RDKV: RATE-DISTORTION BIT ALLOCATION FOR JOINT EVICTION AND QUANTIZATION OF THE KV CACHE

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have shown strong performance across diverse tasks, but their inference with long input contexts is bottlenecked by memory size and bandwidth. The Key-Value (KV) cache size grows linearly with sequence length and needs to be re-read from off-chip high-bandwidth memory (HBM) to on-chip memory at every decoding step, resulting in memory-bound inference. Existing methods reduce the cache by either eviction or quantization, but typically treat the two in isolation. In this paper, we cast KV cache compression as a rate-distortion problem, under which eviction and quantization are two end-points of the same bit allocation scheme. This exposes the need to optimize them jointly, motivating our method, **RDKV** (**R**ate-**D**istortion **KV** cache compression). RDKV derives the weight of each token or channel from the distortion that compression induces on the attention computation. Based on these weights, it assigns each token or channel a bit-width ranging from full precision down to zero bits via Lagrangian relaxation. Experiments on LongBench, RULER, and InfiniteBench show that RDKV outperforms the best evaluated baseline by 9.1% on average. On LongBench it recovers 97.81% of full-cache accuracy with only 2.48% cache retention. Compared with full-cache FlashAttention-2 decoding, it achieves **4.5**$\times$ decode speedup and **1.9**$\times$ peak memory reduction with 128K context length, while maintaining comparable performance.

1 INTRODUCTION

Large language models (LLMs) [1, 7, 34] have demonstrated strong performance from open-ended generation to complex multi-step reasoning. As these capabilities move into production, the input contexts they require have grown from thousands to hundreds of thousands of tokens. Retrieval-augmented generation [30], multi-document reasoning, and agentic workflows all routinely operate at this scale. At these lengths, inference becomes memory-bound. Transformer decoding stores a Key-Value (KV) pair for every token in every layer. The cache therefore grows linearly with sequence length and reaches tens of gigabytes per request [27]. Because every generated token re-reads this entire cache through the attention computation [12], memory traffic rather than arithmetic sets the per-token decode latency [38]. Reducing the cache footprint without retraining, while preserving generation quality, has thus become a central problem for efficient long-context inference [31, 35].

A growing number of works address this bottleneck by compressing the KV cache. These methods fall along two axes: eviction and quantization. Eviction methods score each token or channel and keep the top-ranked units at full precision, dropping the rest [33, 46, 52]. Quantization methods retain every token and instead reduce the bit-width [22, 37]. Yet cache units differ widely in importance: a few are critical, most are not. A binary keep-or-evict action leaves no precision tier for the broad middle, while mixed-precision quantization retains every token, including the many that contribute negligibly to the output. Recent analysis shows that combining eviction with quantization outperforms either action alone [50]. Several hybrids route each token among full-precision, low-bit, and evicted states [4, 29]. But their routing scores remain heuristic and the budget ratios are preset or searched before the final assignment. A natural question is whether both the score and the allocation can be derived from a single objective function.

In this paper, we present RDKV, a rate–distortion framework for KV cache compression. RDKV poses a bit-allocation problem: given a fixed bit budget, assign each token in the V cache and each

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.

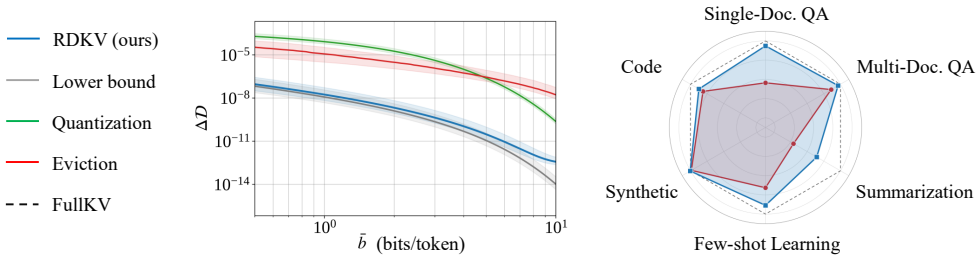


Figure 1: **Left:** Per-sequence weighted distortion $\Delta\mathcal{D} = \sum_u w_u \, \varepsilon_u(b_u)$ (Eq. (6)) versus average bit-width $\bar{b}$. Lines: median across sequences; shaded: IQR. *Lower bound*: continuous relaxation (Thm. 3.3). **Right:** LongBench score by task category at a per-layer cache budget of 128 FP16-equivalent tokens ($B_{\text{total}} = 128L$), normalized by FullKV; per-task scores in Tab. 1. *Eviction* refers to Ada-SnapKV [15] in the right panel. Model: LLaMA-3.1-8B-Instruct [19].

channel in the K cache a bit-width to minimize a weighted distortion. The weight of each token or channel is defined as the deviation in the attention distribution or the attention logit when it is evicted. Lagrangian relaxation maps these weights to bit-widths, from full precision for critical units down to zero bits for negligible ones. Eviction and quantization are thus at the two ends of one allocation curve and are explored jointly rather than in a staged fashion.

We make three **contributions**. (1) We formulate KV cache compression as a rate–distortion problem and show why eviction and quantization should be optimized jointly. The formulation places eviction and quantization under the same objective. Restricting the allocation to either action alone leaves a clear gap as Fig. 1 (left) shows on calibration data. (2) We derive per-unit distortion coefficients from the attention computation and use them to instantiate the rate–distortion objective. We formulate the resulting discrete rate–distortion allocation as a multiple-choice knapsack and solve it with Lagrangian relaxation. (3) We introduce TriZone to enable efficient packed decoding over the resulting mixed-bit cache. With TriZone, RDKV achieves decode efficiency comparable to eviction-only methods. Overall, Fig. 1 (right) shows that RDKV approaches FullKV performance across all six categories while consistently outperforming the strongest evaluated baseline.

2 Related Work

Existing KV cache compression methods differ in when compression is applied, with some compressing the prefix cache at prefill time and others continuing compression during decoding [44].

Prefill-and-Decode Compression. Unified compression is typically realized through quantization, which stores both prefill and decode KV states at reduced precision. This setting is particularly suited to long-output tasks, where the KV cache continues to grow throughout decoding. Methods such as KIVI [37], KVQuant [23], ZipCache [22], and KVTuner [32] exploit asymmetric K/V quantization, outlier handling, salient-token preservation, or searched precision configurations to reduce the bit-width of stored keys and values. QAQ [14] and MiKV [49] adapt precision according to token or KV-pair importance, while AQUA-KV [41] exploits K/V dependencies by quantizing prediction residuals. More recent methods further combine quantization with eviction. HqeKV [4] searches the fraction of tokens assigned to different precision levels, whereas ARKV [29] estimates quantization ratios from prefill-time attention statistics, with scoring thresholds searched offline.

Prefill-Time Compression. Prefill-time compression is typically realized through eviction, which removes low-importance KV states from the input prefix. This setting generally faces long-input tasks, where the prompt dominates the KV-cache footprint. StreamingLLM [45] keeps sink tokens and a recent window. H2O [52], SnapKV [33], Scissorhands [36], and NaCl [9] use attention-derived token scores. AdaKV [15], PyramidKV [8], and CAKE [39] further distribute the budget across heads or layers. Expected Attention [13] estimates future-query scores rather than relying only on the observed window. Recent work sharpens the scoring signal beyond attention weights.

CriticalKV [16] optimizes worst-case attention-output perturbation incorporating value states. OB-Cache [20] applies the Optimal Brain framework [21, 28] to estimate each pair's impact on the attention output. CAOTE [18] scores via the output error induced by removal. AnDPro [17] projects value states onto anchor directions. ReST-KV [3] stabilizes scores through layer-wise reconstruction and spatial-temporal smoothing. ThinK [46] extends eviction from tokens to key channels.

However, existing budget-driven methods restrict the action to a binary keep-or-drop decision. This restriction leaves no intermediate precision tier for tokens or channels between the two extremes. The framework in Sec. 3 extends this paradigm by allowing each cache unit to be either evicted or retained at different bit-widths according to its importance. Unlike the hybrid methods in prefill-and-decode compression, it determines eviction and quantization within a single optimization problem, without setting an offline ratio or threshold.

3 METHODOLOGY

Consider a decoder-only transformer [38, 43] with L layers, H_q query heads, H_{kv} KV heads (group size $g = H_q/H_{\text{kv}}$) [2], and per-head dimension d. For a prefilled context of T tokens the KV cache per layer comprises $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{H_{kv} \times T \times d}$, with attention weights $a_{\tau,t} = \mathrm{softmax}_t(q_\tau^\top k_t/\sqrt{d})$ and output $o_\tau = \sum_t a_{\tau,t}\, v_t$. We assign each cache unit u (a token or a channel) a bit-width $b_u \in \mathcal{B} = \{0, 2, 4, 8, 16\}$, from FP16 retention to outright removal, under a total bit budget B. Our goal is to find the allocation $\{b_u^\star\}$ that minimizes the distortion introduced to the attention computation under this budget constraint.

In the following, we first quantify the compression cost of each token or channel, and formulate the bit allocation as a rate-distortion problem (Sec. 3.1). We then describe the bit allocation pipeline (Sec. 3.2) and the packed-decode layout that realizes the mixed-bit cache in hardware (Sec. 3.3). Due to space constraints, formal proofs in this section are deferred to Sec. A.

3.1 RATE-DISTORTION FORMULATION

To quantify the compression cost of each unit, we analyze the effect of removing a single token or channel from the uncompressed cache. In the V cache, each token enters the output as a single weighted term $a_{\tau,t}\, v_t$. Removing it perturbs the attention distribution a_τ. In the K cache, persistent channel-wise outliers [37] make the channel the natural compression unit. Zeroing a channel perturbs the logit matrix Z. Because both perturbations are linear, they admit closed-form weights.

Distortion weight in V cache. Consider evicting token t from the V cache of a single head. Because a_τ is produced by softmax, the remaining tokens automatically absorb the freed probability mass. The post-eviction distribution is

$$\hat{a}_{\tau,t} = 0, \qquad \hat{a}_{\tau,t'} = \frac{a_{\tau,t'}}{1 - a_{\tau,t}} \quad \text{for } t' \neq t. \tag{1}$$

To measure how much this changes the attention pattern, we use the total variation (TV) distance between a_τ and $\hat{a}_\tau$. It yields the per-token distortion weight:

Proposition 3.1 (Token weight in V cache)**.** *Evicting token t yields $\|a_\tau - \hat{a}_\tau\|_{\text{TV}} = a_{\tau,t}$ for each query τ. The per-token distortion weight, obtained by summing over all queries, is*

$$w_t \;:=\; \sum_\tau a_{\tau,t}. \tag{2}$$

The weight w_t is the total attention that token t receives across all query positions, recovering the cumulative attention widely used as a token-level eviction criterion [15, 33, 52]. In prior work this quantity serves as a ranking score for binary keep-or-evict decisions. Here it enters the objective as a multiplicative coefficient. Its magnitude therefore affects the allocated bit-width, not merely its rank.

Distortion weight in K cache. An analogous construction applies to the K cache, but with a different partition. Channel c contains the T scalars $\{k_{t,c}\}_{t=1}^T$ across all token positions. Consider removing channel c, i.e., setting $K[:, c]$ to zero. Since each logit entry is the inner product $q_\tau^\top k_t/\sqrt{d}$, zeroing one channel removes the contribution of that coordinate from every entry simultaneously. The

resulting deviation in the logit matrix is

$$\delta Z_c \;=\; -\frac{1}{\sqrt{d}}\, Q[:,c]\, K[:,c]^\top, \tag{3}$$

which yields the per-channel distortion weight:

Proposition 3.2 (Channel weight in K cache)**.** *The average squared logit deviation induced by removing channel c is proportional to*

$$w_c \;:=\; \|Q[:,c]\|_2^2\, \|K[:,c]\|_2^2. \tag{4}$$

The weight w_c measures the average squared logit deviation induced by removing channel c: channels whose Q and K columns have larger second moments are more expensive to remove.

Notably, w_t (Eq. (2)) and w_c (Eq. (4)) are both derived from eviction, i.e., the zero-rate removal of a token or channel. To extend these weights to finite-rate quantization, we separate task sensitivity from quantization distortion. The weights w_t and w_c quantify the sensitivity of the attention computation to each cache unit, while $\varepsilon_V(b)$ and $\varepsilon_K(b)$ quantify the reconstruction distortion incurred at bit-width b. Their product therefore defines the compression cost used in the bit-allocation objective.

Rate-Distortion Aware Bit Allocation. Given these coefficients, the remaining question is how to choose bit-widths b_u under a total budget, where u indexes tokens ($n_u = d$ scalars) or channels ($n_u = T$ scalars). Under Bennett's high-rate approximation [6], a uniform scalar quantizer with dynamic range R_u achieves per-coordinate mean-squared error $\sigma_u^2 2^{-2b}$, where $\sigma_u := R_u/(2\sqrt{3})$. Combining the per-unit weight with this reconstruction distortion gives

$$\Delta D(\{b_u\}) = \sum_u w_u \sigma_u^2 2^{-2b_u}, \qquad \text{s.t.} \quad \sum_u b_u \le B. \tag{5}$$

The coefficient $w_u\,\sigma_u^2$ combines the distortion weight (w_u) with quantization hardness (σ_u^2). The continuous relaxation admits a reverse water-filling form [10, 40]:

Theorem 3.3 (Optimal continuous bit allocation)**.** *The minimizer of Eq.* (5) *subject to $b_u \ge 0$ is*

$$b_u^\star \;=\; \left[\frac{1}{2}\log_2 \frac{2\ln 2 \cdot w_u\,\sigma_u^2}{\lambda}\right]_+,$$

with $\lambda > 0$ chosen so that the budget binds. The water level $\lambda/(2\ln 2)$ induces a phase transition: units with $w_u\,\sigma_u^2 < \lambda/(2\ln 2)$ receive $b_u^\star = 0$, while units above receive positive bit-widths.

Theorem 3.3 reveals that eviction and quantization are not separate actions but two regimes of one allocation curve, selected jointly by the budget B rather than composed under pre-allocated ratios [29]. Tightening the budget B raises λ and pushes more units past the zero-rate boundary into eviction. Loosening it promotes previously evicted units to low-bit retention. This continuous solution provides the rate-distortion interpretation underlying the joint allocation of eviction and quantization.

Hardware restricts bit-widths to the discrete set $\mathcal{B} = \{0, 2, 4, 8, 16\}$. We replace Bennett's curve with an empirical per-coordinate distortion $\varepsilon_u(b)$ calibrated at each $b \in \mathcal{B}$ (see Sec. M), including $b = 0$ for removal, and solve the resulting multiple-choice knapsack problem (MCKP):

$$\{b_u^\star\} \;=\; \arg\min_{b_u \in \mathcal{B}} \sum_u w_u\, \varepsilon_u(b_u) \qquad \text{s.t.} \quad \sum_u b_u \le B. \tag{6}$$

Lagrangian relaxation decouples this into independent per-unit table lookups. A one-dimensional bisection over λ recovers a feasible primal allocation in $O(U|\mathcal{B}|)$ time, adding negligible latency compared to prefill (Sec. H.1). Weak-duality bounds certify near-optimality (Sec. A). The additive surrogate is validated end-to-end in Sec. I.

