and at least one positive-weight item is unsaturated. Thus $\lambda > 0$. If $\lambda = 0$, the negative derivative of that unsaturated coordinate cannot satisfy its one-sided optimality condition. Putting $\tau = \lambda / \ln 4$ gives $t_i = \log_4(a_i/\tau)$ on an interior item. The one-sided optimality conditions are equally explicit:

$$t_i = 0 \implies a_i \le \tau, \qquad t_i = W \implies a_i \ge 4^W \tau.$$

Together with the interior equation, these conditions give exactly the clipped formula in Proposition 8.

For $\tau > 0$, let

$$H(\tau) = \sum_{i \in P} \min\{W, \max\{0, \log_4(a_i/\tau)\}\}.$$

This function is continuous and nonincreasing. It satisfies $H(\tau) \to W|P|$ as $\tau \downarrow 0$ and $H(\tau) \to 0$ as $\tau \uparrow \infty$, so it attains the value B. The resulting clipped vector satisfies all KKT conditions. It is the unique positive-weight optimum, which proves Proposition 8. The multiplier need not be unique because every item may be clipped over an interval of thresholds. For example, $a = (64, 1)$, $W = B = 1$ has the unique allocation $(1, 0)$ for every $\tau \in [1, 16]$. □

E.2.3 Integer Rounding for Geometric Distortion

For nonnegative integers W and B, define the integer optimum

$$I_W(B) = \min_{\substack{t_i \in \{0,\dots,W\} \\ \sum_i t_i \le B}} \sum_{i=1}^{N} a_i 4^{-t_i}.$$

To compare this optimum with its continuous relaxation, start from a continuous optimizer $t_i^\star = \ell_i + f_i$. Since the total budget is an integer, the fractional parts sum to an integer. A dependent rounding step chooses exactly that many coordinates to round upward while preserving every marginal probability $\mathbb{P}(T_i = \ell_i + 1) = f_i$ (**?**). The rounded objective can then be bounded one coordinate at a time by $(1 - 3f_i/4)4^{f_i}$.

Proposition 9 (Integer Rounding Bound)**.** *Let W and B be nonnegative integers, and let $a_1, \dots, a_N \ge 0$. Then*

$$C_W(B) \le I_W(B) \le \kappa_{\mathrm{int}} C_W(B), \qquad \kappa_{\mathrm{int}} = \max_{0 \le f \le 1} (1 - 3f/4)4^f = 1.263740721\ldots < 1.264.$$

Proof. Let $P = \{i : a_i > 0\}$. If P is empty, if $B = 0$, or if $B \ge W|P|$, the continuous optimum already has integer depths on every positive-weight item, and $C_W(B) = I_W(B)$. It remains to consider $0 < B < W|P|$.

Let $t^\star$ be the continuous optimum and write $t_i^\star = \ell_i + f_i$, where $\ell_i \in \{0, \dots, W\}$ and $0 \le f_i < 1$. The budget binds, so $\sum_i t_i^\star = B$. Since both B and $\sum_i \ell_i$ are integers, $K = \sum_i f_i$ is also an integer. We next construct a random vector $Z \in \{0,1\}^N$ such that

$$\sum_i Z_i = K \quad \text{with probability one,} \qquad \mathbb{E}[Z_i] = f_i.$$

Start from the vector f. While coordinates i and j are both strictly fractional, define

$$\alpha = \min\{1 - f_i, f_j\}, \qquad \beta = \min\{f_i, 1 - f_j\}.$$

Replace (f_i, f_j) by $(f_i + \alpha, f_j - \alpha)$ with probability $\beta/(\alpha + \beta)$ and by $(f_i - \beta, f_j + \beta)$ with probability $\alpha/(\alpha + \beta)$. The pair's sum is unchanged. Each coordinate preserves its conditional expectation, and at least one becomes integral in either outcome. Repeating the operation terminates after at most $N - 1$ steps. Because the total is the integer K, the final vector cannot have exactly one fractional coordinate. Iterated expectation now gives the claimed properties of Z.

The rounded depths $T_i = \ell_i + Z_i$ respect both the exact total budget and the write cap: if $f_i > 0$, then $\ell_i \le W - 1$. For each coordinate with $a_i > 0$,

$$\frac{\mathbb{E}[a_i 4^{-T_i}]}{a_i 4^{-t_i^\star}} = (1 - f_i + f_i/4)4^{f_i} \le \kappa_{\mathrm{int}}$$

because $\mathbb{P}(Z_i = 1) = f_i$. Summing over i gives

$$\mathbb{E}\left[\sum_i a_i 4^{-T_i}\right] \le \kappa_{\mathrm{int}} C_W(B).$$

At least one realization is no larger than its expectation, so a feasible integer allocation has objective at most $\kappa_{\mathrm{int}} C_W(B)$; the optimal integer allocation can only improve it.

To evaluate the constant, differentiate $h(f) = (1 - 3f/4)4^f$ on $[0, 1]$. Its only interior stationary point is $f^\star = 4/3 - 1/\ln 4$. The derivative changes from positive to negative there, while $h(0) = h(1) = 1$, so $\kappa_{\mathrm{int}} = h(f^\star) = 1.263740721\ldots$. The reverse inequality $C_W(B) \le I_W(B)$ follows because every integer allocation is also continuous-feasible. □

The rounding is only a comparison between the geometric continuous and integer objectives. The implementation instead applies Proposition 7 directly to the measured codec curve.

E.3 Attention-Output Error Analysis

The worst-case formulation asks what any legal stored representation can achieve. The allocation model addresses the narrower operational problem of selecting refinements for the current query from our representation. The results below connect the allocation scores to attention error without requiring an independent query distribution. The deterministic bound retains explicit calibration and residual-correlation factors; without controlling those factors, the surrogate alone gives no uniform worst-case guarantee.

E.3.1 Attention-Output Error Decomposition

Fix one query head with query $q \in \mathbb{R}^{d_h}$ and T cached keys and values. We work in the transformed value coordinates used by ReadKV; the inverse value transform is orthogonal and preserves the output norm. Let $V, \widehat{V} \in \mathbb{R}^{T\times d_h}$ contain the exact and reconstructed transformed values as rows, and put $E_V = V - \widehat{V} \in \mathbb{R}^{T\times d_h}$. For exact and reconstructed keys $k_j, \widehat{k}_j \in \mathbb{R}^{d_h}$, define $z, \widehat{z} \in \mathbb{R}^T$ by $z_j = q^\top k_j/\sqrt{d_h}$ and $\widehat{z}_j = q^\top \widehat{k}_j/\sqrt{d_h}$. Their attention vectors are $\alpha = \mathrm{softmax}(z) \in [0,1]^T$ and $\widehat{\alpha} = \mathrm{softmax}(\widehat{z}) \in [0,1]^T$. Let $\mathbf{1}_T \in \mathbb{R}^T$ denote the all-ones vector.

The proof has two steps. Adding and subtracting $V^\top\widehat{\alpha}$ separates key-induced attention error from value reconstruction error. The softmax Jacobian then gives $\|\alpha - \widehat{\alpha}\|_2 \le \|z - \widehat{z}\|_2/2$, which converts the first term into a logit-error bound.

Lemma 10 (Attention-Output Error Decomposition)**.** *For the exact output $o = V^\top\alpha \in \mathbb{R}^{d_h}$ and reconstructed output $\widehat{o} = \widehat{V}^\top\widehat{\alpha} \in \mathbb{R}^{d_h}$,*

$$o - \widehat{o} = V^\top(\alpha - \widehat{\alpha}) + E_V^\top\widehat{\alpha}. \tag{4}$$

Moreover, for every $c_0 \in \mathbb{R}^{d_h}$ and $V_{c_0} = V - \mathbf{1}_T c_0^\top$,

$$\|o - \widehat{o}\|_2 \le \tfrac{1}{2}\|V_{c_0}\|_{2\to 2}\|z - \widehat{z}\|_2 + \|E_V^\top\widehat{\alpha}\|_2. \tag{5}$$

Here $\|V_{c_0}\|_{2\to 2}$ is its operator norm: its largest amplification of a vector's Euclidean length.

Proof. The decomposition is the algebraic identity

$$V^\top\alpha - \widehat{V}^\top\widehat{\alpha} = V^\top(\alpha - \widehat{\alpha}) + (V - \widehat{V})^\top\widehat{\alpha}.$$

Because both probability vectors sum to one, $\mathbf{1}_T^\top(\alpha - \widehat{\alpha}) = 0$, and hence $V^\top(\alpha - \widehat{\alpha}) = V_{c_0}^\top(\alpha - \widehat{\alpha})$.

It remains to bound the change in probabilities. The softmax Jacobian at probability vector p is $J(p) = \mathrm{diag}(p) - pp^\top$. Its absolute row sum in row j is

$$p_j(1 - p_j) + \sum_{k\neq j} p_j p_k = 2p_j(1 - p_j) \le \frac{1}{2}.$$

The matrix is symmetric, so $\|J(p)\|_{2\to 2} \le \|J(p)\|_{\infty\to\infty} \le 1/2$. Integrating the Jacobian along the line segment between z and $\widehat{z}$ gives

$$\|\alpha - \widehat{\alpha}\|_2 \le \frac{1}{2}\|z - \widehat{z}\|_2.$$

For a fixed attention mask, the same argument applies after restricting the two logit vectors to their unmasked coordinates. The triangle inequality and $\|V_{c_0}^\top u\|_2 \le \|V_{c_0}\|_{2\to 2}\|u\|_2$ now prove Equation (5). The constant $1/2$ is sharp: at $p = (1/2, 1/2)$, the vector $(1, -1)$ is an eigenvector of $J(p)$ with eigenvalue $1/2$. See also the general sharp treatment of Nair (2025). □

In Equation (4), the first term is the change in attention weights caused by key error. The second is value reconstruction error weighted by the probabilities observed by the value planner. Both terms concern the same fixed key–query pair. Equation (5) is deterministic, but it does not imply that every reduction in the surrogate improves perplexity.

E.3.2 Diagonal Approximation

Let $e^K_{g,j,c}$ be the key reconstruction residual in transformed channel c. The exact squared logit error for query head h is

$$\frac{1}{d_h}\left(\sum_c q'_{h,c} e^K_{g(h),j,c}\right)^2.$$

Expanding the square produces diagonal terms $(q'_{h,c})^2(e^K_{g(h),j,c})^2$ and cross terms between different channels. We approximate the residual second moment at depth t by $(\sigma^K_{g,c})^2 D(t)$ and retain the diagonal terms. Summing over cached tokens and sibling query heads then gives

$$\frac{T}{d_h}\sum_{g,c}\left[\sum_{h\in\mathcal{S}(g)}(q'_{h,c}\sigma^K_{g,c})^2\right] D(t^K_{g,c}).$$

The common factor T/d_h does not affect the allocation. The bracketed expression is the key score in Equation (1).

For values, consider the exact second term in Equation (4). It satisfies

$$\left\|\sum_j \widehat{\alpha}_{h,j} e^V_{g,j}\right\|_2^2 = \sum_j \widehat{\alpha}^2_{h,j}\|e^V_{g,j}\|_2^2 + 2\sum_{j<k}\widehat{\alpha}_{h,j}\widehat{\alpha}_{h,k}\langle e^V_{g,j}, e^V_{g,k}\rangle.$$

Retaining the diagonal terms and using the physical-scale estimate $\|e^V_{g,j}\|_2^2 \approx [\sum_c (\sigma^V_{g,c})^2] D(t^V_{g,j})$ gives the scale-aware value score in Equation (2). The language-model experiments instead set the head factor to one and optimize standardized residual error, as stated in the main text.

These expansions explain why query coordinates are squared for keys and attention probabilities are squared for values. They also identify the two approximation errors: the calibration curve may misestimate residual second moments, and the cross terms may matter. Neither expansion assumes that keys and queries are independent. A deterministic Lloyd codec also does not make the cross terms vanish. The exact allocation theorem concerns the resulting separable objective. The language-model experiments test the quality obtained from that objective.

E.3.3 Deterministic Attention-Error Bound

The omitted terms can be represented exactly by residual Gram matrices. Apply the shared orthogonal key transform and write $q' = U_K q$, $k'_j = U_K k_j$, and $\widehat{k}'_j = U_K \widehat{k}_j$. Orthogonality preserves the logits, so $z_j - \widehat{z}_j = (q')^\top(k'_j - \widehat{k}'_j)/\sqrt{d_h}$. Let $E_K \in \mathbb{R}^{T\times d_h}$ have rows $e^K_j = k'_j - \widehat{k}'_j$. Retain the transformed-value residual matrix $E_V = V - \widehat{V}$ from Lemma 10. Define the realized diagonal energies

$$A_K = \frac{1}{d_h}\sum_{j=1}^{T}\sum_{c=1}^{d_h}(q'_c e^K_{j,c})^2, \qquad A_V = \sum_{j=1}^{T}\widehat{\alpha}^2_j\|e^V_j\|_2^2.$$

For the selected key depths t^K_c and value depths t^V_j, define the corresponding calibrated objectives, including the factors omitted during allocation,

$$S_K = \frac{T}{d_h}\sum_{c=1}^{d_h}(q'_c\sigma^K_c)^2 D(t^K_c), \qquad S_V = s\sum_{j=1}^{T}\widehat{\alpha}^2_j D(t^V_j).$$

Here $s = \sum_c(\sigma^V_c)^2$ gives the physical-scale objective, while $s = 1$ gives the standardized objective used in the language-model experiments. Define the calibration ratios by

$$\gamma_K = \begin{cases} A_K/S_K, & S_K > 0, \\ 0, & A_K = S_K = 0, \\ +\infty, & S_K = 0 < A_K, \end{cases} \qquad \gamma_V = \begin{cases} A_V/S_V, & S_V > 0, \\ 0, & A_V = S_V = 0, \\ +\infty, & S_V = 0 < A_V. \end{cases}$$

Thus $A_K = \gamma_K S_K$ and $A_V = \gamma_V S_V$ whenever the ratios are finite, with the convention $0 \cdot 0 = 0$. These factors isolate calibration mismatch from residual correlation. Any finite upper bounds on the canonical ratios may be substituted in Proposition 11.

Put $G_K = E_K^\top E_K$. On the coordinates with $(G_K)_{cc} > 0$, define the normalized Gram matrix

$$(C_K)_{cc'} = \frac{(G_K)_{cc'}}{\sqrt{(G_K)_{cc}(G_K)_{c'c'}}}, \qquad \kappa_K = \lambda_{\max}(C_K).$$

If $E_K = 0$, set $C_K = [1]$ and $\kappa_K = 1$ by convention. Similarly, put $G_V = E_V E_V^\top$. On the rows with $(G_V)_{jj} > 0$, define

$$(C_V)_{jk} = \frac{(G_V)_{jk}}{\sqrt{(G_V)_{jj}(G_V)_{kk}}}, \qquad \kappa_V = \lambda_{\max}(C_V).$$

If $E_V = 0$, set $C_V = [1]$ and $\kappa_V = 1$ by convention.

The proof establishes three subclaims. Normalizing the key residual Gram matrix gives $\|z - \widehat{z}\|_2^2 \le \kappa_K A_K$. Normalizing the value residual Gram matrix gives $\|E_V^\top \widehat{\alpha}\|_2^2 \le \kappa_V A_V$. The calibration ratios then replace A_K, A_V by S_K, S_V, and Lemma 10 combines the two bounds.

Proposition 11 (Correlation-Aware Attention Bound)**.** *For every fixed query, cache, selected depth schedule, and $c_0 \in \mathbb{R}^{d_h}$ for which γ_K and γ_V are finite,*

$$\|o - \widehat{o}\|_2^2 \le \frac{1}{2}\|V - \mathbf{1}_T c_0^\top\|_{2\to 2}^2 \kappa_K \gamma_K S_K + 2\kappa_V \gamma_V S_V.$$

The statement is deterministic. It permits arbitrary dependence among the query, keys, values, and residuals.