3.2 Per-Head Allocation Pipeline

We instantiate the weights and discrete allocation of Sec. 3.1 as a three-stage pipeline (Fig. 2) that runs once after prefill, separately for each layer-head pair (ℓ, h). Under FP16 reference, a per-head budget of B_{tok} tokens corresponds to $B_{\text{head}} = 2\,B_{\text{tok}}\,d \cdot 16$ total bits, split equally between V and K: $B_V = B_K = \frac{1}{2}B_{\text{head}}$.

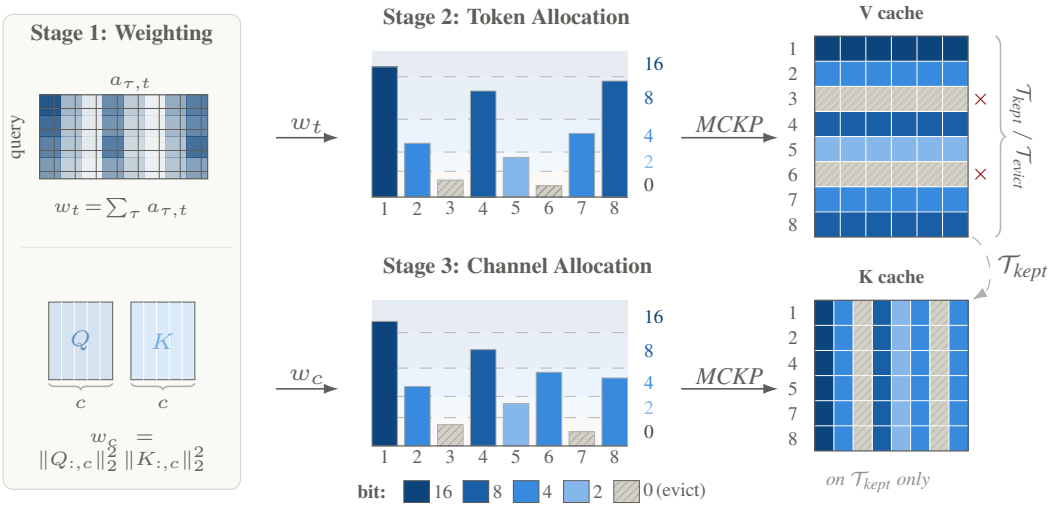


Figure 2: RDKV per-head bit-allocation pipeline (illustrated for 8 tokens and 8 channels). **Stage 1.** Token weights $w_t = \sum_\tau a_{\tau,t}$ are derived from the attention matrix, and channel weights w_c from Q/K column second moments. **Stage 2.** Discrete bit allocation on w_t assigns each token one of five bit-widths $\{16, 8, 4, 2, 0\}$. In this example tokens 3 and 6 fall below the lowest threshold and are evicted (hatched rows in V cache). **Stage 3.** The same allocation on w_c assigns per-channel bit-widths to the K cache, which retains only the six kept tokens ($\mathcal{T}_{\text{kept}} = \{1, 2, 4, 5, 7, 8\}$).

Stage 1: Distortion Weight Computation. From the prefill forward pass we compute the token weights w_t (Prop. 3.1) and channel weights w_c (Prop. 3.2), as illustrated in Fig. 2 (a). Both are computed once from the full uncompressed cache. w_t requires only the attention matrix and w_c requires only per-channel second moments of the probe queries and stored keys.

Stage 2: Token Allocation in V Cache. Since each token contains d scalars, the total V budget B_V translates to $\bar{B}_V := B_V/d$ in units of summed bit-widths. Given the token weights w_t and a calibrated distortion table $\varepsilon_V(b)$ (Sec. B), Stage 2 solves the MCKP of Eq. (6), as shown in Fig. 2 (b):

$$\{b_t^V\} = \arg\min_{b_t^V \in \mathcal{B}} \sum_t w_t \, \varepsilon_V(b_t^V) \qquad \text{s.t.} \quad \sum_t b_t^V \le \bar{B}_V.$$

The output determines $\mathcal{T}_{\text{kept}} := \{t : b_t^V > 0\}$ and its complement $\mathcal{T}_{\text{evict}}$. Retained tokens receive bit-widths in $\{2, 4, 8, 16\}$. Evicted tokens are removed entirely.

Stage 3: Channel Allocation in K Cache. As shown in Fig. 2 (c), Stage 3 distributes the K budget B_K across only the retained tokens $\mathcal{T}_{\text{kept}}$. Each channel contains $|\mathcal{T}_{\text{kept}}|$ scalars, so the budget translates to $\bar{B}_K := B_K/|\mathcal{T}_{\text{kept}}|$:

$$\{b_c^K\} = \arg\min_{b_c^K \in \mathcal{B}} \sum_c w_c \, \varepsilon_K(b_c^K) \qquad \text{s.t.} \quad \sum_c b_c^K \le \bar{B}_K.$$

A channel assigned $b_c^K = 0$ is removed under the same zero-rate interpretation as token eviction. Across retained tokens, K bit-widths are assigned per-channel independently of the per-token b_t^V. A single token's K and V can therefore carry different bit-widths, the prerequisite for the TriZone layout of Sec. 3.3. Visualization of the resulting bit-width allocations of tokens and channels is in Sec. L.

The V cache is allocated before K because the eviction decision determines which tokens remain in the K cache. Reversing the order would waste K budget on soon-to-be-evicted tokens. The channel weights w_c are reused from Stage 1 without recomputation after V-side eviction, avoiding a second forward pass. The complete pseudocode is given in Sec. C.

3.3 Efficient Packed Decode

The pipeline of Sec. 3.2 produces a mixed-bit cache per (ℓ, h) pair. A mixed-bit allocation alone does not reduce decode cost: if the quantized entries are unpacked to FP16 before the attention kernel reads them, peak HBM usage stays unchanged and the extra dequantization pass adds latency. To translate bit-width savings into actual speedup and memory reduction, the cache must stay packed in HBM while dequantization is fused into the attention computation. The challenge is that tokens within one head now carry different bit-widths, yet GPU kernels require uniform-precision input segments to avoid per-element branching.

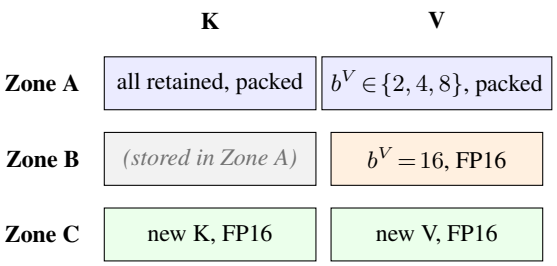


Figure 3: TriZone cache layout for one (ℓ, h) pair. Each zone admits a uniform dequantization path; cell labels show the stored format.

We address this by organizing each (ℓ, h) pair into three storage zones that separate packed quantized entries, full-precision retained entries, and newly generated decode tokens. Each zone admits a uniform dequantization path during the attention kernel (Fig. 3; we suppress (ℓ, h) in what follows).

Zone A is the packed old-cache stream: all retained K rows from $\mathcal{T}_{\text{kept}}$, together with the V rows whose $b_t^V \in \{2, 4, 8\}$. Within Zone A, V rows are grouped into uniform-bit segments by b_t^V and K channels by b_c^K (Sec. B). Each segment uses a single dequantization path, avoiding per-element branching. Zone B holds the FP16 V rows for $\mathcal{T}_{V16} := \{t \in \mathcal{T}_{\text{kept}} : b_t^V = 16\}$; their K rows remain in Zone A, since K bit-widths follow the per-channel allocation of Sec. 3.2 rather than b^V. A single token's K and V can therefore reside in different zones. Zone C appends FP16 K and V for newly generated tokens $\mathcal{T}_{\text{new}}$, growing by one entry per decode step.

At each decode step, the query q_τ attends over retained prefill tokens and newly appended decode tokens. Logits combine the quantized old K with the FP16 new K,

$$z_\tau \;=\; q_\tau \big[\, \hat{K}^\top_{\mathcal{T}_{\text{kept}}},\; K^\top_{\mathcal{T}_{\text{new}}} \big] / \sqrt{d}, \qquad a_\tau = \mathrm{softmax}(z_\tau),$$

and the output decomposes into three value sources:

$$o_\tau \;=\; \underbrace{\sum_{t \in \mathcal{T}_{\text{kept}} \setminus \mathcal{T}_{V16}} a_{\tau,t}\, \hat{v}_t}_{\text{quantized V}} \;+\; \underbrace{\sum_{t \in \mathcal{T}_{V16}} a_{\tau,t}\, v_t}_{\text{FP16 V } (b^V=16)} \;+\; \underbrace{\sum_{t \in \mathcal{T}_{\text{new}}} a_{\tau,t}\, v_t}_{\text{FP16 V (new)}}. \tag{7}$$

The three terms correspond to the three zones: the first reads packed quantized V from Zone A, the second reads FP16 V from Zone B, and the third reads FP16 V from Zone C. Equation (7) is the numerical reference implemented by the packed decode path: dense attention on the compressed cache $(\hat{K}, \hat{V})$, with dequantization fused into the attention computation. The latency measurements in Sec. 4 use this implementation. Section B gives layout details including bit-width grouping, padding, and masking.

4 Experiments

Models. We primarily evaluate on LLaMA-3.1-8B-Instruct [19]. Cross-architecture results (Mistral-7B-Instruct-v0.3 [25], Qwen3-4B [47]) and cross-scale results (LLaMA2-13B-Chat [42], Qwen2.5-72B-Instruct [48]) are listed in Sec. D.

Baselines. We compare against four budget-driven methods. SnapKV [33] scores tokens by attention within a recent observation window and retains the top-k per head. Ada-SnapKV [15] (hereafter AdaKV) extends this with adaptive per-head budget allocation. ThinK [46] applies pruning at the channel level of the K cache rather than the token level (paired with SnapKV). SnapKV+ZipCache [22, 33] combines token eviction with post-hoc quantization of the retained cache.

Benchmarks. We evaluate on LongBench [5] (16 tasks covering 6 different task categories), Needle-in-a-Haystack [26] (single-fact retrieval at varying depths), RULER [24] (11 retrieval and reasoning tasks from 4K to 128K), and InfiniteBench [51] (10 tasks with context lengths up to 2M tokens).

Table 1: Performance on 16 LongBench [5] datasets for LLaMA-3.1-8B-Instruct across cache budgets $B_{\text{total}} \in \{64L, 128L, 256L, 512L, 1024L\}$. Snap+Zip denotes SnapKV [33]+ZipCache [22]. The best result in each row is in **bold**; the second-best is underlined.

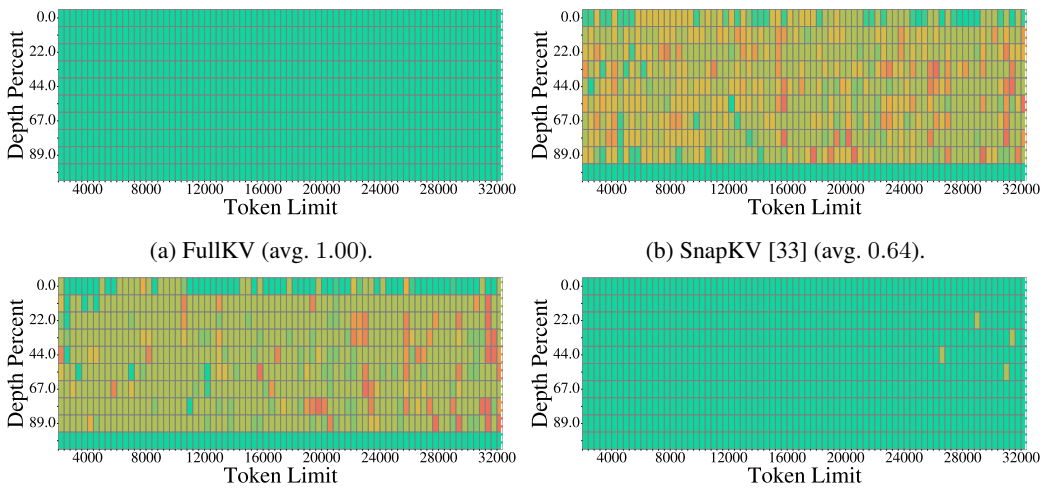
Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
LLaMA-3.1-8B-Instruct, B_{total} = Full																	
FullKV	29.38	44.82	55.46	57.62	49.22	32.80	34.58	25.34	26.90	73.00	91.51	43.64	9.04	99.50	65.24	59.08	49.82
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 64L$																	
SnapKV[33]	25.20	21.05	39.35	51.69	45.25	26.57	15.06	21.24	14.60	39.00	81.43	36.62	9.00	95.00	54.74	47.41	38.95
AdaKV[15]	24.62	19.62	39.41	50.93	42.13	26.10	16.08	22.17	15.71	39.50	85.62	37.83	9.00	97.50	56.57	50.70	39.59
ThinK[46]	24.93	23.54	45.55	53.54	44.17	27.79	18.38	22.77	18.10	42.00	84.35	38.50	9.00	98.50	55.81	49.22	41.01
Snap+Zip	**29.85**	32.53	52.16	55.24	46.72	**30.60**	19.96	22.88	21.26	52.50	90.50	40.03	**9.01**	75.50	61.01	51.97	43.23
RDKV	28.00	**34.47**	**54.16**	**55.90**	**47.45**	29.65	**23.10**	**23.78**	**22.54**	**59.50**	**91.76**	**41.52**	8.70	**99.17**	**61.70**	**54.10**	**45.97**
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 128L$																	
SnapKV[33]	24.41	24.19	48.91	55.98	45.51	28.41	18.64	22.76	18.72	47.00	88.91	39.41	**9.00**	98.00	59.14	52.18	42.57
AdaKV[15]	26.35	23.50	51.59	57.11	46.38	28.36	19.68	23.09	19.58	49.50	89.92	40.25	**9.00**	99.00	60.42	54.59	43.64
ThinK[46]	25.40	30.67	49.96	55.71	46.76	28.47	21.44	23.35	20.83	50.00	89.87	40.97	**9.00**	99.00	60.18	53.22	44.05
Snap+Zip	29.40	36.64	53.26	57.27	47.23	**31.09**	22.59	23.66	23.97	61.00	**92.00**	**42.17**	8.22	95.50	62.24	53.57	46.24
RDKV	**29.45**	**41.34**	**55.55**	**57.39**	**49.38**	30.89	**25.55**	**24.59**	**24.29**	**65.00**	91.92	41.80	8.75	**100.00**	**62.68**	**55.41**	**47.75**
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 256L$																	
SnapKV[33]	26.90	31.27	53.45	56.80	47.97	29.52	21.99	24.34	21.31	55.00	91.41	40.75	8.61	99.00	62.20	55.15	45.35
AdaKV[15]	25.22	32.02	52.51	56.82	47.65	30.71	22.49	23.86	21.78	61.00	91.27	40.92	8.63	**99.75**	**63.42**	**56.58**	45.91
ThinK[46]	26.83	34.14	52.82	56.77	49.14	29.08	23.63	23.47	22.87	61.50	91.51	41.35	8.60	99.50	62.82	55.44	46.22
Snap+Zip	28.02	41.07	54.05	57.74	48.02	**31.98**	24.27	23.80	24.82	66.00	91.67	**42.07**	7.90	99.50	61.72	54.18	47.30
RDKV	**29.63**	**44.85**	**56.33**	**57.85**	**49.64**	31.73	**28.01**	**24.66**	**25.66**	**70.00**	**91.87**	41.79	**8.83**	99.50	63.27	56.01	**48.73**
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 512L$																	
SnapKV[33]	29.68	39.51	**55.94**	56.94	48.87	30.56	24.31	24.07	23.60	65.50	91.46	41.44	8.57	**100.00**	64.50	56.58	47.60
AdaKV[15]	27.93	40.26	54.76	56.96	48.68	31.45	25.27	24.09	23.71	69.50	91.96	41.77	8.65	99.50	**64.89**	**57.66**	47.94
ThinK[46]	27.96	40.16	55.25	56.85	**49.60**	31.16	26.00	24.36	24.28	68.00	**92.00**	41.79	8.63	**100.00**	64.08	57.30	47.96
Snap+Zip	29.32	41.46	53.15	55.47	46.77	**33.26**	23.64	24.88	24.89	68.00	91.75	42.32	8.22	99.50	61.75	55.03	47.46
RDKV	**30.24**	**45.67**	55.58	**57.06**	48.95	31.67	**30.68**	**25.06**	**26.46**	**72.00**	91.83	**43.15**	**8.83**	**100.00**	63.36	57.02	**49.22**
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 1024L$																	
SnapKV[33]	29.51	43.17	**56.26**	57.43	49.18	32.07	27.21	24.63	25.26	69.50	91.70	42.28	8.15	**100.00**	**64.71**	**58.52**	48.72
AdaKV[15]	29.76	43.39	55.82	57.62	48.31	32.35	27.22	24.75	25.18	**72.00**	91.78	42.12	8.15	**100.00**	64.56	58.37	48.84
ThinK[46]	27.34	42.48	55.59	**57.78**	49.17	**32.36**	28.23	24.82	25.85	71.50	**92.50**	42.25	8.27	**100.00**	64.44	**58.52**	48.82
Snap+Zip	30.23	40.65	54.13	56.06	46.23	31.45	22.84	24.50	24.28	69.00	91.76	**44.15**	8.61	99.00	60.90	56.38	47.51
RDKV	**30.41**	**44.54**	55.82	57.14	**49.25**	31.77	**33.14**	**25.19**	**26.92**	**72.00**	91.84	43.90	**9.08**	**100.00**	63.56	57.01	**49.47**

Evaluation Protocol. All methods are evaluated under matched per-layer cache budgets $B_{\text{total}} = nL$ with $n \in \{64, 128, 256, 512, 1024\}$. For RDKV, n is the FP16-equivalent token count reallocated across tokens (V) and channels (K) at bit-widths $\mathcal{B} = \{0, 2, 4, 8, 16\}$. All score-based methods share the same size of observation window and pooling kernel. Further details in Sec. B.

4.1 Main Results

LongBench. LongBench [5] evaluates long-context understanding across 16 tasks spanning single/multi-document QA, summarization, few-shot learning, synthetic, and code. Table 1 reports per-task scores for LLaMA-3.1-8B-Instruct [19] across five cache budgets. RDKV achieves the highest average at every budget: at $B_{\text{total}} = 1024L$ it reaches 49.47, within 0.35 points of FullKV; at $64L$ the lead over the strongest baseline widens to 2.7 points. This reflects a structural advantage of joint allocation. Binary baselines discard every token below the top-k threshold. RDKV instead chooses from a larger action space via discrete bit allocation, assigning borderline tokens a low bit-width rather than evicting them. SnapKV+ZipCache [22, 33] also quantizes, but treats eviction and quantization as separate stages rather than two ends of one allocation curve. The advantage spans all six categories; the largest per-task gain appears on GovRep (3–5 points), where attention dispersed across lengthy documents favors mixed-precision over binary eviction. Cross-model results (Mistral-7B [25], Qwen3-4B [47], LLaMA-2-13B [42], Qwen2.5-72B [48]) are in Sec. D.

Needle-In-A-Haystack. Needle-in-a-Haystack [26] tests single-fact retrieval by inserting a target fact at varying depths. Figure 4 compares methods under $B_{\text{total}} = 64L$ at 32k context length. SnapKV [33] and AdaKV [15] develop failure bands at intermediate depths where the needle falls below the top-k



(a) FullKV (avg. 1.00).

(b) SnapKV [33] (avg. 0.64).

(c) AdaKV [15] (avg. 0.68).

(d) RDKV (avg. 0.99).

Figure 4: Needle-in-a-Haystack [26] on LLaMA-3.1-8B-Instruct at $B_{\text{total}} = 64L$. RDKV preserves a near-uniform retrieval pattern; SnapKV and AdaKV drop accuracy in mid-depth bands.

Table 2: RULER [24] 11-task average across sequence lengths (LLaMA-3.1-8B-Instruct).

Method	4K	8K	16K	32K	64K	128K
FullKV	98.58	98.88	96.98	90.33	88.49	79.91
SnapKV[33]	89.48	84.75	80.43	75.82	72.77	62.85
AdaKV[15]	92.03	85.49	80.70	75.67	72.47	58.43
ThinK[46]	91.76	86.44	81.45	75.94	72.44	62.85
Snap+Zip	76.49	77.84	76.26	75.05	74.59	62.37
RDKV	**98.62**	**98.61**	**95.65**	**88.28**	**80.07**	**66.95**

Table 3: InfiniteBench [51] 10-task average (LLaMA-3.1-8B-Instruct).

Method	Avg.
FullKV	45.38
SnapKV[33]	37.91
AdaKV[15]	38.06
ThinK[46]	36.73
Snap+Zip	37.76
RDKV	**39.46**

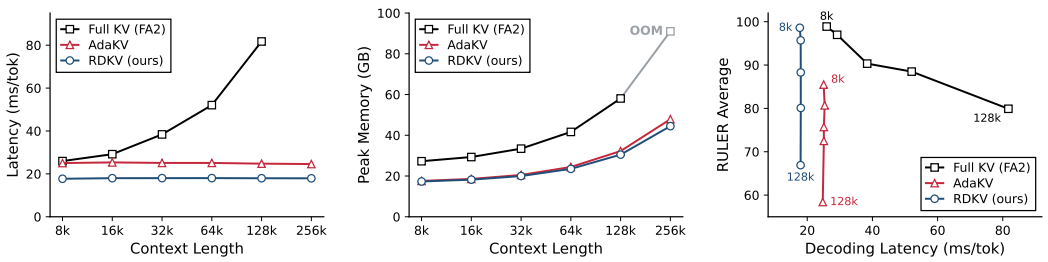
threshold and is evicted entirely. RDKV assigns the same token a low bit-width instead of discarding it. A 2- or 4-bit copy suffices to recover the answer. RDKV (avg. 0.99) thus maintains near-uniform retrieval close to FullKV (1.00). Further results are in Sec. F.

RULER. RULER [24] evaluates retrieval and reasoning at extreme context lengths, averaging 11 subtasks at sequence lengths from 4k to 128k. Table 2 reports results on LLaMA-3.1-8B-Instruct under $B_{\text{total}} = 1024L$. Eviction-only baselines degrade substantially as context grows, falling over 6 points below FullKV even at 4k and 17 points at 128k. RDKV stays within 1 point of FullKV up to 8k and leads the strongest baseline by 4.1–14.2 points across all lengths, because tokens that a top-k policy would discard are instead retained at low bit-width. Per-task breakdowns are in Sec. E.

InfiniteBench. InfiniteBench [51] spans 10 tasks with sequences exceeding 100k tokens (average ∼200k; the longest task, Zh.QA, reaches ∼2M tokens). Table 3 reports results on LLaMA-3.1-8B-Instruct under $B_{\text{total}} = 1024L$. RDKV ranks first among compression methods, ahead of AdaKV by 1.40 and SnapKV by 1.55 points, confirming that the allocation advantage persists at ultra-long contexts well beyond the lengths covered by LongBench and RULER. Per-task results are in Sec. G.

4.2 Memory and Latency

Latency. Fig. 5 reports decode latency, peak memory, and the latency-accuracy trade-off on LLaMA-3.1-8B-Instruct from 8K to 256K on a single A100 64 GB. Both FullKV and AdaKV [15] use FA2 [11]; reporting FullKV under SDPA would inflate RDKV's relative speedup to roughly 13×. FullKV per-token decode latency grows from 26 ms at 8K to 82 ms at 128K, while RDKV stays flat at ∼18 ms, a 4.5× speedup at 128K (Fig. 5a), since TriZone's packed cache size is fixed by B_{total} rather than by T. AdaKV is also at ∼25 ms under the same token budget, but RDKV is 1.4× faster.



(a) Decode latency. (b) Peak memory. (c) Latency vs. Accuracy.

Figure 5: Decode latency, peak memory, and latency-accuracy trade-off for LLaMA-3.1-8B-Instruct from 8K to 256K context length on a single A100 64 GB. Both FullKV and AdaKV use FA2.

Memory. FullKV peak memory grows linearly to 58.1 GB at 128K (Fig. 5b) and OOMs at 256K. RDKV uses 30.5 GB at 128K (1.9× reduction vs. FullKV) and 44.5 GB at 256K, running on the same device where FullKV cannot.

Latency-Accuracy Trade-off. Fig. 5c plots latency against RULER accuracy at each context length. RDKV and AdaKV operate at comparable latency, but RDKV is more accurate at every context length: at 128K the gap is 8.5 points (67.0 vs. 58.4). RDKV thus sits strictly above AdaKV on the latency–accuracy Pareto front.

4.3 Ablation Studies

Action Space. We compare four bit-width sets under the same distortion weights and discrete allocator. (Tab. 4, LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 128L$ and $512L$). Removing eviction (Quant-only) costs 8.7 / 8.6 points: the allocator must spend bits on every token, diluting precision for the critical few. Removing quantization (Evict-only) costs 5.2 / 2.0 points: moderate-importance tokens are discarded entirely when a 2- or 4-bit copy would suffice. Tri-state ($\{0, 4, 16\}$) closes most of the gap (0.8 / 0.2), confirming that *mixing* eviction with quantization matters more than bit-width granularity.

Table 4: Action-space ablation on LongBench average.

Config.	$128L$	$512L$
Quant-only	39.06	40.59
Evict-only	42.53	47.18
Tri-state	<u>46.91</u>	<u>49.05</u>
Joint (RDKV)	**47.75**	**49.22**

TriZone Packed Decode. Without TriZone (Sec. 3.3), the mixed-bit cache must be unpacked to FP16 before FA2, reading the same HBM as full-precision. Table 5 measured on LLaMA-3.1-8B-Instruct shows the effect: without TriZone, per-token latency is ∼36 ms at both 16K and 64K; with TriZone, it drops to ∼18 ms, a 2× reduction with bit-identical output. Combined with the smaller cache footprint from mixed-bit allocation, this gives the 4.5× decode speedup over FullKV reported in Fig. 5a. Additional ablation experiments are in Sec. H.

Table 5: TriZone ablation: decode latency (ms/tok).

Config.	16K	64K
RDKV w/ TriZone	**17.99**	**18.02**
RDKV w/o TriZone	36.05	36.47

5 Conclusion

We formulate KV cache compression as a rate-distortion problem. The solution obtained via Lagrangian relaxation allocates each cache unit a bit-width from $\mathcal{B} = \{0, 2, 4, 8, 16\}$. Eviction emerges as the zero-rate boundary of the same curve as quantization, so eviction and quantization are selected from a single optimization rather than composed under pre-allocated ratios. A packed-decode layout fuses dequantization into attention, turning the mixed-bit allocation into actual HBM savings. Across LongBench, RULER, and InfiniteBench on five open-source LLMs spanning different architectures and scales, RDKV consistently outperforms the evaluated baselines in accuracy, decoding speed, and peak memory at long context, and reaches 256K on a single A100 64 GB where Full KV does not. Our results show that treating eviction and quantization jointly can improve accuracy at no cost to efficiency. Extending the allocation to long-output decoding is left to future work (Sec. N).

AI USE STATEMENT

In this work, we used generative AI tools to assist in implementing methods and in writing proofs. Additionally, we used generative AI tools to draft and edit parts of the paper. All AI-assisted code, proofs, and text were reviewed, verified, and revised by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or newly collected datasets. All experiments use publicly available models and benchmarks under their respective licenses. RDKV is an inference-time compression method that does not modify model weights, and it introduces no capabilities beyond those of the underlying models. We do not foresee ethical concerns specific to this work beyond those associated with the deployment of LLMs in general.

REPRODUCIBILITY STATEMENT

The rate-distortion formulation and allocation pipeline are described in Sec. 3, with complete proofs in Sec. A and pseudocode in Sec. C. Implementation details, including probe hyperparameters, calibration protocol, bisection tolerance, baseline configurations, and the TriZone packed layout, are given in Sec. B. The calibrated distortion tables used by the allocator are reported in Sec. M. All models and benchmarks are publicly available. Anonymized source code is provided in the supplementary material.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901, 2023.

[3] Yongqi An, Chang Lu, Kuan Zhu, Tao Yu, Chaoyang Zhao, Hong Wu, Ming Tang, and Jinqiao Wang. ReST-KV: Robust KV cache eviction with layer-wise output reconstruction and spatial-temporal smoothing. In *The Fourteenth International Conference on Learning Representations*, 2026.

[4] Anonymous. HqeKV: Towards hybrid quantization and eviction for KV cache in long-context LLM inference. In *Submitted to ACL Rolling Review - January 2026*, 2026. under review.

[5] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137, 2024.

[6] William Ralph Bennett. Spectra of quantized signals. *The Bell System Technical Journal*, 27(3): 446–472, 1948.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[8] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

[9] Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. NACL: A general and effective kv cache eviction framework for llm at inference time. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7913–7926, 2024.

[10] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory 2nd Edition (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, July 2006. ISBN 0471241954.

[11] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[12] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[13] Alessio Devoto, Maximilian Jeblick, and Simon Jégou. Expected attention: KV cache compression by estimating attention from future queries distribution. *arXiv preprint arXiv:2510.00636*, 2025.