Proof. Coordinates outside the support used to define C_K are zero columns of E_K and contribute nothing. On the remaining coordinates, set $u_c = q'_c \sqrt{(G_K)_{cc}}$. The normalized Gram definition gives

$$\frac{\|u\|_2^2}{d_h} = A_K \quad \text{and} \quad \|z - \widehat{z}\|_2^2 = \frac{1}{d_h}(q')^\top G_K q' = \frac{1}{d_h} u^\top C_K u \le \kappa_K A_K \le \kappa_K \gamma_K S_K.$$

For values, set $v_j = \widehat{\alpha}_j \sqrt{(G_V)_{jj}}$ on the nonzero residual rows. The same argument yields

$$\|v\|_2^2 = A_V \quad \text{and} \quad \|E_V^\top \widehat{\alpha}\|_2^2 = \widehat{\alpha}^\top G_V \widehat{\alpha} \le \kappa_V A_V \le \kappa_V \gamma_V S_V.$$

Apply Equation (5) and then $(a+b)^2 \le 2a^2 + 2b^2$. The coefficient of the key term is $2(1/2)^2 = 1/2$, which proves the claim. □

For a physical KV head shared by several query heads, apply the proposition to each query head and sum the resulting inequalities. Taking the largest key coefficient and value coefficient across the sibling heads factors out the sums $\sum_h S_{K,h}$ and $\sum_h S_{V,h}$. These are exactly the grouped objectives produced by Equations (1) and (2).

The Gram factors admit a concrete bound when pairwise residual correlations are controlled.

Corollary 12 (Calibrated Coherence Bound)**.** *Suppose C_K has size m_K and every off-diagonal entry has magnitude at most μ_K. Define m_V and μ_V analogously for C_V. Suppose also that the realized residuals satisfy*

$$\sum_{j=1}^{T} (e_{j,c}^K)^2 \le T(\sigma_c^K)^2 D(t_c^K) \quad \textit{for every } c, \qquad \|e_j^V\|_2^2 \le s D(t_j^V) \quad \textit{for every } j.$$

Then, for every center $c_0 \in \mathbb{R}^{d_h}$,

$$\|o - \widehat{o}\|_2^2 \le \frac{1}{2}\|V - \mathbf{1}_T c_0^\top\|_{2\to 2}^2 [1 + (m_K - 1)\mu_K] S_K + 2[1 + (m_V - 1)\mu_V] S_V.$$

Proof. Each normalized Gram matrix has unit diagonal. Gershgorin's theorem gives $\kappa_K \le 1 + (m_K - 1)\mu_K$ and $\kappa_V \le 1 + (m_V - 1)\mu_V$. The two residual envelopes imply $A_K \le S_K$ and $A_V \le S_V$. Hence the canonical ratios satisfy $\gamma_K \le 1$ and $\gamma_V \le 1$. Substitute these inequalities and the coherence bounds into Proposition 11. □

The dependence on residual correlation cannot be removed. For any $m \ge 1$, take one key-residual row with m unit entries and $q' = \mathbf{1}_m/\sqrt{m}$, with $d_h = m$. Then $\|z - \widehat{z}\|_2^2 = 1$ but $A_K = 1/m$. Likewise, take m identical unit-norm value-residual rows and $\widehat{\alpha} = \mathbf{1}_m/m$. Their weighted sum has squared norm one, while $A_V = 1/m$. Thus both exact-to-diagonal ratios can equal m, so no dimension-independent constant can replace the correlation factors.

Proposition 11 provides a valid deterministic certificate from the allocation objectives to attention-output error. Its strength depends on explicit, auditable quantities. The γ factors measure how well calibration predicts realized diagonal error, while the κ factors measure the amplification from omitted cross terms. The proposition does not assert that these factors are uniformly small for every language-model cache.

E.4 Stagewise Allocation and Attention-Error Certificate

We now analyze Algorithm 1 on an arbitrary fixed cache and query. Fix positive integers T, d_h, G, H and the query-head-to-KV-head map $g : \{1, \dots, H\} \to \{1, \dots, G\}$ from Section 3. The result has two parts. Each stage chooses the best legal prefix depths for its calibrated objective. Those optimized objectives then control the resulting attention-output error through explicit factors for calibration mismatch and residual correlation.

A *depth schedule* assigns an integer prefix depth to every allocation item. The legal key schedules use depths $t^K_{g,c}$ for physical head g and transformed channel c. The legal value schedules use depths $t^V_{g,j}$ for physical head g and cached token j. Their feasible sets are

$$\mathcal{T}_K(B_K) = \left\{ t^K \in \{0, \dots, W\}^{G \times d_h} : \sum_{g,c} t^K_{g,c} \le B_K \right\}, \qquad \mathcal{T}_V(B_V) = \left\{ t^V \in \{0, \dots, W\}^{G \times T} : \sum_{g,j} t^V_{g,j} \le B_V \right\}.$$

Using the importance scores from Equations (1) and (2), define the calibrated objectives

$$S_K(t^K) = \frac{T}{d_h} \sum_{g,c} a^K_{g,c} D(t^K_{g,c}), \qquad S_V(t^V; \widehat{\alpha}) = \sum_{g,j} a^V_{g,j}(\widehat{\alpha}) D(t^V_{g,j}).$$

S_K is the diagonal approximation to squared logit error, aggregated over the query heads that share each physical KV head. S_V is the corresponding calibrated value objective: it uses physical value scales when $s_g = \sum_c (\sigma^V_{g,c})^2$, and standardized units when $s_g = 1$. The calibration ratios below account for either choice. The factor T/d_h is constant across key schedules and therefore does not change the key allocation.

For later use, separate these grouped objectives by query head:

$$S_{K,h}(t^K) = \frac{T}{d_h} \sum_c (q'_{h,c} \sigma^K_{g(h),c})^2 D(t^K_{g(h),c}), \qquad S_{V,h}(t^V; \widehat{\alpha}) = s_{g(h)} \sum_j \widehat{\alpha}^2_{h,j} D(t^V_{g(h),j}).$$

The definitions of the grouped importance scores imply $S_K(t^K) = \sum_h S_{K,h}(t^K)$ and $S_V(t^V; \widehat{\alpha}) = \sum_h S_{V,h}(t^V; \widehat{\alpha})$.

The exact attention error can differ from these diagonal objectives for two reasons. First, calibration may mispredict the realized diagonal residual energy. For each query head h, $\gamma_{K,h}$ and $\gamma_{V,h}$ are the ratios between that realized energy and its calibrated prediction. Second, residuals from different key channels or value tokens may be correlated. A *residual Gram matrix* records their pairwise inner products. The factors $\kappa_{K,h}$ and $\kappa_{V,h}$ are the largest eigenvalues of the corresponding normalized Gram matrices. They measure the amplification caused by the omitted cross terms. Appendix E.3.3 gives the exact formulas, including the zero-residual conventions.

For query head h, let $V_{g(h)} \in \mathbb{R}^{T \times d_h}$ be the matrix whose rows are the full-precision transformed values in its physical KV head, and let $\mathbf{1}_T \in \mathbb{R}^T$ be the all-ones vector. For any transformed center

$c_h \in \mathbb{R}^{d_h}$, subtracting c_h from every row does not change the output error caused by a change in attention probabilities, because both probability vectors sum to one. The notation $\|\cdot\|_{2\to 2}$ denotes the operator norm, or the largest Euclidean amplification of a matrix.

The theorem is proved through three subclaims. First, diminishing refinement gains make the marginal-gain rule minimize S_K over every legal key schedule. Second, once that schedule fixes $\widehat{\alpha}_{\text{READKV}}$, the same argument minimizes $S_V(\,\cdot\,;\widehat{\alpha}_{\text{READKV}})$ over every legal value schedule. Third, Proposition 11 bounds each $\|o_h - \widehat{o}_h\|_2^2$ by a linear combination of $S_{K,h}$ and $S_{V,h}$. Summing those bounds gives the final certificate. Its coefficients account for calibration mismatch, residual correlation, and the centered-value operator norm.

Lemma 4 in Appendix D verifies that the deployed depth-eight Lloyd tree has decreasing positive gains $\Delta_1 > \cdots > \Delta_8 > 0$, where $\Delta_t = D(t-1) - D(t)$. Hence, the diminishing-gain hypothesis and the exact stagewise allocation conclusions below apply to the codec used in the experiments.