[14] Shichen Dong, Wen Cheng, Jiayu Qin, and Wei Wang. QAQ: Quality adaptive quantization for LLM KV cache. *arXiv preprint arXiv:2403.04643*, 2024.

[15] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference. *arXiv preprint arXiv:2407.11550*, 2024.

[16] Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S Kevin Zhou. Identify critical KV cache in LLM inference from an output perturbation perspective. *arXiv preprint arXiv:2502.03805*, 2025.

[17] Zijie Geng, Jie Wang, Ziqi Liu, Feng Ju, Yiming Li, Xing Li, Mingxuan Yuan, Jianye Hao, Defu Lian, Enhong Chen, et al. Accurate KV cache eviction via anchor direction projection for efficient LLM inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[18] Raghavv Goel, Junyoung Park, Mukul Gagrani, Dalton Jones, Matthew Morse, Harper Langston, Mingu Lee, and Chris Lott. CAOTE: KV cache selection for LLMs via attention output error-based token eviction. *arXiv preprint arXiv:2504.14051*, 2025.

[19] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[20] Yuzhe Gu, Xiyu Liang, Jiaojiao Zhao, and Enmao Diao. OBCache: Optimal brain KV cache pruning for efficient long-context LLM inference. *arXiv preprint arXiv:2510.07651*, 2025.

[21] Hang Guo, Yawei Li, and Luca Benini. Optimal brain restoration for joint quantization and sparsification of llms. *arXiv preprint arXiv:2509.11177*, 2025.

[22] Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. ZipCache: Accurate and efficient KV cache quantization with salient token identification. *Advances in Neural Information Processing Systems*, 37:68287–68307, 2024.

[23] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

[24] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[25] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7B, 2023.

[26] Gregory Kamradt. Needle in a haystack - pressure testing LLMs, 2023.

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

[28] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2, 1989.

[29] Jianlong Lei and Shashikant Ilager. ARKV: Adaptive and resource-efficient KV cache management under limited memory budget for long-context inference in LLMs. *arXiv preprint arXiv:2603.08727*, 2026.

[30] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[31] Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on KV cache management. *arXiv preprint arXiv:2412.19442*, 2024.

[32] Xing Li, Zeyu Xing, Yiming Li, Linping Qu, Hui-Ling Zhen, Wulong Liu, Yiwu Yao, Sinno Jialin Pan, and Mingxuan Yuan. KVTuner: Sensitivity-aware layer-wise mixed-precision KV cache quantization for efficient and nearly lossless LLM inference. *arXiv preprint arXiv:2502.04420*, 2025.

[33] Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[34] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. DeepSeek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[35] Yanyu Liu, Jingying Fu, Sixiang Liu, Yitian Zou, Shouhua Zhang, and Jiehan Zhou. KV cache compression for inference efficiency in LLMs: A review. In *Proceedings of the 4th International Conference on Artificial Intelligence and Intelligent Information Processing*, pp. 207–212, 2025.

[36] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

[37] 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. *arXiv preprint arXiv:2402.02750*, 2024.

[38] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

[39] Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences. *arXiv preprint arXiv:2503.12491*, 2025.

[40] Claude E Shannon et al. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat. Conv. Rec*, 4(142-163):1, 1959.

[41] Alina Shutova, Vladimir Malinovskii, Vage Egiazarian, Denis Kuznedelev, Denis Mazur, Nikita Surkov, Ivan Ermakov, and Dan Alistarh. Cache me if you must: Adaptive key-value quantization for large language models. *arXiv preprint arXiv:2501.19392*, 2025.

[42] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[44] Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. Scope: Optimizing key-value cache compression in long-context generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10775–10790, 2025.

[45] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[46] 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. *arXiv preprint arXiv:2407.21018*, 2024.

[47] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[48] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025.

[49] 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.

[50] Jiebin Zhang, Dawei Zhu, Yifan Song, Wenhao Wu, Chuqiao Kuang, Xiaoguang Li, Lifeng Shang, Qun Liu, and Sujian Li. More tokens, lower precision: Towards the optimal token-precision trade-off in KV cache compression. *arXiv preprint arXiv:2412.12706*, 2024.

[51] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, et al. ∞ bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15262–15277, 2024.

[52] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

A PROOFS

This appendix collects the proofs for the results in Sec. 3.1. All quantities below are local to a single layer and KV head (ℓ, h), with indices suppressed.

Token weight in V cache.

Proof of Prop. 3.1. By definition, $\|a_\tau - \hat{a}_\tau\|_{\mathrm{TV}} = \frac{1}{2}\sum_{t'} |a_{\tau,t'} - \hat{a}_{\tau,t'}|$. The evicted entry contributes $|a_{\tau,t} - 0| = a_{\tau,t}$. Each surviving entry $t' \neq t$ shifts by $|\hat{a}_{\tau,t'} - a_{\tau,t'}| = a_{\tau,t'} \cdot a_{\tau,t}/(1 - a_{\tau,t})$. Summing the surviving terms,

$$\sum_{t' \neq t} a_{\tau,t'} \cdot \frac{a_{\tau,t}}{1 - a_{\tau,t}} = (1 - a_{\tau,t}) \cdot \frac{a_{\tau,t}}{1 - a_{\tau,t}} = a_{\tau,t}.$$

Therefore $\|a_\tau - \hat{a}_\tau\|_{\mathrm{TV}} = \frac{1}{2}(a_{\tau,t} + a_{\tau,t}) = a_{\tau,t}$. Summing over queries, $w_t = \sum_\tau a_{\tau,t}$. □

Channel weight in K cache.

Proof of Prop. 3.2. Removing channel c gives $\delta Z = -(1/\sqrt{d})\, Q[:, c]\, K[:, c]^\top$. Therefore,

$$\|\delta Z\|_F^2 = \frac{1}{d}\|Q[:, c]\, K[:, c]^\top\|_F^2 = \frac{1}{d}\|Q[:, c]\|_2^2\, \|K[:, c]\|_2^2 = \frac{1}{d} w_c.$$

Averaging over logit entries only introduces a constant factor, so the average squared logit deviation is proportional to w_c. □

Bennett high-rate distortion.

Lemma A.1 (Bennett high-rate distortion)**.** *For a uniform scalar quantizer with per-unit dynamic range R_u at bit-width b (high-rate regime), the per-coordinate mean-squared error is $\sigma_u^2\, 2^{-2b} + o(2^{-2b})$ with $\sigma_u := R_u/(2\sqrt{3})$.*

Proof. With per-unit scale, all n_u coordinates within a unit share a single uniform quantizer of cell width $\Delta = R_u/2^b$. Bennett's high-rate uniform-cell approximation [6] gives the per-coordinate mean-squared error as

$$\Delta^2/12 + o(\Delta^2) = \sigma_u^2\, 2^{-2b} + o(2^{-2b}).$$

□

Optimal allocation.

Proof of Thm. 3.3. The objective $\sum_u w_u \sigma_u^2\, 2^{-2b_u}$ is strictly decreasing in each b_u, so the budget is active at any optimum: $\sum_u b_u^\star = B$. Each summand $w_u \sigma_u^2\, 2^{-2b_u}$ is convex in b_u (second derivative $4(\ln 2)^2 w_u \sigma_u^2\, 2^{-2b_u} > 0$), so the objective is convex and KKT is sufficient for global optimality. The Lagrangian, with $\lambda \geq 0$ for the budget and $\mu_u \geq 0$ for nonnegativity, is

$$\mathcal{L}(\{b_u\}, \lambda, \{\mu_u\}) = \sum_u w_u \sigma_u^2\, 2^{-2b_u} + \lambda\Big(\sum_u b_u - B\Big) - \sum_u \mu_u\, b_u.$$

Stationarity in b_u gives $-2(\ln 2)\, w_u \sigma_u^2\, 2^{-2b_u} + \lambda - \mu_u = 0$. For an interior solution ($\mu_u = 0$ by complementary slackness),

$$2^{-2b_u} = \frac{\lambda}{2 \ln 2 \cdot w_u \sigma_u^2}, \qquad \text{i.e.,} \qquad b_u = \frac{1}{2}\log_2\left(\frac{2 \ln 2 \cdot w_u \sigma_u^2}{\lambda}\right).$$

When this interior value is negative, the boundary $b_u^\star = 0$ activates instead, which occurs iff $w_u \sigma_u^2 < \lambda/(2 \ln 2)$. The two cases combine into the $[\cdot]_+$ projection. Since $\sum_u b_u^\star(\lambda)$ is monotone non-increasing in λ, the budget equation $\sum_u b_u^\star(\lambda) = B$ is solvable by 1D search. □

Weak duality for discrete allocation.

Proposition A.2 (Weak duality). *For* $\mathrm{OPT} := \min_{b_u \in \mathcal{B}} \sum_u w_u\, \varepsilon_u(b_u)$ *s.t.* $\sum_u b_u \le B$ *with* $\mathcal{B} = \{0, 2, 4, 8, 16\}$ *and* $\varepsilon_u(b)$ *the calibration-measured per-coordinate distortion at bit-width* b*, let* $g(\lambda) := \sum_u \min_{b \in \mathcal{B}} [w_u\, \varepsilon_u(b) + \lambda b] - \lambda B$*. Then* $g(\lambda) \le \mathrm{OPT}$ *for every* $\lambda \ge 0$*; when* $\{b_u^\star(\lambda)\}$ *is primal feasible,* $\sum_u w_u \varepsilon_u(b_u^\star(\lambda))$ *is an upper bound.*

Proof. For any feasible $\{b_u\}$ and $\lambda \ge 0$,

$$\sum_u w_u \varepsilon_u(b_u) \;\ge\; \sum_u w_u \varepsilon_u(b_u) + \lambda\Big(\sum_u b_u - B\Big) \;\ge\; \sum_u \min_{b \in \mathcal{B}} [w_u \varepsilon_u(b) + \lambda b] - \lambda B = g(\lambda),$$

using $\sum_u b_u - B \le 0$ in the first inequality and per-unit minimization in the second. Taking minimum over feasible primals yields $\mathrm{OPT} \ge g(\lambda)$. Conversely, any feasible $\{b_u^\star(\lambda)\}$ satisfies $\sum_u w_u \varepsilon_u(b_u^\star(\lambda)) \ge \mathrm{OPT}$ by definition. □

B Detailed Implementation

This appendix expands on the implementation details abbreviated in Sec. 4.

Prefill stage. Probe-based score computation, rate-distortion bit allocation, and TriZone packing are performed once at the end of prefill, on the same hidden states already produced by the forward pass. The probe shares the SnapKV observation window of size $S_w = 32$ and pooling kernel $w = 5$ across all eviction-style baselines (SnapKV, AdaKV, ThinK, SnapKV+ZipCache) to ensure that score-based comparisons isolate the indicator and the allocation, not the probe geometry. The bit-width set is fixed to $\mathcal{B} = \{0, 2, 4, 8, 16\}$. The K/V budget is split equally, $B^V = B^K = \frac{1}{2} B_{\text{head}}$, as a symmetric default validated by the ablation in Sec. H.2, which shows that $r_K = 0.5$ consistently achieves the highest average score across budget levels.

Calibration. The empirical per-coordinate distortion table $\varepsilon_u(b)$ is estimated on calibration sequences drawn from the prefill prefixes of the LongBench tasks. We use 32 sequences (truncated to 4k tokens each) to estimate $\varepsilon_u(b)$ for each $b \in \{2, 4, 8\}$ at the per-token (V) and per-channel (K) granularities. The estimate at $b = 0$ is computed from the same sample but refers to outright removal rather than quantization, as discussed in Sec. 3. Calibration is performed once per (ℓ, h) slice and cached to disk.

Lagrangian search. The discrete allocation problem in Eq. (6) is solved by Lagrangian relaxation with one-dimensional bisection on λ. At each step we evaluate the per-element minimizer $b_u^\star(\lambda) = \arg\min_{b \in \mathcal{B}} w_u \varepsilon_u(b) + \lambda b$ via table lookup, and bisect on λ until the average bit budget $\frac{1}{|U|} \sum_u b_u^\star(\lambda)$ matches the target B within a relative tolerance of 10^{-2}, or after at most 64 bisection steps. The resulting allocation is treated as the final $b_u^\star$.

Hardware. All models up to 8B parameters (LLaMA-3.1-8B-Instruct, Mistral-7B-Instruct-v0.3, Qwen3-4B) run on a single NVIDIA A100 64 GB GPU. LLaMA-2-13B-Chat and Qwen2.5-72B-Instruct run on NVIDIA GH200 96 GB GPUs; Qwen2.5-72B uses 2-GPU tensor parallelism. Decoding latency and peak memory measurements (Sec. 4, Fig. 5) are collected on the A100 64 GB.

Baseline implementations. SnapKV and AdaKV use the official `kvpress` implementations. ThinK [46] and ZipCache [22] use their respective official open-source code. SnapKV+ZipCache combines the `kvpress` SnapKV with the official ZipCache mixed-precision quantizer, with no code modifications. All baselines use the default hyperparameters recommended by their respective authors. All methods receive the same prefill probe ($S_w = 32$, $w = 5$) so that any difference in the resulting weights is attributable to the indicator, not the calibration window.

Baseline configurations and budget accounting. Matched budgets are computed from nominal bit-widths; quantization metadata such as scales and zero-points is excluded for all methods.

Baseline selection rationale. For the eviction+quantization slot we compose SnapKV [33] with ZipCache [22] rather than adopting QPruningKV [50] directly. QPruningKV focuses on discussing the combination of existing eviction and quantization methods rather than proposing a new joint algorithm. In addition, the quantization methods it adopts assign precision only at layer level [37], a coarse granularity consistently outperformed by token-level alternatives such as ZipCache [22].

TriZone packed layout. Within Zone A (Sec. 3.3), V rows are sorted by b_t^V into three contiguous sub-segments, each with a distinct byte-packing convention over the head dimension d: 2-bit uses

quarter-split packing (four channels per byte at bit offsets $0, 2, 4, 6$; $d/4$ packed bytes per row), 4-bit uses *half-split* packing (two channels per byte at offsets $0, 4$; $d/2$ bytes per row), and 8-bit stores one channel per byte directly. Each sub-segment admits a single shift-and-mask dequantization path with no per-element branching. K channels within each retained row follow the per-channel allocation b_c^K: channels are sorted by bit-width into three analogous 2/4/8-bit segments along the channel axis, and the query q_τ is permuted to the same sorted order once at the end of prefill.

K-cache dequantization fusion. Per-channel quantization encodes channel c of the retained K cache as $\hat{k}_{t,c} = s_c\,(\tilde{k}_{t,c} - z_c)$, where s_c, z_c are the per-channel scale and zero-point and $\tilde{k}_{t,c} \in \{0, \ldots, 2^{b_c^K} - 1\}$ is the raw integer code. The kernel avoids materializing a dequantized FP16 tile by rewriting the dot product as

$$q_\tau^\top \hat{k}_t \;=\; \textstyle\sum_c (s_c\, q_{\tau,c})\; \tilde{k}_{t,c} \;-\; \sum_c s_c\, z_c\, q_{\tau,c}\,.$$

The scaled query $\tilde{q}_c = s_c\, q_{\tau,c}$ is computed once per decode step and cast to FP16 for the tensor-core matrix multiply. Raw codes $\tilde{k}_{t,c}$ are cast exactly to FP16 (all values ≤ 255) and serve as the right operand. The second sum is a per-query-head bias independent of t, subtracted once after the dot product accumulates over all retained tokens. This avoids one write/read pass of $|\mathcal{T}_{\text{kept}}| \times d$ FP16 values per decode step.

Padding and alignment. Bit-packing imposes alignment constraints on K-channel segment sizes: the 2-bit segment is zero-padded so that the channel count is a multiple of 4 (four channels per packed byte), and the 4-bit segment is padded to a multiple of 2. Padded channels receive $s_c = z_c = 0$ and thus contribute zero to both the dot product and the bias term. The 8-bit segment has no alignment constraint. V sub-segments are padded analogously along d.

Masking. Two boolean masks guard out-of-range accesses in the Triton kernels: a token-axis mask ($i < T_{\text{eff}}$) zeroes loads beyond the effective token count, and a channel-axis mask ($j < N_{\text{ch}}$, K-side only) zeroes loads beyond the true segment width. Since padded channels have $s_c = z_c = 0$, masked positions dequantize to zero and contribute nothing to the final result.

C Algorithm

Algorithm 1 summarizes the full RDKV pipeline executed once at the end of prefill. The algorithm takes the prefill KV cache and a per-head bit budget as input, and produces the TriZone packed cache used during decoding.

MCKP (Alg. 2) solves the multiple-choice knapsack via Lagrangian bisection on λ. For each candidate λ, every unit independently picks the bit-width minimizing its cost $w_u \varepsilon_u(b) + \lambda b$. The bisection terminates when the average bit-width is within tolerance of the budget.

D Additional Experiments on LongBench

In this section, we provide comprehensive experimental results on LongBench [5], a benchmark focused on long-context understanding with 16 tasks spanning single-document QA, multi-document QA, summarization, few-shot learning, synthetic retrieval, and code completion. We perform detailed evaluations with cache budgets ranging from $64L$ to $1024L$ on four additional models beyond the primary LLaMA-3.1-8B-Instruct reported in Sec. 4.1: Mistral-7B-Instruct-v0.3 [25] and Qwen3-4B [47] to test cross-architecture generality, and LLaMA-2-13B-Chat [42] and Qwen2.5-72B-Instruct [48] (2-GPU tensor parallelism) to test cross-scale generality. All methods share the same probe configuration ($S_w = 32$, $w = 5$) so that any performance difference is attributable to the indicator and allocation, not the probe geometry.

Tables 6 to 8 present the detailed per-task scores. Overall, RDKV achieves the highest average at every (model, budget) combination—without architecture-specific tuning. The advantage is largest under aggressive compression: at $B_{\text{total}} = 64L$, RDKV leads the strongest baseline by 1.5/5.5/2.7 points on Mistral-7B/Qwen3-4B/LLaMA-3.1-8B, because RDKV retains moderate-importance tokens at low bit-width rather than evicting them entirely. At $B_{\text{total}} = 1024L$ it recovers 98.6–99.4% of FullKV across models; on the two larger models (LLaMA-2-13B, Qwen2.5-72B) the recovery reaches 99.5–99.6% already at $B_{\text{total}} = 512L$. The consistency across architectures and scales suggests that the

Algorithm 1 RDKV: Rate-Distortion KV Cache Compression

Require: Prefill KV cache $K^{(\ell)}, V^{(\ell)}$ for each layer ℓ; per-head budget B_{head}; observation window size S_w; pooling kernel w; bit-width set $\mathcal{B} = \{0, 2, 4, 8, 16\}$; empirical distortion tables $\varepsilon^K(b), \varepsilon^V(b)$

Ensure: TriZone packed cache for each (ℓ, h)

for each layer $\ell = 1, \ldots, L$ **do**
  // Stage 1: Weight computation
  $A \leftarrow \mathrm{Softmax}\big(Q^{(\ell)}_{[\tau - S_w:\tau]} K^{(\ell)\top} / \sqrt{d}\big)$
  for each KV head $h = 1, \ldots, H_{\text{kv}}$ **do**
    $w_t^{(h)} \leftarrow \sum_{\tau, g} a^{(h)}_{\tau, g, t}$ for all t ▷ token weight in V cache
    $w_t^{(h)} \leftarrow \mathrm{AvgPool1d}(w_t^{(h)},\ w)$
    $w_c^{(h)} \leftarrow \|Q^{(h)}_{:,c}\|_2^2 \cdot \|K^{(h)}_{:,c}\|_2^2$ for all c ▷ channel weight in K cache
  end for

  // Stage 2: V-side token allocation (per head)
  $B^V \leftarrow \frac{1}{2} B_{\text{head}}; \quad \bar{B}^V \leftarrow B^V / d$
  for each KV head h **do**
    $\{b_t^V\}_h \leftarrow \mathrm{MCKP}(w_t^{(h)},\ \varepsilon^V,\ \bar{B}^V / T)$
    $\mathcal{T}^{(h)}_{\text{kept}} \leftarrow \{t : b_t^V > 0\}$
  end for

  // Stage 3: K-side channel allocation (per head)
  $B^K \leftarrow \frac{1}{2} B_{\text{head}}$
  for each KV head h **do**
    $k^{(h)}_{\text{avg}} \leftarrow B^K \,/\, \big(|\mathcal{T}^{(h)}_{\text{kept}}| \cdot d\big)$
    $\{b_c^K\}_h \leftarrow \mathrm{MCKP}(w_c^{(h)},\ \varepsilon^K,\ k^{(h)}_{\text{avg}})$
  end for

  // Stage 4: TriZone packing
  for each KV head h **do**
    Sort $\mathcal{T}^{(h)}_{\text{kept}}$ by b_t^V into sub-segments $\mathcal{S}_2, \mathcal{S}_4, \mathcal{S}_8$
    Sort channels by b_c^K into segments; permute q to match
    Quantize and byte-pack V sub-segments $\rightarrow$ Zone A (V)
    Quantize and byte-pack K rows of $\mathcal{T}^{(h)}_{\text{kept}} \rightarrow$ Zone A (K)
    Store $\{t : b_t^V = 16\}$ V rows in FP16 $\rightarrow$ Zone B
  end for
end for

distortion weights (Sec. 3) capture a model-agnostic signal, enabling the allocation to transfer without modification.

E Additional Experiments on RULER

In this section, we present a detailed evaluation of RDKV on the various subtasks of the RULER benchmark [24]. RULER is specifically designed to assess core long-context capabilities through a diverse suite of tasks. The retrieval suite includes four variants of needle-in-a-haystack tests: Single-Needle (S-NIAH-1/2/3) at increasing difficulty, Multi-Key (MK-NIAH-1/2/3), Multi-Query (MQ-NIAH), and Multi-Value (MV-NIAH), which evaluate recall accuracy under diverse distractor settings and query formulations. Beyond retrieval, Variable Tracking (VT) measures multi-hop reasoning by requiring models to resolve transitive variable references scattered throughout the input. Lastly, the aggregation tasks Common Word Extraction (CWE) and Frequent Word Extraction (FWE) test the ability to compress and synthesize high-density signal distributed across long contexts. These

Algorithm 2 MCKP: Lagrangian Bisection Knapsack Solver

Require: Weights $\{w_u\}$; distortion table $\varepsilon(b)$; target average bits $\bar{b}$; bit-width set $\mathcal{B}$; tolerance δ; max iterations I
Ensure: Bit-width assignment $\{b_u^\star\}$

$\lambda_{\text{lo}} \leftarrow 0$; $\lambda_{\text{hi}} \leftarrow \max_u w_u$
for $i = 1, \dots, I$ **do**
  $\lambda \leftarrow (\lambda_{\text{lo}} + \lambda_{\text{hi}})/2$
  for each unit u **do**
    $b_u^\star \leftarrow \arg\min_{b \in \mathcal{B}} \; w_u\, \varepsilon(b) + \lambda\, b$
  end for
  $\bar{b}_{\text{cur}} \leftarrow \text{mean}(\{b_u^\star\})$
  if $|\bar{b}_{\text{cur}} - \bar{b}|/\bar{b} < \delta$ **then**
    return $\{b_u^\star\}$
  else if $\bar{b}_{\text{cur}} > \bar{b}$ **then**
    $\lambda_{\text{lo}} \leftarrow \lambda$
  else
    $\lambda_{\text{hi}} \leftarrow \lambda$
  end if
end for
return $\{b_u^\star\}$

tasks collectively pose distinct challenges for context retention, salience estimation, and compositional reasoning.

We evaluate RDKV on LLaMA-3.1-8B-Instruct with $B_{\text{total}} = 1024L$, across input lengths from 4k to 128k tokens. The evaluation compares RDKV with SnapKV, AdaKV, ThinK, SnapKV+ZipCache, and FullKV (oracle).

As reported in Tab. 9, RDKV achieves the highest average accuracy among compression methods at every context length. At 4k and 8k, RDKV matches FullKV (98.62/98.61 vs. 98.58/98.88), while the best baseline reaches only 92.03 (AdaKV) and 86.44 (ThinK). The margin over the strongest baseline peaks at 14.2 points at 16k (95.65 vs. 81.45) and remains 4.1 points at 128k (66.95 vs. 62.85).

A breakdown by task clarifies the source of the advantage. On S-NIAH-3, RDKV stays at 100 through 32k and 98.61 at 64k, whereas the eviction-only baselines fall to 53–70 at 64k; at 128k RDKV reaches 63.19 against at most 29.05 for all baselines. The gap is largest on MK-NIAH-3, where distractor keys compete with the target: at 16k RDKV scores 79.70 vs. at most 24.07 for the baselines. On the CWE aggregation task at 16k, eviction discards the low-attention tokens that carry the frequency signal, leaving the eviction-only baselines at 22.60–30.35; RDKV retains these tokens at low bit-width and scores 76.92, 17.8 points above the next best method (SnapKV+ZipCache, 59.15) and above FullKV (73.38). FWE is the exception: at 64k and 128k, RDKV trails the best baseline (71.47 vs. 78.09 and 31.20 vs. 52.32).

Comparison with HqeKV. HqeKV [4] is a concurrent method that also combines eviction and quantization, but fixes the tier ratios rather than deriving them from a single allocation curve. Tab. 10 compares RDKV and HqeKV on the 11 RULER tasks at $B_{\text{total}} = 1024L$ across three context lengths on LLaMA-3.1-8B-Instruct. RDKV leads on average at every length, by $+5.3$ / $+1.9$ / $+5.0$ points at 16k / 32k / 64k. The advantage is concentrated on aggregation and multi-target retrieval: on CWE, RDKV leads by $+50.1$ / $+19.5$ / $+3.8$ points; on MV-NIAH, by $+4.9$ / $+9.0$ / $+14.9$; on MQ-NIAH, by $+3.3$ / $+6.2$ / $+13.6$. HqeKV outperforms RDKV on MK-NIAH-3 at all three lengths and on FWE at 32k and 64k, where its fixed high-precision tier retains more tokens at full bit-width; RDKV's allocator trades off these tokens for finer-grained precision elsewhere.

F Additional Experiments on Needle-in-a-Haystack

Figure 6 extends the Needle-in-a-Haystack evaluation of Sec. 4.1 to a larger cache budget $B_{\text{total}} = 128L$. Even with the relaxed budget, SnapKV and AdaKV still exhibit failure bands at intermediate depths (11–89%) for longer contexts: a needle that does not rank among the top-k tokens is evicted

Table 6: Performance on 16 LongBench datasets for Mistral-7B-Instruct-v0.3 across cache budgets $B_{\text{total}} \in \{64L, 128L, 256L, 512L, 1024L\}$. Snap+Zip denotes SnapKV+ZipCache. The best result in each row is in **bold**; the second-best is underlined.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Mistral-7B-Instruct-v0.3, B_{total} = Full																	
FullKV	25.91	38.13	49.62	52.07	39.03	28.10	34.11	25.73	26.47	76.00	88.59	47.46	7.01	98.00	61.58	62.34	47.51
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 64L$																	
SnapKV	17.53	17.78	32.36	40.94	30.84	19.34	16.27	20.33	13.77	38.00	87.64	38.80	**5.00**	76.50	48.46	47.72	34.45
AdaKV	17.33	18.43	34.33	42.13	31.16	21.08	16.79	20.64	14.23	38.00	88.18	39.61	3.50	83.00	51.28	48.93	35.54
ThinK	20.27	24.90	40.42	45.00	33.54	21.44	19.83	21.26	18.45	39.50	87.94	40.66	3.00	83.00	53.19	51.51	37.74
Snap+Zip	22.80	25.83	**47.55**	**47.38**	**35.23**	**24.03**	21.54	21.59	21.61	55.00	89.05	44.84	4.00	88.00	56.59	56.61	41.35
RDKV	**24.70**	**29.46**	47.51	46.48	34.37	22.73	**23.40**	**23.23**	**22.73**	**60.00**	**89.07**	**44.99**	4.50	**96.00**	**57.78**	**58.19**	**42.82**
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 128L$																	
SnapKV	21.25	22.92	42.30	44.75	33.34	21.24	21.00	21.37	19.30	44.50	**89.35**	42.47	4.50	91.00	55.98	54.12	39.34
AdaKV	21.97	23.31	46.28	46.67	34.52	21.47	20.92	21.85	19.71	49.00	89.19	42.63	**6.50**	91.50	56.41	55.60	40.47
ThinK	21.90	29.83	48.32	47.25	34.76	23.27	21.73	22.37	21.18	47.50	88.36	43.10	5.00	93.00	57.62	56.39	41.35
Snap+Zip	23.62	32.30	48.55	**50.23**	35.41	**26.96**	24.22	22.23	23.58	67.50	88.63	45.19	5.50	92.00	59.01	58.30	43.95
RDKV	**25.61**	**33.76**	**48.58**	48.65	**36.19**	25.37	**25.39**	**24.14**	**24.14**	**70.00**	89.18	**45.52**	5.00	**97.50**	**59.30**	**59.69**	**44.88**
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 256L$																	
SnapKV	24.37	29.08	47.68	48.14	34.66	24.45	23.17	23.24	22.37	57.00	88.85	44.36	**6.00**	96.50	59.32	57.98	42.95
AdaKV	24.71	30.19	48.31	49.08	34.69	**26.03**	23.13	23.46	22.19	66.00	88.93	44.57	4.50	96.50	59.38	59.67	43.83
ThinK	24.82	31.21	**50.22**	**50.79**	35.00	23.70	23.75	23.55	23.17	66.00	**89.61**	43.90	5.50	94.50	59.37	59.04	44.01
Snap+Zip	25.85	**36.05**	48.22	49.31	**36.59**	24.75	27.34	23.59	**25.26**	71.00	88.75	45.37	5.50	94.50	**59.76**	59.88	45.11
RDKV	**26.00**	35.68	49.89	49.11	35.50	25.99	**27.78**	**24.48**	25.22	**73.50**	88.58	**46.40**	**6.00**	**97.00**	59.49	**60.60**	**45.70**
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 512L$																	
SnapKV	25.89	32.99	48.59	50.39	36.72	26.24	24.86	23.81	24.17	69.00	**89.28**	45.39	**6.00**	**97.50**	60.21	61.03	45.13
AdaKV	26.04	33.06	**50.75**	50.29	35.81	26.58	25.03	23.39	23.65	71.50	88.86	46.03	**6.00**	**97.50**	**60.74**	**62.25**	45.47
ThinK	27.21	36.41	49.96	50.54	**37.33**	25.04	25.82	23.87	24.70	72.50	88.94	45.01	**6.00**	96.00	60.58	62.08	45.75
Snap+Zip	**27.57**	35.17	48.23	**51.43**	36.70	25.40	28.66	24.37	25.61	**75.50**	89.01	46.09	**6.00**	95.50	60.34	60.70	46.02
RDKV	26.06	**37.26**	49.62	50.07	36.58	**26.99**	**30.48**	**25.14**	**26.23**	**75.50**	89.08	**46.77**	5.50	96.50	59.14	60.58	**46.34**
Mistral-7B-Instruct-v0.3, $B_{\text{total}} = 1024L$																	
SnapKV	26.05	35.63	**50.05**	50.39	36.44	27.40	27.24	23.97	25.22	73.00	**89.19**	45.84	6.00	**98.50**	61.42	61.55	46.12
AdaKV	26.31	36.49	49.95	**51.76**	37.37	27.07	27.11	24.84	25.01	73.00	89.03	45.83	5.50	**98.50**	61.03	**62.35**	46.32
ThinK	**27.97**	**38.68**	49.96	49.97	**38.69**	25.28	28.05	24.57	25.75	74.00	88.72	45.83	**6.50**	95.00	**61.67**	61.85	46.41
Snap+Zip	26.41	37.75	48.90	51.54	35.26	25.74	30.97	24.25	26.07	**76.00**	88.70	46.25	6.06	95.50	59.18	60.71	46.21
RDKV	27.00	37.96	49.88	50.56	37.63	**28.20**	**32.33**	**25.05**	**26.57**	**76.00**	88.48	**47.88**	5.00	97.50	59.35	60.40	**46.86**