Theorem 13 (READKV Allocation and Attention-Error Certificate)**.** *Fix an attention layer and query as above. Let $W \geq 1$ and $B_K, B_V \geq 0$ be integers. Let $D(t) \geq 0$ for $t = 0, \ldots, W$, and suppose the scalar distortion gains $D(t-1) - D(t)$ are nonnegative and nonincreasing for $t = 1, \ldots, W$. Let t^K_{READKV} be the key schedule returned by Algorithm 1, let $\widehat{\alpha}_{\text{READKV}}$ be the attention computed from the resulting key reconstruction, and let t^V_{READKV} be the value schedule returned in the second stage. Then*

$$S_K(t^K_{\text{READKV}}) = \min_{t^K \in \mathcal{T}_K(B_K)} S_K(t^K), \qquad S_V(t^V_{\text{READKV}}; \widehat{\alpha}_{\text{READKV}}) = \min_{t^V \in \mathcal{T}_V(B_V)} S_V(t^V; \widehat{\alpha}_{\text{READKV}}).$$

Let $o_h, \widehat{o}_h \in \mathbb{R}^{d_h}$ be the exact and reconstructed outputs for query head h. Their distance can be evaluated in transformed value coordinates because U_V is orthogonal. Choose any center $c_h \in \mathbb{R}^{d_h}$, and let $\kappa_{K,h}, \kappa_{V,h}$ and $\gamma_{K,h}, \gamma_{V,h}$ be the residual-correlation and calibration factors for this head, as defined precisely in Appendix E.3.3. Evaluate these factors at the READKV *reconstructions and define the worst per-head conversion coefficients*

$$\overline{\beta}_K = \max_h \frac{1}{2}\|V_{g(h)} - \mathbf{1}_T c_h^\top\|_{2\to 2}^2 \kappa_{K,h}\gamma_{K,h}, \qquad \overline{\beta}_V = \max_h 2\kappa_{V,h}\gamma_{V,h}.$$

For every fixed cache and query for which these factors are finite,

$$\sum_h \|o_h - \widehat{o}_h\|_2^2 \leq \overline{\beta}_K \min_{t^K \in \mathcal{T}_K(B_K)} S_K(t^K) + \overline{\beta}_V \min_{t^V \in \mathcal{T}_V(B_V)} S_V(t^V; \widehat{\alpha}_{\text{READKV}}).$$

This bound is deterministic and permits arbitrary dependence among the queries, keys, values, and quantization residuals.

Applied headwise, Corollary 12 gives a directly checkable sufficient condition for finite conversion factors. It bounds the calibration mismatch coordinatewise and the residual interactions through pairwise coherence.

The value-stage optimum is conditional on the reconstructed attention from the key stage; it is not a joint optimum of the true attention error. The conversion factors depend on the selected reconstruction, so this is an a posteriori certificate rather than an approximation ratio to the best attention-error schedule. Computing the canonical factors requires the full-precision cache as well as the reconstruction; the certificate is an offline check, not an estimate available from the fetched prefixes alone.

Proof of Theorem 13. Within either stage, item i has importance $a_i \geq 0$, depth t_i, and weighted marginal gain $a_i[D(t_i) - D(t_i + 1)]$. The assumed nonnegative, nonincreasing gains are exactly the hypotheses of Proposition 7. The legal key schedules in $\mathcal{T}_K(B_K)$ are the feasible prefix allocations in that proposition. Algorithm 1 uses the weights $a^K_{g,c}$, while S_K multiplies their objective by the positive constant T/d_h. This constant does not change the optimizer. Therefore,

$$S_K(t^K_{\text{READKV}}) = \min_{t^K \in \mathcal{T}_K(B_K)} S_K(t^K).$$

This key schedule fixes $\widehat{\alpha}_{\text{READKV}}$ and hence every value weight. Applying Proposition 7 again, now with those fixed weights, gives

$$S_V(t^V_{\text{READKV}}; \widehat{\alpha}_{\text{READKV}}) = \min_{t^V \in \mathcal{T}_V(B_V)} S_V(t^V; \widehat{\alpha}_{\text{READKV}}).$$

It remains to connect these two optimized quantities to attention error. Proposition 11 gives

$$\|o_h - \widehat{o}_h\|_2^2 \le \beta_{K,h} S_{K,h}(t^K_{\text{READKV}}) + \beta_{V,h} S_{V,h}(t^V_{\text{READKV}}; \widehat{\alpha}_{\text{READKV}}),$$

where

$$\beta_{K,h} = \tfrac{1}{2}\|V_{g(h)} - \mathbf{1}_T c_h^\top\|_{2\to 2}^2 \kappa_{K,h}\gamma_{K,h}, \qquad \beta_{V,h} = 2\kappa_{V,h}\gamma_{V,h}.$$

The per-head objectives defined above sum to the grouped objectives because the grouped importance scores sum over sibling query heads. Explicitly,

$$\sum_h S_{K,h}(t^K_{\text{READKV}}) = \frac{T}{d_h}\sum_{g,c}\left[\sum_{h\in\mathcal{S}(g)} (q'_{h,c}\sigma^K_{g,c})^2\right] D(t^K_{g,c}) = S_K(t^K_{\text{READKV}}),$$

and

$$\sum_h S_{V,h}(t^V_{\text{READKV}}; \widehat{\alpha}_{\text{READKV}}) = \sum_{g,j} s_g \left[\sum_{h\in\mathcal{S}(g)} \widehat{\alpha}^2_{h,j}\right] D(t^V_{g,j}) = S_V(t^V_{\text{READKV}}; \widehat{\alpha}_{\text{READKV}}).$$

Summing the per-head bounds and replacing each coefficient by its maximum over h proves

$$\sum_h \|o_h - \widehat{o}_h\|_2^2 \le \overline{\beta}_K S_K(t^K_{\text{READKV}}) + \overline{\beta}_V S_V(t^V_{\text{READKV}}; \widehat{\alpha}_{\text{READKV}}).$$

Substituting the two optimality equalities proves the displayed deterministic bound in the theorem. □

F Experimental Details and Results

Section 5 gives the principal C4 and LongBench tables and compares selected operating points in Figure 3. This appendix specifies the protocol and comparison methods, then adds C4 controls, shorter-prompt QA, and passkey retrieval. The remaining experiments test current-query information, intermediate precision levels, surrogate quality, and the restrictions used by the GPU reader.

F.1 Evaluation Protocol

Language Modeling. For each model, the C4 suite contains 32 calibration documents, eight development documents, and 32 disjoint test documents. A fixed rule selects one 2,048-token window from each document. The principal seed-dependent READKV comparisons use transform seeds 20260801–20260803. Every comparison retains its matched DENSE reference and document manifest.

Model weights are bf16. Both compressed and DENSE conditions use float32 attention, softmax, and final log-softmax. Queries and keys are captured after rotary position encoding and use one shared orthogonal transform. Values and reconstructed outputs use a second transform. The compressed cache is used throughout causal evaluation. READKV selects its prefixes anew for each query. Compression errors can therefore affect subsequent model states.

Question Answering and Retrieval. The LongBench evaluation uses Qasper, HotpotQA, and 2WikiMultihopQA (Bai et al., 2024), with 100 questions per task and instruction model. The same questions are evaluated at prompt limits of 3,500 and 7,500 tokens, giving paired short- and long-context conditions. We use greedy decoding, allow 128 output tokens for Qasper and 32 for the other tasks, and score the first answer line by normalized token F1, taking the maximum over reference answers.

The main protocol uses DENSE prompt prefill, which determines the first answer token, then applies each method's cache representation and read rule to subsequent decoding.

Passkey retrieval (Mohtashami & Jaggi, 2023) uses 200 five-digit keys at each context length, balanced across five insertion positions, with a 24-token output cap. A trial succeeds when the generated answer contains the key. It uses the same DENSE-prefill boundary as question answering.

Hardware and Calibration Provenance. Reference quality runs use NVIDIA L4 GPUs, while Mistral instruction-model runs and later controls use NVIDIA A10G GPUs. Each comparison retains

its original hardware and calibration cohort. In the Qwen generation study, the READKV $(4, 2)$, FULL READER $(4, 4)$, Lloyd controls, and new protected READKV rows are NVIDIA A10G follow-ups that use the NVIDIA L4 cohort's DENSE scores on the same questions. The protected Qwen rows also use a different calibration artifact and are compared with the existing NVIDIA L4 TurboQuant references. The joins verify question identifiers, prompt lengths and truncation, gold answers, and generation and scoring settings; they do not establish hardware or calibration equivalence. These cross-cohort comparisons describe operating points but do not isolate a single algorithmic change. The protected Mistral rows and their references use an NVIDIA A10G and the same calibration artifact. All latency and DRAM measurements use an NVIDIA A10G.