regardless of the budget headroom. RDKV preserves a near-uniform success pattern comparable to FullKV; the residual misses concentrate at depths and lengths where FullKV itself begins to lose recall.

G Additional Experiments on InfiniteBench

In this section, we present a detailed evaluation of RDKV on the InfiniteBench benchmark [51], which extends long-context evaluation to sequences exceeding 100k tokens. The 10 tasks span three categories: (i) retrieval—Passkey Retrieval (Retr.Pass), Number Retrieval (Retr.Num), and KV Retrieval (Retr.KV)— which test the ability to locate and extract specific information embedded in long contexts; (ii) language understanding—English Dialogue (En.Dia), English Summarization (En.Sum), English Multiple-Choice (En.MC), English QA (En.QA), and Chinese QA (Zh.QA)—which evaluate comprehension and reasoning over novel-length inputs; and (iii) structured reasoning—Math Find (Math.Find) and Code Debugging (Debug)—which require precise numerical or logical retrieval. These tasks collectively pose challenges for both context retention and fine-grained information extraction at extreme sequence lengths.

We evaluate RDKV on LLaMA-3.1-8B-Instruct with $B_{\text{total}} = 1024L$. The evaluation compares RDKV with SnapKV, AdaKV, ThinK, SnapKV+ZipCache, and FullKV (oracle).

As reported in Tab. 11, RDKV achieves the highest average (39.46) among all compression methods, leading AdaKV (38.06) by 1.40 points. A breakdown by task category reveals where the advantage

Table 7: Performance on 16 LongBench datasets for Qwen3-4B-Instruct-2507 across cache budgets $B_{\text{total}} \in \{64L, 128L, 256L, 512L, 1024L\}$. Snap+Zip denotes SnapKV+ZipCache. The best result in each row is in **bold**; the second-best is underlined.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Qwen3-4B-Instruct-2507, B_{total} = Full																	
FullKV	27.96	44.61	49.84	58.84	43.27	25.54	30.64	22.56	24.06	74.50	87.15	45.36	2.15	100.00	64.69	57.56	47.42
Qwen3-4B-Instruct-2507, $B_{\text{total}} = 64L$																	
SnapKV	16.95	26.10	35.04	43.94	34.68	15.70	12.69	19.63	12.69	39.50	74.68	35.99	1.51	45.50	53.03	45.78	32.09
AdaKV	16.83	25.08	36.52	45.79	35.94	15.06	12.97	19.40	12.97	42.50	76.67	36.46	**2.73**	47.00	53.54	45.54	32.81
ThinK	18.65	28.98	39.41	51.50	39.33	19.25	14.72	20.81	14.53	44.50	80.45	38.43	0.93	85.50	55.67	48.79	37.59
Snap+Zip	18.22	31.79	46.14	52.48	38.27	20.22	**20.43**	21.16	19.33	61.00	84.02	40.83	1.67	60.29	**60.95**	51.23	39.25
RDKV	**24.94**	**39.23**	**49.24**	**59.71**	**43.46**	**25.50**	20.01	**23.15**	**20.18**	**70.50**	**84.64**	**41.88**	0.75	**100.00**	60.43	**52.29**	**44.74**
Qwen3-4B-Instruct-2507, $B_{\text{total}} = 128L$																	
SnapKV	21.46	31.15	43.96	53.83	40.79	22.12	15.77	21.90	16.16	51.50	83.44	39.72	1.30	93.03	59.07	50.75	40.37
AdaKV	23.78	31.71	45.03	57.81	40.54	23.84	16.40	22.70	16.45	56.50	81.10	40.47	1.60	98.00	60.64	51.76	41.77
ThinK	22.88	35.83	44.32	56.70	40.32	23.61	18.68	22.77	17.87	59.00	84.64	40.93	1.32	97.00	61.76	52.80	42.53
Snap+Zip	23.84	38.35	48.62	57.59	41.88	24.04	23.43	22.18	21.62	68.50	85.73	42.24	**1.91**	98.00	62.43	53.41	44.61
RDKV	**26.79**	**41.89**	**49.87**	**59.14**	**42.95**	**26.02**	**24.01**	**23.39**	**22.46**	**74.00**	**86.53**	**43.00**	1.25	**100.00**	**63.10**	**53.48**	**46.12**
Qwen3-4B-Instruct-2507, $B_{\text{total}} = 256L$																	
SnapKV	25.92	36.55	48.56	59.14	41.07	25.18	19.13	22.75	19.34	65.00	84.21	40.26	1.72	99.75	62.46	53.74	44.05
AdaKV	25.46	38.20	48.10	59.34	42.49	**26.44**	19.46	23.01	19.60	69.00	86.32	40.80	1.75	**100.00**	63.18	55.07	44.89
ThinK	25.30	39.37	46.06	59.10	42.73	25.10	21.58	23.14	20.80	68.50	86.58	41.22	1.88	**100.00**	63.50	**55.32**	45.01
Snap+Zip	24.21	40.99	48.26	58.93	42.75	24.22	25.96	22.62	23.13	72.50	**87.13**	43.42	**1.90**	99.50	**63.77**	54.68	45.87
RDKV	**26.45**	**43.49**	**50.30**	**59.60**	**42.91**	25.52	**27.30**	**23.80**	**23.30**	**74.00**	87.03	**44.27**	1.25	**100.00**	63.39	53.91	**46.66**
Qwen3-4B-Instruct-2507, $B_{\text{total}} = 512L$																	
SnapKV	26.57	39.86	48.52	59.27	43.23	26.10	23.00	22.74	21.67	70.00	**87.95**	41.81	1.99	**100.00**	64.72	56.35	45.86
AdaKV	26.72	40.87	48.17	59.10	42.93	**26.61**	23.28	22.58	21.54	70.50	87.15	42.20	1.78	**100.00**	64.69	**56.58**	45.92
ThinK	26.48	41.08	46.44	59.15	42.90	25.06	24.79	23.38	22.38	72.50	87.64	42.51	**2.06**	**100.00**	**65.07**	56.34	46.11
Snap+Zip	25.47	42.27	47.80	57.58	43.17	23.24	28.44	22.60	23.79	**75.00**	86.82	43.79	1.94	99.50	65.03	56.52	46.44
RDKV	**27.75**	**43.89**	**49.81**	**59.97**	**43.87**	25.37	**30.07**	**23.67**	**24.00**	74.50	86.73	**44.60**	0.50	**100.00**	63.29	54.42	**47.03**
Qwen3-4B-Instruct-2507, $B_{\text{total}} = 1024L$																	
SnapKV	**27.80**	42.45	48.56	59.01	43.04	25.83	26.55	23.26	22.91	72.50	87.44	42.80	1.83	**100.00**	64.83	56.65	46.59
AdaKV	27.64	42.54	48.65	**59.57**	43.31	**26.35**	26.75	23.02	23.16	74.00	**88.08**	43.49	1.60	**100.00**	64.86	**57.44**	46.90
ThinK	26.73	42.98	47.88	58.62	43.18	24.97	27.90	**23.55**	23.43	**75.00**	87.85	42.95	1.97	**100.00**	**65.48**	57.04	46.85
Snap+Zip	26.60	43.46	47.80	**59.57**	**43.76**	22.11	29.78	22.67	23.84	73.50	87.36	44.65	**2.16**	99.50	64.09	56.08	46.68
RDKV	27.35	**44.36**	**50.40**	59.35	43.63	24.87	**30.91**	23.29	**24.45**	74.50	87.26	**45.24**	0.75	**100.00**	62.86	54.61	**47.12**

concentrates. On the retrieval tasks, RDKV scores 7.20 on KV Retrieval vs. ≤ 2.20 for all baselines—a $3.3\times$ improvement—because the needle-style retrieval pattern aligns directly with the rate-distortion allocation: high-attention tokens are preserved at high precision while low-attention tokens are aggressively quantized or removed. On language understanding tasks, RDKV leads on En.Dia (13.00 vs. $\leq$ 11.50), En.Sum (25.13 vs. $\leq$ 23.66), and En.QA (14.49 vs. $\leq$ 13.43), matching FullKV accuracy on En.MC (68.56). On structured reasoning tasks (Math.Find, Debug), all compression methods cluster near each other, indicating that cache compression is not the bottleneck once the relevant information is preserved.

H Additional Experiments on Ablation Study

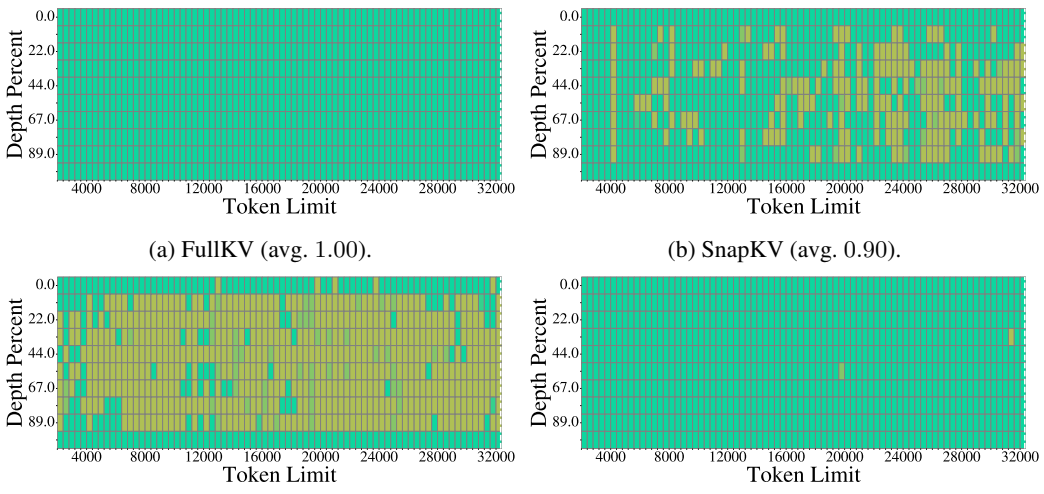
H.1 Prefill Overhead and End-to-End Latency

Tab. 12 breaks down the time-to-first-token (TTFT) for RDKV on LLaMA-3.1-8B-Instruct with a 128K-token prefill on a single A100 64 GB. The baseline is a standard FullKV prefill with FlashAttention-2. RDKV bypasses the HuggingFace `generate` wrapper and calls transformer layers directly, saving 593 ms of framework overhead. The four RDKV-specific stages—weight computation (w_t, w_c), MCKP bisection, and TriZone packing—add a combined 2,331 ms. The net overhead is 1,740 ms, or 6.0% of the FullKV prefill, yielding a measured TTFT of 30,583 ms.

This one-time cost is amortized over decoding. Tab. 13 reports end-to-end latency, including both prefill and decoding, for the same 128K input. With per-token decode latency of ~ 82 ms for FullKV

Table 8: Performance on 16 LongBench datasets for LLaMA-2-13B-Chat and Qwen2.5-72B-Instruct (2-GPU tensor parallelism) at $B_{\text{total}} \in \{128L, 512L\}$. Snap+Zip denotes SnapKV+ZipCache. The best result in each row is in **bold**; the second-best is underlined.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
LLaMA-2-13B-Chat, B_{total} = Full																	
FullKV	16.25	17.54	27.88	14.81	14.97	6.31	27.47	20.58	26.12	69.00	87.83	38.12	2.91	12.38	51.17	51.91	30.33
LLaMA-2-13B-Chat, $B_{\text{total}} = 128L$																	
SnapKV	13.96	16.25	25.98	15.90	14.40	5.78	19.40	20.07	20.52	41.00	86.12	34.97	2.78	12.13	46.23	44.95	26.28
AdaKV	14.71	**17.81**	**28.54**	16.27	**15.12**	5.52	19.71	20.19	21.51	62.50	87.02	34.93	3.06	10.25	49.59	48.58	28.46
ThinK	14.35	14.81	26.36	16.29	14.35	5.54	19.95	20.24	21.53	48.00	87.01	35.45	2.67	12.25	48.02	46.59	27.09
Snap+Zip	**16.75**	15.93	27.73	**16.40**	13.94	5.61	21.66	18.93	23.18	58.00	**87.49**	35.79	**5.90**	**17.38**	**50.33**	47.14	28.88
RDKV	16.03	16.40	27.09	15.17	15.01	**6.89**	**22.42**	**20.29**	**24.10**	**68.50**	87.05	**36.86**	3.01	13.88	49.83	**50.73**	**29.58**
LLaMA-2-13B-Chat, $B_{\text{total}} = 512L$																	
SnapKV	15.57	16.55	28.23	15.49	**15.15**	6.24	22.52	20.13	24.04	67.50	86.53	**38.47**	3.09	**13.88**	**50.49**	50.36	29.64
AdaKV	15.53	17.10	28.45	15.84	15.14	**6.38**	22.98	**20.68**	24.22	**69.50**	87.11	37.52	2.42	13.12	49.50	51.25	29.80
ThinK	15.52	17.64	28.49	16.12	14.88	6.29	23.16	19.95	24.52	68.00	86.75	37.97	2.84	**13.88**	50.10	50.60	29.79
Snap+Zip	13.18	14.74	28.41	**17.03**	13.03	5.84	24.33	20.29	25.32	68.50	86.76	38.19	**5.50**	9.50	49.65	50.99	29.45
RDKV	**16.35**	**18.08**	**28.78**	15.23	15.10	6.19	**25.54**	20.27	**25.84**	68.50	**87.39**	37.41	2.68	13.12	50.00	**52.67**	**30.20**
Qwen2.5-72B-Instruct (2-GPU TP), B_{total} = Full																	
FullKV	32.21	49.59	51.88	65.26	65.51	41.37	34.00	24.35	24.74	78.00	91.76	47.87	21.50	99.50	66.36	71.42	54.08
Qwen2.5-72B-Instruct (2-GPU TP), $B_{\text{total}} = 128L$																	
SnapKV	28.76	32.00	46.33	62.37	62.21	36.60	20.94	21.38	17.90	53.00	90.26	40.81	**23.00**	97.42	59.07	63.89	47.25
AdaKV	30.68	33.07	49.54	63.92	62.93	38.35	21.80	22.02	18.46	61.50	90.91	42.15	21.50	99.25	59.94	65.71	48.86
ThinK	28.02	29.54	46.30	62.17	61.47	37.73	20.72	21.27	17.70	47.00	90.28	40.22	21.50	96.17	58.42	63.51	46.38
Snap+Zip	30.60	**46.38**	**50.50**	**65.47**	**64.97**	**40.35**	26.64	22.30	**22.14**	73.00	91.86	**45.29**	19.50	99.00	64.03	68.52	51.91
RDKV	**31.76**	44.79	49.57	63.86	64.42	39.72	**26.93**	**23.49**	22.10	**76.00**	**92.16**	45.15	22.00	**99.50**	**65.48**	**69.80**	**52.30**
Qwen2.5-72B-Instruct (2-GPU TP), $B_{\text{total}} = 512L$																	
SnapKV	32.50	45.81	51.23	64.56	64.39	40.55	26.89	23.24	22.07	75.50	92.16	45.03	21.50	**99.50**	65.04	69.95	52.49
AdaKV	**33.78**	45.16	50.82	63.96	64.80	40.83	27.41	23.02	22.29	76.50	91.89	45.60	**22.00**	**99.50**	65.48	70.48	52.72
ThinK	31.03	43.89	50.81	64.59	64.58	39.35	26.33	23.42	21.92	73.50	92.19	45.76	21.00	99.17	65.34	69.89	52.05
Snap+Zip	32.50	**50.24**	**51.80**	64.21	64.33	**41.00**	31.63	23.87	**24.23**	**78.00**	**92.61**	46.28	21.50	**99.50**	65.03	69.85	53.54
RDKV	32.65	49.51	51.68	**64.77**	**64.95**	40.41	**31.78**	**24.21**	23.94	**78.00**	92.06	**47.91**	21.50	**99.50**	**66.34**	**71.90**	**53.82**



(a) FullKV (avg. 1.00).

(b) SnapKV (avg. 0.90).

(c) AdaKV (avg. 0.75).

(d) RDKV (avg. 0.99).

Figure 6: Needle-in-a-Haystack on LLaMA-3.1-8B-Instruct at $B_{\text{total}} = 128L$ and context length up to 32k. Green denotes successful retrieval; warmer colours denote partial or complete failure.

and ~ 18 ms for RDKV, RDKV breaks even within the first ~ 30 generated tokens and reduces end-to-end latency by $1.41\times$ at 256 generated tokens and $2.30\times$ at 1,024.

Table 9: Performance on the 11 RULER tasks across context lengths $\{4k, 8k, 16k, 32k, 64k, 128k\}$ for LLaMA-3.1-8B-Instruct at $B_{\text{total}} = 1024L$. The best result in each row is in **bold**; the second-best is underlined. FullKV (no compression) is reported as a reference upper bound and excluded from the ranking.

Method	S-NIAH-1	S-NIAH-2	S-NIAH-3	MK-NIAH-1	MK-NIAH-2	MK-NIAH-3	MQ-NIAH	MV-NIAH	VT	CWE	FWE	Avg.
Sequence length = 4k												
FullKV	100.00	100.00	100.00	100.00	100.00	99.80	99.80	99.70	99.96	98.92	86.20	98.58
SnapKV	**100.00**	**100.00**	87.45	**100.00**	99.65	44.63	99.80	99.40	99.81	85.63	67.91	89.48
AdaKV	99.98	99.98	88.99	99.98	99.58	65.79	99.54	99.44	99.86	86.87	72.32	92.03
ThinK	99.01	99.41	95.61	**100.00**	96.01	63.60	99.71	99.36	99.69	86.89	70.07	91.76
Snap+Zip	82.38	81.88	82.21	82.55	82.55	82.05	81.06	82.38	31.70	81.64	70.99	76.49
RDKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**99.96**	**99.90**	**100.00**	**99.38**	**85.58**	**98.62**
Sequence length = 8k												
FullKV	99.99	99.99	99.99	99.80	99.80	99.79	99.70	99.50	99.76	93.76	95.60	98.88
SnapKV	**100.00**	**100.00**	73.07	99.90	99.30	27.83	99.95	98.60	99.54	52.51	81.55	84.75
AdaKV	**100.00**	**100.00**	73.05	99.67	99.27	42.63	99.77	97.71	99.43	45.53	83.33	85.49
ThinK	99.79	99.79	86.20	99.39	98.19	38.40	98.79	98.10	98.67	52.52	81.00	86.44
Snap+Zip	80.78	84.72	85.40	85.06	85.40	78.04	83.18	84.89	34.06	78.64	76.07	77.84
RDKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**97.26**	**100.00**	**100.00**	**100.00**	**93.91**	**93.54**	**98.61**
Sequence length = 16k												
FullKV	100.00	100.00	100.00	100.00	99.99	99.80	99.70	98.70	99.68	73.38	95.53	96.98
SnapKV	99.99	97.99	56.79	99.59	99.79	14.40	99.49	98.04	98.31	30.35	89.99	80.43
AdaKV	**100.00**	98.70	54.65	99.30	99.90	22.42	99.35	98.95	98.94	22.60	92.89	80.70
ThinK	99.88	97.29	74.51	98.28	99.08	17.58	99.08	97.73	94.93	27.37	90.22	81.45
Snap+Zip	75.45	92.32	93.84	92.89	94.60	24.07	91.85	92.80	35.92	59.15	85.97	76.26
RDKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**79.70**	**100.00**	**100.00**	**100.00**	**76.92**	**95.53**	**95.65**
Sequence length = 32k												
FullKV	100.00	100.00	100.00	99.40	99.61	99.20	99.60	99.20	99.04	10.18	87.40	90.33
SnapKV	99.96	98.56	60.98	98.76	98.76	10.20	99.56	97.71	92.97	8.18	68.38	75.82
AdaKV	**100.00**	98.86	54.15	98.86	99.47	16.24	99.67	97.86	95.38	9.24	62.64	75.67
ThinK	**100.00**	95.81	74.21	97.41	98.21	12.20	98.32	98.01	89.37	6.12	65.68	75.94
Snap+Zip	95.55	99.46	**100.00**	**100.00**	**100.00**	11.74	99.15	**100.00**	37.69	7.52	74.44	75.05
RDKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	**57.66**	**100.00**	**100.00**	**100.00**	**34.82**	**78.60**	**88.28**
Sequence length = 64k												
FullKV	100.00	100.00	100.00	99.41	98.41	97.01	99.06	94.50	96.65	1.22	87.13	88.49
SnapKV	**100.00**	95.52	53.06	98.52	93.91	2.40	98.27	92.21	87.23	1.26	78.09	72.77
AdaKV	**100.00**	94.65	53.83	98.45	91.84	6.00	98.25	92.55	88.76	1.54	71.30	72.47
ThinK	99.98	93.79	69.99	95.59	93.19	3.60	97.19	92.54	73.59	0.72	76.66	72.44
Snap+Zip	93.14	96.86	**99.33**	99.33	78.68	6.81	95.93	**96.86**	74.35	0.31	**78.89**	74.59
RDKV	**100.00**	**100.00**	98.61	**100.00**	**96.95**	**19.06**	**100.00**	95.86	**94.88**	**3.94**	71.47	**80.07**
Sequence length = 128k												
FullKV	99.99	99.39	99.79	97.59	88.60	66.20	98.44	90.65	66.56	0.00	71.80	79.91
SnapKV	**100.00**	99.35	20.43	97.55	75.12	0.80	95.65	87.59	65.50	0.08	49.28	62.85
AdaKV	**100.00**	99.38	17.03	97.57	64.71	2.00	94.77	85.80	64.11	0.20	17.16	58.43
ThinK	**100.00**	99.58	29.05	97.17	76.34	0.80	95.42	86.90	61.67	0.14	44.28	62.85
Snap+Zip	**100.00**	**100.00**	28.58	99.60	57.78	0.00	92.56	**94.29**	60.73	**0.21**	**52.32**	62.37
RDKV	**100.00**	**100.00**	**63.19**	**100.00**	**82.76**	**3.19**	**100.00**	87.28	**68.76**	0.07	31.20	**66.95**

H.2 K/V Budget Split Ratio

The main experiments fix an equal K/V split ($B^V = B^K = \frac{1}{2}B_{\text{head}}$) as a design convention (Sec. B). To validate this choice, we sweep the K budget ratio $r_K \in \{0.4, 0.5, 0.6\}$ (so $B^K = r_K B_{\text{head}}$ and $B^V = (1 - r_K)B_{\text{head}}$) on LLaMA-3.1-8B-Instruct at three cache budgets $B_{\text{total}} \in \{128L, 256L, 512L\}$. All other settings (observation window $S_w = 32$, pooling kernel $w = 5$, $\mathcal{B} = \{0, 2, 4, 8, 16\}$) are kept identical.

Tab. 14 reports the 16-task LongBench average. At every budget level the equal split $r_K = 0.5$ achieves the highest score. The margins are small (≤ 0.63 points), indicating that the allocation is robust to moderate shifts in the K/V ratio. Shifting more budget toward K ($r_K = 0.6$) consistently hurts: the V cache carries per-token information that directly scales the attention output (Prop. 3.1), so starving it degrades quality faster than under-provisioning per-channel K precision. Conversely,

Table 10: RDKV vs. HqeKV [4] on the 11 RULER tasks across context lengths $\{16\text{k}, 32\text{k}, 64\text{k}\}$ for LLaMA-3.1-8B-Instruct at $B_{\text{total}} = 1024L$. The best result in each row is in **bold**.

Method	S-NIAH-1	S-NIAH-2	S-NIAH-3	MK-NIAH-1	MK-NIAH-2	MK-NIAH-3	MQ-NIAH	MV-NIAH	VT	CWE	FWE	Avg.
Sequence length $= 16\text{k}$												
HqeKV	**100.00**	99.40	99.00	99.60	99.80	**86.40**	96.70	95.10	96.20	26.80	95.00	90.40
RDKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	79.70	**100.00**	**100.00**	**100.00**	**76.92**	**95.53**	**95.65**
Sequence length $= 32\text{k}$												
HqeKV	99.80	99.20	88.80	96.60	98.60	**85.20**	93.80	91.00	95.30	15.30	**87.20**	86.40
RDKV	**100.00**	**100.00**	**100.00**	**100.00**	**100.00**	57.66	**100.00**	**100.00**	**100.00**	**34.82**	78.60	**88.28**
Sequence length $= 64\text{k}$												
HqeKV	99.60	96.00	72.40	97.20	94.80	**28.00**	86.40	81.00	86.20	0.10	**84.10**	75.10
RDKV	**100.00**	**100.00**	**98.61**	**100.00**	**96.95**	19.06	**100.00**	**95.86**	**94.88**	**3.94**	71.47	**80.07**

Table 11: Performance on the 10 InfiniteBench tasks for LLaMA-3.1-8B-Instruct at $B_{\text{total}} = 1024L$. The best result among compression methods is in **bold**; the second-best is underlined. FullKV is reported as a reference upper bound.

Method	Retr.Pass	Retr.Num	Retr.KV	En.Dia	En.Sum	En.MC	En.QA	Zh.QA	Math.Find	Debug	Avg.
FullKV	100.0	99.32	56.20	18.00	27.52	68.56	14.63	13.28	34.00	22.34	45.38
SnapKV	**100.00**	96.61	1.40	8.50	23.35	68.12	12.58	12.44	**34.00**	22.08	37.91
AdaKV	**100.00**	94.41	1.80	10.50	23.66	68.12	12.94	12.30	**34.00**	**22.84**	38.06
ThinK	**100.00**	87.97	1.80	7.00	23.03	68.12	11.21	12.35	33.71	22.08	36.73
Snap+Zip	**100.00**	91.86	2.20	11.50	22.16	67.69	13.43	12.40	**34.00**	22.34	37.76
RDKV	**100.00**	**96.95**	**7.20**	**13.00**	**25.13**	**68.56**	**14.49**	**12.65**	**34.00**	22.59	**39.46**

Table 12: Prefill overhead breakdown for RDKV on LLaMA-3.1-8B-Instruct at 128K context length (A100 64 GB). Percentages are relative to the FullKV prefill baseline.

Component	Time (ms)	% of FullKV Prefill
FullKV prefill (FA2)	28 843	— (baseline)
Forward path saving	-593	-2.1%
w_t computation	$+308$	$+1.1\%$
w_c computation	$+550$	$+1.9\%$
MCKP bisection	$+609$	$+2.1\%$
TriZone packing	$+863$	$+3.0\%$
Net RDKV overhead	$+1{,}740$	$+6.0\%$
RDKV TTFT	30 583	106.0%

Table 13: End-to-end generation latency (prefill + decoding) for a 128K-token input on LLaMA-3.1-8B-Instruct (A100 64 GB).