Aggregation and Uncertainty. For language modeling, perplexity is the exponential of token-weighted negative log likelihood (NLL). Tables average perplexity across seeds, while $\Delta\mathrm{PPL}$ uses unrounded means and matched DENSE scores. Confidence intervals are computed for NLL differences and do not represent uncertainty in the arithmetic mean of perplexity.

Bootstrap intervals resample paired documents or task examples rather than individual tokens, layers, or heads. Reported intervals use at least 10,000 replicates, and the depth and factorial controls use 20,000. Passkey resampling is stratified by insertion position. Intervals from a single transform seed capture document variation but not variation across transforms or model families.

Budget Realization. Whenever a quality method permits a configurable K/V split, we select its integer stage rates on development NLL using the primary transform seed and freeze them before testing. Test data and the remaining transform seeds do not retune the split. Requested rates are converted to nonnegative integer increment budgets over the Gd_h key items and GT value items. Ties are resolved in a fixed order while preserving prefix feasibility. Every reported schedule is checked against its requested and realized budgets.

F.2 BASELINES AND CONTROLS

Section 5 introduces the principal comparison methods. This section defines the appendix-only controls and the implementation choices that affect their quality and resource costs.

Fixed-Depth Controls. The *Lloyd-2 (norm-corrected)* control tests direct two-bit quantization without a progressive code. It uses the same transforms and per-channel standardization as READKV, but maps each scalar to a non-nested four-level Gaussian Lloyd–Max codebook (Lloyd, 1982). It stores and reads two payload bits per scalar. The encoder also stores each key norm in fp16 and uses it during reconstruction. Values are not norm-corrected. Resource costs include these norms, the calibration arrays, and the shared codebooks.

KIVI. KIVI (Liu et al., 2024) quantizes keys per channel over groups of 32 tokens and values per token over groups of 32 channels. Each group stores an fp16 scale and offset. Both the two-bit configuration and the appendix-only four-bit row retain a 128-token uncompressed residual buffer. The reported costs include the buffer and all quantization parameters.

TurboQuant. TurboQuant K4/V4 (Zandieh et al., 2026) uses its supported norm-corrected four-bit configuration. Keys are normalized, Hadamard-rotated, and quantized to 16 Lloyd–Max centroids. Values use per-token asymmetric uniform quantization. Resource costs include the retained key norms and the value scales and offsets. The serving configuration leaves the first and last two layers uncompressed, and the protected READKV C4 and QA rows use the same exemptions. The displayed (W, R) rates describe compressed layers; the reported resource costs include all dense boundary layers.

Sparse Selection. SparQ (Ribar et al., 2024) retains a 16-bit cache. It uses r query-selected key channels to estimate token relevance, then reads the complete keys and values of at most k past positions. The current token is always included, giving at most $k + 1$ positions, and a running value mean approximates omitted attention mass. Logical-read costs include the probe channels, selected tokens, and mean state.

The displayed SparQ $r32/k128$ configuration uses $r = 32$ and $k = 128$. The C4 row labeled *SparQ ($R = 2$ target)* uses $r = 16, k = 64$, except on Yi, where development selection chooses $r = 8, k = 64$. The *SparQ + nested code* control applies sparse token selection to the eight-bit

progressive code. It reads depth-two prefixes on the $r = 32$ scoring channels, followed by depth-eight keys and values for at most $k = 128$ selected tokens.

For C4, Qwen3B, Qwen14B, Yi, and DeepSeek select configurations on eight development documents from $r \in \{8, 16, 32\}$, $k \in \{32, 64, 128, 256\}$, and a fixed set of nine sparse-progressive variants. The lowest-NLL candidate within each read budget is fixed before testing. Qwen7B uses the setting selected in its original development run. Mistral uses a 12-candidate pure-SparQ sweep over the same r, k grid on eight development documents, frozen before testing. It selects $r16/k64$ and $r32/k128$ for the two read targets, reproducing the earlier transferred settings and held-out scores. This is a bounded development search, not a claim of global optimality. Mistral's hybrid configuration remains transferred. The QA and passkey studies also use fixed $r32/k128$ and hybrid configurations. Each SparQ test uses one seed.

F.3 Complete Quality Results

Tables 2 and 3 in Section 5 report all principal C4 and 7,500-token QA operating points. The tables below extend them with C4 controls and paired 3,500-token QA. They use the same format: quality change above logical reads and retained KV capacity, both as percentages of DENSE. Positive quality changes are worse.

C4 Language-Modeling Results. To the principal methods in Table 2, Table 5 adds READKV $(8, 2)^{\dagger}$, *KIVI (4-bit K/V)*, *Lloyd-2 (norm-corrected)*, *SparQ ($R = 2$ target)*, and *SparQ + nested code*. Together, these rows compare direct low-bit quantization, sparse retrieval, and a sparse reader applied to the progressive code.

Table 4: Recurring notation. Dimensions are shown when the object is a vector, matrix, or array.

Symbol	Domain or shape	Meaning
Partial-read model		
$d; x, y$	$x, y \in \mathbb{B}_2^d$	Dimension and the stored and query vectors in the worst-case inner-product problem.
$W, R; w, r$	$w = \lfloor Wd \rfloor, r = \lfloor Rd \rfloor$	Normalized storage and read rates, and their integer bit budgets.
Z	$\{0,1\}^w$	Stored binary record produced before the query is known.
θ	Public state	Code and memory-layout description fixed independently of the stored vector and query.
$\Omega_0, \Omega_e, \Omega_r$	Random variables	Public, encoder-local, and reader-local randomness, all independent of (x, y).
$\mathsf{A}, \Pi, \widehat{s}_\Pi$	Access class, protocol, scalar	Reader class, encoder–reader–decoder protocol, and its estimate of $x^\top y$.
$\mathcal{D}^{\mathrm{wc}}_{d,\mathsf{A}}(W, R)$	Nonnegative scalar	Worst-case mean-squared error optimized over protocols in A.
$\mathsf{staged}(J)$	At most J batches	Reader that may adapt each batch to the query and earlier batches.
Attention and READKV		
G, H, T, d_h	Positive integers	Numbers of physical KV heads, query heads, cached tokens, and coordinates per head.
$g(h), \mathcal{S}(g)$	$g(h) \in \{1, \dots, G\}$	Physical head used by query head h, and the query heads sharing physical head g.
$q_h, k_{g,j}, v_{g,j}, o_h$	$\mathbb{R}^{d_h}$	Query, cached key, cached value, and attention output.
$\alpha_h, \widehat{\alpha}_h$	$[0,1]^T$	Exact and reconstructed attention vectors for query head h; each sums to one.
$\widehat{\alpha}$	$[0,1]^{H \times T}$	Collection of reconstructed attention vectors across query heads.
U_K, U_V	$\mathbb{R}^{d_h \times d_h}$	Orthogonal transforms used for keys and queries, and for values.
$\widehat{K}, \widehat{V}$	$\mathbb{R}^{G \times T \times d_h}$	Collections of reconstructed transformed keys and values.
Allocation and analysis		
$D(t)$	Nonnegative scalar	Predicted standardized scalar mean-squared reconstruction error after t prefix bits.
a^K	$[0,\infty)^{G \times d_h}$	Query-dependent importance scores for transformed key channels.
$a^V(\widehat{\alpha})$	$[0,\infty)^{G \times T}$	Attention-dependent importance scores for cached value tokens.
B_K, B_V	Nonnegative integers	Group-refinement budgets: each key increment reads T bits and each value increment reads d_h bits.
t^K, t^V	$\{0, \dots, W\}^{G \times d_h}$, $\{0, \dots, W\}^{G \times T}$	Integer key-channel and value-token prefix schedules.
R_K, R_V, R	Nonnegative scalars	Budgeted mean key depth, value depth, and their average.
$\mathcal{T}_K, \mathcal{T}_V$	Sets of schedules	Feasible key and value schedules under their respective budgets.
$S_K, S_V; S_K^\star, S_V^\star$	Nonnegative scalars	Calibrated key and value objectives, and their stagewise minima.
$S_{K,h}, S_{V,h}$	Nonnegative scalars	Contributions of query head h to the grouped key and value objectives.
$\gamma_{K,h}, \gamma_{V,h}$	$[0,\infty]$	Ratios between realized diagonal residual energy and its calibrated prediction.
$\kappa_{K,h}, \kappa_{V,h}$	Nonnegative scalars	Largest eigenvalues of normalized residual Gram matrices.
$\overline{\beta}_K, \overline{\beta}_V$	Nonnegative scalars when finite	Worst-head conversion coefficients in the attention-error bound, evaluated at the selected reconstruction.
$V_{g(h)}, \mathbf{1}_T$	$\mathbb{R}^{T \times d_h}$, $\mathbb{R}^T$	Full-precision value matrix for head h and the length-T all-ones vector.

Table 5: Additional C4 language-modeling quality and KV costs on the held-out documents used in Table 2. † denotes configurations in which the first and last two layers remain uncompressed. The Qwen2.5-14B Lloyd-2 condition was not run. The low-read SparQ control uses $r16/k64$, except on Yi where it uses $r8/k64$. SparQ + nested code uses $r32/k128$.

Method (W, R)	Qwen2.5 3B	Qwen2.5 7B	Qwen2.5 14B	Yi-1.5 6B	DeepSeek LLM-7B	Mistral 7B-v0.3
DENSE PPL	11.36	10.13	8.74	8.96	8.49	7.62
Top: ΔPPL (%)						
Bottom: logical reads / retained KV capacity (% of DENSE)						
READKV $(8, 2)^{\dagger}$	+3.05 22.7 / 55.6	+1.93 25.3 / 57.2	+2.37 20.1 / 54.3	+7.82 23.8 / 56.3	+4.39 24.4 / 56.8	+8.39 23.7 / 56.3
KIVI (4-bit K/V) Liu et al., 2024	+0.07 37.6 / 35.5	+0.09 37.6 / 35.5	+0.06 37.6 / 35.5	+0.07 37.6 / 35.5	+0.10 37.6 / 35.5	+0.09 37.6 / 35.5
Lloyd-2 (norm-corrected) Lloyd, 1982	+40.99 13.1 / 13.0	+33.83 13.1 / 13.0	not run	+159.37 13.1 / 13.0	+3766.75 13.1 / 13.0	+1366.07 13.1 / 13.0
SparQ ($R = 2$ target) Ribar et al., 2024	+5.22 12.5 / 100.0	+11.57 12.5 / 100.0	+6.13 12.5 / 100.0	+291.84 9.4 / 100.0	+2.98 12.5 / 100.0	+1112.46 12.5 / 100.0
SparQ + nested code Ribar et al., 2024	+2.33 8.3 / 50.1	+2.48 8.1 / 50.1	+1.57 8.0 / 50.1	+5.55 8.1 / 50.1	+0.92 7.9 / 50.1	+196.78 8.0 / 50.1

Table 6: LongBench question-answering quality and KV costs at a 3,500-token prompt limit. The evaluation uses the same 100 questions and dense-prefill protocol as Table 3. † denotes configurations in which the first and last two layers remain uncompressed. Color marks the observed quality–read frontier: at shown precision, no other method row in the column improves loss or reads without worsening the other. Reads recur per query; retained capacity is shown but not ranked.

	Qwen2.5-7B-Instruct			Mistral-7B-Instruct-v0.3		
Method (W, R)	Qasper	HotpotQA	2Wiki MultihopQA	Qasper	HotpotQA	2Wiki MultihopQA
DENSE F1	35.51	42.49	44.87	28.10	41.42	37.46
Top: ΔF1 (points) Color: quality–read frontier Bottom: logical reads / retained KV capacity (% of DENSE)						
KIVI (2-bit K/V) Liu et al., 2024	+0.51 21.2 / 20.7	+2.59 21.2 / 20.7	+0.90 21.2 / 20.7	-0.39 21.0 / 20.7	-0.17 20.9 / 20.7	+0.32 20.9 / 20.7
TurboQuant K4/V4† Zandieh et al., 2026	+1.14 36.7 / 36.7	+1.81 36.7 / 36.7	+1.00 36.7 / 36.7	+1.01 35.4 / 35.4	+0.17 35.4 / 35.4	+0.87 35.4 / 35.4
SparQ $r32/k128$ Ribar et al., 2024	+0.52 16.4 / 100.0	+1.38 16.2 / 100.0	+1.36 16.2 / 100.0	+0.47 16.3 / 100.0	+0.21 16.3 / 100.0	+1.03 16.2 / 100.0
READKV $(4, 2)$	+2.51 12.6 / 25.1	+0.50 12.6 / 25.1	+2.05 12.6 / 25.1	-1.35 12.6 / 25.1	-0.70 12.6 / 25.1	+3.10 12.6 / 25.1
READKV $(8, 2)$	+1.30 12.6 / 50.1	-0.08 12.6 / 50.1	+1.42 12.6 / 50.1	-1.65 12.6 / 50.1	+0.14 12.6 / 50.1	+1.77 12.6 / 50.1
READKV $(8, 4)$	+0.11 25.1 / 50.1	+0.48 25.1 / 50.1	+0.07 25.1 / 50.1	+0.36 25.1 / 50.1	+0.06 25.1 / 50.1	+1.37 25.1 / 50.1
READKV $(8, 2)$†	+2.45 25.1 / 57.2	+2.01 25.1 / 57.2	+1.65 25.1 / 57.2	-1.97 23.5 / 56.3	-0.86 23.5 / 56.3	+1.81 23.5 / 56.3
READKV $(8, 4)$†	+1.01 35.8 / 57.2	+0.05 35.8 / 57.2	+0.92 35.8 / 57.2	+1.10 34.5 / 56.3	+0.06 34.5 / 56.3	+0.14 34.5 / 56.3

The low-read configurations vary more across model families than the $R = 4$ results in the main text. On the three Qwen2.5 models, READKV $(4, 2)$ stays within 4.5% of DENSE perplexity; retaining eight bits at the same read budget improves all three results. The sparse reader with the nested code uses about 8% of DENSE logical reads and 50% of its retained KV capacity. Its ΔPPL is below 5.56% on five of the six models. The transferred Mistral hybrid setting is less accurate; it was not retuned in the pure-SparQ sweep. Direct two-bit Lloyd quantization and low-read SparQ also incur large losses on some models. At the low-read target, SparQ is more accurate on DeepSeek than READKV $(8, 2)$ (2.98% versus 6.27% PPL increase), while retaining about twice the cache capacity. The four-bit KIVI control has at most 0.10% PPL increase across the six models, at a higher logical-read rate than READKV $(8, 4)$.

LongBench Question-Answering Results. Each of the six model–dataset conditions uses the same 100 questions across methods and prompt limits. To test dependence on prompt length, Table 6 reports the paired 3,500-token condition for the methods shown at 7,500 tokens in Table 3.

At the 3,500-token limit, READKV $(8, 2)$ has ΔF1 between -1.65 and 1.78 points while using about 12.6% of DENSE logical reads and 50.1% of its retained KV capacity. At the same prompt limit, the largest loss for READKV $(8, 4)$ is below 1.38 points.

The † rows cover all 24 protected READKV conditions: two read rates, two models, three datasets, and both prompt limits. Across these conditions, the largest losses are below 2.46 F1 points for $(8, 2)^\dagger$ and 1.11 points for $(8, 4)^\dagger$. Protected $(8, 4)$ scores higher than TurboQuant in seven of twelve conditions, with a mean difference of -0.04 F1 points (READKV minus TurboQuant); every paired 95% interval includes zero. These intervals do not establish equivalence. Protected $(8, 2)$ reads about one third less than TurboQuant but scores lower in eight of twelve conditions, with a mean difference of -0.51 F1 points; the quality gap is larger at the 7,500-token limit. Retained capacity is about 56–57% of DENSE, versus about 35–37% for TurboQuant. The Qwen cross-cohort qualification above applies to these comparisons.

F.4 PASSKEY RETRIEVAL

Passkey retrieval tests whether compressed decoding preserves a specific piece of information over a long context. Several methods nearly saturate the task, so Table 7 serves as a retention check rather than the primary quality comparison.

Table 7: Passkey retrieval quality and KV costs at 4K and 8K context lengths, with 200 trials per condition. Every method uses dense prompt prefill. † denotes configurations in which the first and last two layers remain uncompressed. SparQ + nested code uses $r32/k128$. READKV $(4, 2)$ recovers all keys with about half the logical reads of the matched-storage FULL READER $(4, 4)$. Color marks the observed quality–read frontier: at shown precision, no other method row in the column improves recovered keys or reads without worsening the other (more recovered keys and fewer reads are better). Reads recur per query; retained capacity is shown but not ranked.

	Qwen2.5-7B-Instruct		Mistral-7B-Instruct-v0.3	
Method (W, R)	4K	8K	4K	8K
DENSE recovered	200	200	200	200
Top: recovered keys (out of 200) Color: quality–read frontier Bottom: logical reads / retained KV capacity (% of DENSE)				
FULL READER $(4, 4)$	200 25.1 / 25.1	200 25.1 / 25.0	200 25.1 / 25.0	200 25.0 / 25.0
READKV $(4, 2)$	200 12.6 / 25.1	200 12.6 / 25.0	200 12.6 / 25.0	200 12.5 / 25.0
READKV $(8, 2)$	200 12.6 / 50.1	200 12.6 / 50.0	200 12.6 / 50.0	200 12.5 / 50.0
READKV $(8, 4)$	200 25.1 / 50.1	200 25.1 / 50.0	200 25.1 / 50.0	200 25.0 / 50.0
KIVI (2-bit K/V) Liu et al., 2024	200 21.0 / 21.3	200 19.9 / 20.0	199 21.1 / 21.3	200 19.8 / 20.0
TurboQuant K4/V4† Zandieh et al., 2026	200 36.7 / 36.7	200 36.7 / 36.7	200 35.4 / 35.4	200 35.4 / 35.4
SparQ $r32/k128$ Ribar et al., 2024	200 15.8 / 100.0	197 14.1 / 100.0	200 15.8 / 100.0	200 14.1 / 100.0
SparQ + nested code Ribar et al., 2024	199 3.3 / 50.1	200 2.4 / 50.0	67 3.3 / 50.1	0 2.4 / 50.0

The quarter-capacity READKV $(4, 2)$ reader recovers all 200 keys in every condition while using about half the logical reads of the matched-storage FULL READER $(4, 4)$. The flexible reader recovers all 200 keys in every condition with both READKV $(8, 2)$ and READKV $(8, 4)$. The READKV $(8, 2)$ configuration uses 12.5–12.6% of DENSE logical reads while retaining about 50% of the KV capacity. KIVI nearly saturates the task with a smaller retained cache but higher logical reads, while SparQ retains the full cache. The nested-code sparse hybrid is less reliable on Mistral, recovering 67/200 keys at 4K and none at 8K. Its favorable C4 results on other families therefore do not imply reliable retrieval across models and tasks.

F.5 CURRENT-QUERY ALLOCATION

To determine whether the current query improves allocation, we compare three readers built from the same progressive codec. Uniform reading stores and reads its depth-two prefix. Calibration-only and current-query reading retain the same eight-bit code: the former fixes importance before testing, while the latter allocates key depth from the query and value depth from the reconstructed attention.

Table 8 holds the codec and calibration fixed but compares complete policies with different retained depths and development-selected K/V splits. On Qwen and Yi, the current-query reader uses $(R_K, R_V) = (3, 1)$, while the other policies use $(2, 2)$. DeepSeek uses $(2, 2)$ throughout. The table therefore measures the full policy difference. A paired ablation below holds the split fixed to isolate current-query key allocation.

Table 8: C4 perplexity under three allocation policies at mean payload depth $R = 2$. Entries are arithmetic means over three transform seeds on the same 32 test documents. Calibration-only and current-query allocation retain the same $W = 8$ progressive code. Uniform reading retains a depth-two code. The policies use their development-selected K/V splits, so the table compares complete policies rather than isolating one factor.

Allocation	Qwen3B	Qwen7B	Yi6B	DeepSeek7B
Uniform depth 2	21.66	19.91	39.34	568.35
Calibration-only	19.11	14.67	16.31	1 095.1
Current query	11.80	10.44	9.87	9.03

The current-query policy has the lowest perplexity on all four models in the table. To isolate the key stage, we compare current-query and calibration-only key allocation at the same $(R_K, R_V) = (3, 1)$ split on Qwen3B. Replacing current-query allocation increases PPL from 11.789 to 12.011, with matched DENSE PPL 11.362. The paired NLL increase is 0.0187 nats per target, with a 95% document-bootstrap interval of $[0.0139, 0.0247]$. Thirty-one of 32 documents favor current-query keys. This establishes a key-stage benefit in this setting without attributing a fixed share of the total gain to either stage.

F.6 INTERMEDIATE PRECISION LEVELS

The progressive code offers several reconstruction depths. We test whether these intermediate levels add value beyond selecting a subset of items for high precision. Two-depth controls restrict keys, values, or both to $\{0, W\}$ or $\{1, W\}$ while holding the codec, scores, calibration, seed, and K/V split fixed. With N items and lower depth ℓ, a schedule that assigns depth W to k items spends $\ell N + k(W - \ell)$ depth increments. We therefore compare schedules at the same feasible budget. Value-only comparisons use identical reconstructed keys. When the minimum depth and average value depth are both one, the value schedule is uniform by construction.

Using the exact-budget schedules, the key-only $\{1, W\}$ restriction at the READKV $(8, 2)$ operating point increases NLL over the multilevel reader by 0.227 nats per token on Qwen3B, 0.159 on Qwen7B, and 0.275 on Yi6B for seed 20260801. Across three seeds, the Qwen3B and Qwen7B increases range from 0.227–0.243 and 0.157–0.167 nats. Restricting both stages produces substantially larger increases of 2.408–3.598 and 3.229–3.349 nats. These comparisons show that the benefit is not explained solely by choosing which items receive a single high-precision reconstruction.

F.7 ALLOCATION SCORES AND ATTENTION ERROR

The allocator optimizes separable scores rather than exact attention-output error. We test how closely these scores track the errors they are intended to reduce. Queries, keys, and values are held fixed across schedules using captures from three layers, four C4 documents, and 24 preselected query rows per document. The rows span three context-length ranges for Qwen3B and Qwen7B. Each observation pairs a query row with one physical KV head, aggregating the query heads that share it. Each schedule uses the attention probabilities produced by its own reconstructed keys. Full-precision probabilities serve only as diagnostic targets. The exact decomposition in Appendix E.3 separates the key, value, and interaction terms for each observation.

The diagonal objectives omit interactions between reconstruction errors. For pooled $R = 2$, the median signed contribution of these interactions is 2.1% of the exact squared logit error on Qwen3B and 1.2% on Qwen7B. For attention-weighted value error, the corresponding cross-token fractions are much larger: 64.6% and 72.6%. The value objective can therefore guide placement without accurately estimating the error magnitude. Relative to the multilevel reader, restricting both stages to $\{1, W\}$ worsens output error in 575/576 observations on Qwen3B and all 1152 on Qwen7B. The $\{0, W\}$ restriction worsens it in 573/576 and 1149/1152 observations, respectively.

Weighting value scores by $s_g = \sum_c (\sigma^V_{g,c})^2$ tests whether heads with larger value magnitudes should receive more of the pooled budget. It changes complete-model NLL by less than 0.001 nats per token in all four model-rate conditions. Three paired intervals include zero, and the fourth difference is also below 0.001 nats. With fixed per-head budgets, multiplying all value scores in a head by the same positive factor should leave its allocation unchanged. Nevertheless, repeated whole-model executions

of this control differed by about 0.0024 nats per token; the source of that harness variability remains unresolved.

F.8 GPU READER CONSTRAINTS

The GPU reader trades allocation flexibility for a simpler execution policy. To identify which restrictions drive its quality gap, we vary the K/V split, pooled versus per-head budgets, minimum depth, and whether adjacent pairs of value tokens must share a depth. A factorial study evaluates all $2^4 = 16$ combinations on Qwen3B and Qwen7B at $R = 2$ and $R = 4$. Table 9 follows one cumulative path through the design.

Table 9: Effect of GPU-reader restrictions on C4 negative log likelihood. Entries give changes in nats per predicted token. Positive values are worse. The columns successively impose equal K/V rates, per-head budgets, minimum depth one, and paired values. The total compares the final restricted reader with the flexible reader. Because the restrictions are cumulative, each intermediate increment depends on their order.

Model	R	Equal K/V	Per head	Min. 1	Paired	Total
Qwen3B	2	+0.0729	+0.0050	+0.0012	+0.0004	+0.0795
Qwen7B	2	+0.0575	+0.0043	+0.0061	+0.0014	+0.0694
Qwen3B	4	+0.0032	+0.0002	+0.0004	-0.0003	+0.0035
Qwen7B	4	+0.0034	-0.0005	+0.0002	-0.0003	+0.0028

Equal K/V rates account for most of the low-rate gap along the displayed path. A mandatory minimum depth of one also forces uniform value reading when the selected value budget is $R_V = 1$. At $R = 4$, all four restrictions have much smaller effects. The same distinction appears in Qwen passkey retrieval at 8K: the flexible READKV $(8, 2)$ reader recovers 200 of 200 keys, while the restricted READKV $(8, 2)$ policy recovers 100–101. The restricted READKV $(8, 4)$ reader recovers all 200. These results identify the K/V split as the main target for improving the low-rate GPU policy. Any revised policy should be evaluated jointly for quality, traffic, and latency.

G GPU LATENCY AND MEMORY TRAFFIC

Section 5 summarizes the principal 8K, batch-one measurement. Here we extend that comparison across context lengths and batch sizes, then examine the memory traffic behind the observed latency. All measurements use an NVIDIA A10G and start from an existing cache. They cover one attention layer during decoding, excluding cache construction and the rest of the model.

The timed implementation uses a more constrained reader than the principal quality experiments. It assigns budgets separately to each KV head, requires at least one level per item, divides the budget equally between keys and values, and groups values into two-token packets. The quality reader instead pools its budget across heads and permits depth zero. Appendix F.8 measures the quality effect of these choices. The GPU results below therefore report performance for the restricted reader, while the language-model quality results use the flexible reader.

G.1 LATENCY ACROSS WORKLOADS

Table 10 reports the latency of a complete attention-layer call at context lengths 2K, 8K, and 32K. We use batch sizes $B = 1$ and $B = 4$. Every method runs on the same CUDA stream and uses the same timing boundary. CUDA events measure repeated replay of a recorded GPU execution graph, reducing host launch overhead. The warm-cache measurements reuse the workload without deliberately flushing the GPU cache; they do not assume that the entire KV cache fits on chip. The READKV timing includes its final conversion to bf16.

DENSE uses an uncompressed bf16 KV cache with PyTorch scaled dot-product attention. The TurboQuant row runs its fused K4/V4 codec on the benchmarked layer, although the quality configuration in Section 5 leaves this layer uncompressed. The FULL READER $(4, 4)$ reads all four levels of the READKV code without query-dependent allocation. The remaining READKV rows use the restricted GPU reader described above.

Table 10: Median single-layer decode latency (ms) on an NVIDIA A10G under CUDA Graph replay with a warm cache. The 2K and 8K inputs use Qwen2.5-7B layer 0; the 32K input uses synthetic K/V with the same head geometry. All rows use the same timing boundary, including READKV's final bf16 conversion.

	2K tokens		8K tokens		32K tokens	
Method	$B = 1$	$B = 4$	$B = 1$	$B = 4$	$B = 1$	$B = 4$
DENSE	0.020	0.048	0.051	0.146	0.168	0.585
TurboQuant K4/V4 codec Zandieh et al., 2026	0.055	0.143	0.154	0.518	0.625	1.990
FULL READER $(4, 4)$	0.038	0.076	0.070	0.171	0.265	0.727
READKV $(4, 2)$	0.063	0.095	0.092	0.179	0.287	0.719
READKV $(8, 2)$	0.066	0.096	0.093	0.183	0.304	0.740
READKV $(8, 4)$	0.069	0.108	0.105	0.211	0.336	0.834

The READKV readers are faster than the tested TurboQuant codec at 8K and 32K for both batch sizes. TurboQuant is faster at 2K and batch one, while DENSE is fastest throughout. A cache-eviction check at 8K and batch one preserves this ordering: READKV $(8, 2)$ changes from 0.093 to 0.094 ms, TurboQuant from 0.154 to 0.177 ms, and DENSE from 0.051 to 0.056 ms.

These fixed-batch timings answer how quickly each implementation processes an existing cache. Compression also reduces the memory needed to retain that cache: it can make longer contexts or more concurrent requests fit on a GPU, even when decoding the compressed representation adds latency. Our measurements establish the cache-size reduction, but do not test throughput under a memory-limited serving workload. Thus the faster dense attention call does not eliminate the capacity benefit of TurboQuant or READKV.

G.2 MEMORY AND PHYSICAL TRAFFIC

We next examine whether the reduction in requested payload appears in physical memory traffic. At 8K and batch one, the four-bit and eight-bit READKV caches contain 4.00 and 8.00 MiB of payload, respectively. Each READKV call also uses 3.87 MiB of workspace for scores, probabilities, schedules, and partial outputs. TurboQuant retains 4.19 MiB and uses 0.45 MiB of workspace, while the uncompressed K/V cache occupies 16.00 MiB. These values describe one layer. Tables 2 and 3 report the corresponding analytical cache costs for the complete models.

Table 11 profiles the same 8K, batch-one workload. The READKV $(4, 4)$ scheduled control reads all four levels through the restricted execution path. It produces the same reconstruction as the FULL READER $(4, 4)$ while retaining scheduling and intermediate memory operations. Hardware counters cover the complete eager attention call, including intermediate and output accesses, and measure physical DRAM traffic. Latency is measured separately under graph replay. These physical measurements differ from the logical read counts used in the quality tables. The evicted condition deliberately displaces prior GPU-cache contents before the measured call, checking sensitivity to cache reuse.

Table 11: Physical DRAM traffic for the same NVIDIA A10G single-layer benchmark used in the latency study, with 8,192 cached tokens and batch one. Each entry reports warm / cache-evicted traffic in MiB. Counters cover the complete eager attention call, including intermediate and output accesses. Graph-replay latency is measured separately.

Method	DRAM reads	DRAM writes	Total
DENSE	17.73 / 17.86	3.12 / 3.19	20.85 / 21.06
READKV $(4, 2)$	2.72 / 2.71	4.63 / 4.71	7.35 / 7.42
FULL READER $(4, 4)$	4.70 / 4.66	4.19 / 4.25	8.90 / 8.91
READKV $(4, 4)$, scheduled	4.81 / 4.85	4.64 / 4.73	9.44 / 9.58
READKV $(8, 2)$	2.80 / 2.74	4.67 / 4.73	7.47 / 7.47
READKV $(8, 4)$	4.99 / 4.98	4.72 / 4.75	9.71 / 9.74
TurboQuant K4/V4 Zandieh et al., 2026	7.28 / 7.92	2.10 / 2.92	9.38 / 10.84

For READKV $(8, 2)$, the kernel reads 2.80/2.74 MiB under warm/evicted conditions, compared with 7.28/7.92 MiB for TurboQuant. This reduces DRAM reads by 62–65%. After including intermediate and output writes, total traffic falls from 9.38/10.84 MiB to 7.47/7.47 MiB, a reduction of 20–31%. At the larger read budget, READKV $(8, 4)$ transfers 9.71/9.74 MiB in total, slightly more than TurboQuant in the warm condition. DENSE is faster despite transferring 20.85/21.06 MiB, which shows that memory traffic is only one component of latency.

The READKV $(4, 2)$ and READKV $(8, 2)$ rows further separate stored from fetched precision. Both read two payload bits per scalar and take 0.092–0.093 ms, although the retained payload grows from 4 to 8 MiB. On this fixture, the additional stored precision has little effect on execution time, but it still consumes memory capacity.