Generated tokens	FullKV	RDKV	Speedup
256	49.75 s	35.17 s	$1.41\times$
1,024	112.73 s	49.00 s	$2.30\times$

a mild reduction of K budget ($r_K = 0.4$) is largely absorbed because the per-channel distortion weight w_c (Prop. 3.2) concentrates on a small subset of outlier channels; most channels tolerate lower bit-widths with negligible error. The symmetric optimum $r_K = 0.5$ balances these two effects, justifying the equal split adopted throughout the paper.

Table 14: K/V budget split ablation: per-task LongBench scores for LLaMA-3.1-8B-Instruct at $B_{\text{total}} \in \{128L, 256L, 512L\}$. r_K denotes the fraction of B_{head} allocated to K. The best result in each row is in **bold**; the second-best is underlined.

r_K	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 128L$																	
r_K=0.4	**29.65**	**41.93**	**56.21**	56.51	48.05	**30.93**	**25.58**	**24.98**	**24.37**	63.50	**92.14**	41.11	8.58	99.50	61.09	53.55	47.36
r_K=0.5	29.45	41.34	55.55	**57.39**	**49.38**	30.89	25.55	24.59	24.29	**65.00**	91.92	**41.80**	**8.75**	**100.00**	**62.68**	**55.41**	**47.75**
r_K=0.6	29.22	38.60	55.40	56.47	48.69	30.74	24.69	24.12	23.44	63.50	91.94	41.61	**8.75**	**100.00**	62.41	54.92	47.15
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 256L$																	
r_K=0.4	29.22	43.75	56.13	56.89	49.16	30.85	**28.26**	24.74	**25.71**	67.00	91.61	41.19	8.52	99.50	63.11	53.89	48.10
r_K=0.5	**29.63**	**44.85**	**56.33**	**57.85**	**49.64**	**31.73**	28.01	24.66	25.66	**70.00**	**91.87**	41.79	**8.83**	99.50	**63.27**	56.01	**48.73**
r_K=0.6	29.15	43.62	55.49	57.05	49.21	31.28	27.09	**24.89**	25.28	68.50	91.20	**42.77**	8.68	**100.00**	62.54	**56.51**	48.33
LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 512L$																	
r_K=0.4	29.85	45.35	**56.77**	57.04	**48.99**	31.11	**30.72**	**25.24**	26.37	70.50	**91.91**	42.26	8.75	**100.00**	**63.41**	55.73	49.00
r_K=0.5	**30.24**	**45.67**	55.58	57.06	48.95	31.67	30.68	25.06	**26.46**	**72.00**	91.83	**43.15**	8.83	**100.00**	63.36	57.02	**49.22**
r_K=0.6	29.92	44.96	55.61	**57.31**	48.78	**31.97**	29.78	24.93	26.29	71.00	91.58	43.02	**8.93**	99.50	63.38	**57.17**	49.01

I Additive Objective Validation

We evaluate whether the additive objective in Eq. (6) remains predictive when many cache units are compressed simultaneously. For six held-out 8 k prefixes, we generate 41 matched-budget allocations at each of two budgets: the RDKV allocation, 20 local bit-swap perturbations, and 20 per-head bit-assignment permutations. All candidates preserve the same total bit cost.

For each allocation, we measure vocabulary KL divergence over eight teacher-forced decode positions and compare the average end-to-end KL with the additive objective. We report Spearman correlation, repeated 50/50 cross-validated isotonic R^2 and NRMSE, and normalized KL regret

$$R_{\text{norm}} = \frac{D_{\text{E2E}}(\text{RDKV}) - D_{\min}}{D_{\max} - D_{\min}},$$

where $D_{\min}$ and $D_{\max}$ are computed over the 41 matched-budget allocations.

Table 15 summarizes the agreement between the additive objective and end-to-end vocabulary KL.

Table 15: Agreement between the additive objective and end-to-end vocabulary KL under simultaneous compression.

Budget	Spearman ρ	CV R^2	CV NRMSE	R_{norm}
$128L$	0.762	0.967	0.082	1.09×10^{-4}
$512L$	0.769	0.917	0.123	1.64×10^{-5}

As shown in Tab. 15, the additive objective shows consistent agreement with end-to-end KL, while the RDKV allocation has near-zero normalized regret, supporting its use as a surrogate objective for joint compression.

J Sensitivity to Distortion Calibration

To assess sensitivity to the distortion table $\varepsilon(b)$, we replace the original table used in the main experiments with a WikiText-calibrated table, a Mistral-calibrated table transferred to LLaMA-3.1-8B-Instruct, and a calibration-free Bennett approximation. All other settings are kept fixed. Table 16 reports the per-task RULER results under the four distortion-table sources.

As shown in Tab. 16, the alternative empirical calibrations remain close to the original: WikiText and Mistral differ by at most 0.53 points at 16 k and 0.83 points at 32 k. The calibration-free Bennett approximation incurs a larger but still modest degradation of 0.92 and 1.59 points, respectively. These results show that RDKV is not strongly dependent on the particular source used to estimate $\varepsilon(b)$.

Table 16: Per-task RULER performance under different distortion-table sources for LLaMA-3.1-8B-Instruct at $B_{\text{total}} = 1024L$.

Source	S-NIAH-1	S-NIAH-2	S-NIAH-3	MK-NIAH-1	MK-NIAH-2	MK-NIAH-3	MQ-NIAH	MV-NIAH	VT	CWE	FWE	Avg.
Sequence length = 16k												
Original	100.00	100.00	100.00	100.00	100.00	79.70	100.00	100.00	100.00	76.92	95.53	95.65
WikiText	100.00	99.40	100.00	99.00	99.60	84.80	99.35	98.35	99.12	74.58	93.47	95.24
Mistral	100.00	99.60	99.80	99.00	99.80	87.20	99.30	98.45	99.08	74.02	93.60	95.44
Bennett	100.00	100.00	98.40	99.20	98.80	88.20	98.95	96.80	98.76	69.70	93.20	94.73
Sequence length = 32k												
Original	100.00	100.00	100.00	100.00	100.00	57.66	100.00	100.00	100.00	34.82	78.60	88.28
WikiText	100.00	97.80	99.60	99.20	98.80	71.40	98.95	96.25	97.96	29.96	72.00	87.45
Mistral	100.00	98.00	99.80	99.00	98.60	73.60	99.00	96.55	97.80	30.94	72.00	87.75
Bennett	100.00	99.80	97.00	100.00	98.60	63.60	98.40	95.95	97.92	28.70	73.67	86.69

Table 17: Runtime and memory overhead of dynamic re-budgeting at 128k context with 1,024 generated tokens.

Method	Decode latency (ms/token)	Peak memory (GB)
Static RDKV	18.70	30.36
Dynamic RDKV	24.73	30.32

Table 18: Per-task LongBench [5] performance of static and dynamic RDKV for LLaMA-3.1-8B-Instruct at $B_{\text{total}} = 1024L$. Dynamic RDKV re-runs the allocation every 256 generated tokens.

Method	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot Learning			Synthetic		Code		Avg.
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovRep	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
Static	30.41	44.54	55.82	57.14	49.25	31.77	33.14	25.19	26.92	72.00	91.84	43.90	9.08	100.00	63.56	57.01	49.47
Dynamic	30.37	44.39	55.82	57.14	49.25	31.77	33.06	25.27	26.96	72.00	91.84	44.02	9.08	100.00	63.54	57.01	49.47

K Dynamic Re-budgeting

By default, RDKV keeps the prefix allocation fixed during decoding. We additionally evaluate a dynamic extension that re-runs the allocation every 256 generated tokens and packs each completed decode block into the compressed cache. This allows newly generated KV states to be periodically incorporated into the compressed representation.

Table 17 reports the runtime and memory impact at 128k context with 1,024 generated tokens. Dynamic re-budgeting reduces peak memory only marginally, from 30.36 to 30.32 GB, while increasing decode latency from 18.70 to 24.73 ms/token.

We further evaluate whether dynamic re-budgeting changes task performance on all 16 LongBench datasets. Dynamic RDKV re-runs the allocation every 256 generated tokens. As shown in Table 18, the dynamic variant preserves essentially the same performance as static RDKV: both obtain an average score of 49.47.

The per-task differences in Table 18 are also negligible: the mean change is -0.003 points, with changes ranging from -0.15 to $+0.12$. Together with the systems results in Table 17, this indicates that dynamic re-budgeting provides little practical benefit in this setting: it preserves accuracy and slightly reduces peak memory, but introduces substantial additional decoding latency. We therefore use the static allocation as the default RDKV configuration.

L Visualization of Bit Allocation

To illustrate how the RDKV allocator distributes bit-widths across tokens and channels, we visualize the per-unit distortion weight w_t (V cache, Prop. 3.1) and w_c (K cache, Prop. 3.2) together with

the resulting bit-width assignment on a representative LongBench sample (LLaMA-3.1-8B-Instruct, $B_{\text{total}} = 128L$). Each dot is coloured by its assigned bit-width: • 8-bit, • 4-bit, • 2-bit, • 0-bit (evicted). No unit receives 16 bits at this budget. We show layers 15 (middle) and 31 (final) with two KV heads each.

Token-level V allocation (Fig. 7–Fig. 13). The score distribution spans roughly six orders of magnitude. The sink token (position 0) and recent tokens near the sequence tail receive the highest scores and correspondingly 8-bit retention, with 4-bit tokens bridging the transition—the former carry accumulated attention mass, and the latter fall within the observation window. Most mid-sequence tokens fall below the zero-rate threshold and are evicted; those just above it are kept at 2 bits, and occasional attention spikes are kept at 4 bits. Head 0 and head 1 share the global profile but differ in which mid-sequence positions spike, reflecting head-specific attention patterns.

Channel-level K allocation (Fig. 8–Fig. 14). The 128 channels of each head display a heavy-tailed score distribution. A handful of outlier channels, with scores 2 to 3 orders of magnitude above the median, receive the highest precision (8-bit at this budget). These correspond to the persistent outlier channels documented in prior work on KV cache quantization [37, 46], where a few dimensions exhibit large query and key norms and therefore dominate the score $\|Q_{:,c}\|_2^2 \|K_{:,c}\|_2^2$. The remaining channels are compressed to 2–4 bits, and in most heads a few of the lowest-scoring channels are removed entirely ($b = 0$). The allocator thus concentrates precision on the outlier channels without a hand-set outlier threshold, avoiding the uniform-precision penalty that affects fixed-bit-width quantization methods.

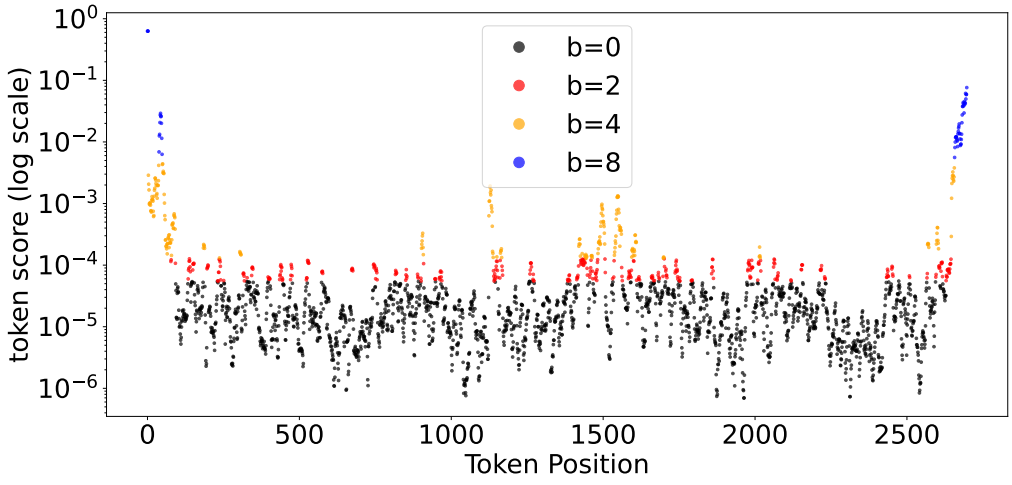


Figure 7: Per-token V-cache bit allocation: layer 15, head 0.

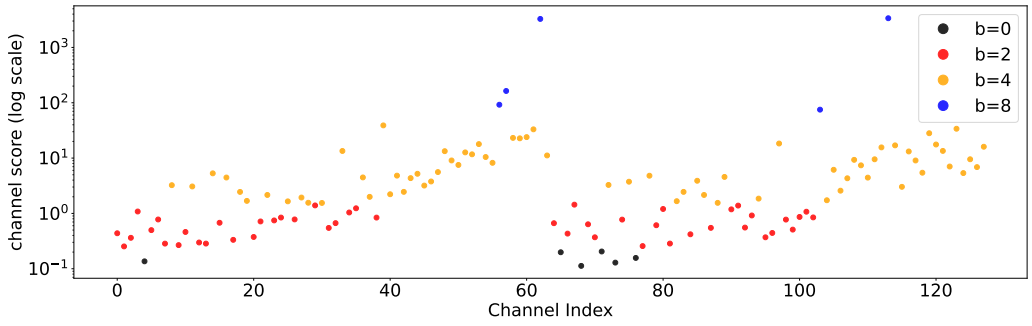


Figure 8: Per-channel K-cache bit allocation: layer 15, head 0.

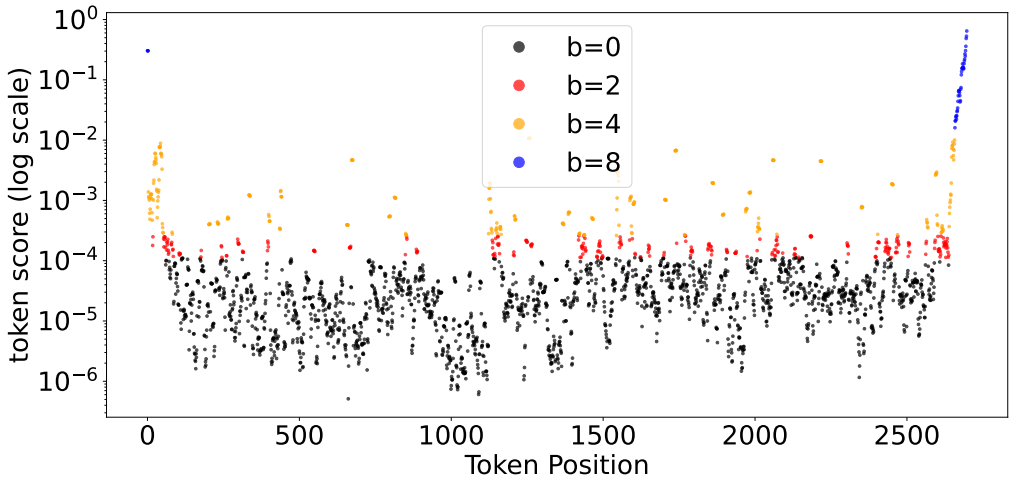


Figure 9: Per-token V-cache bit allocation: layer 15, head 1.

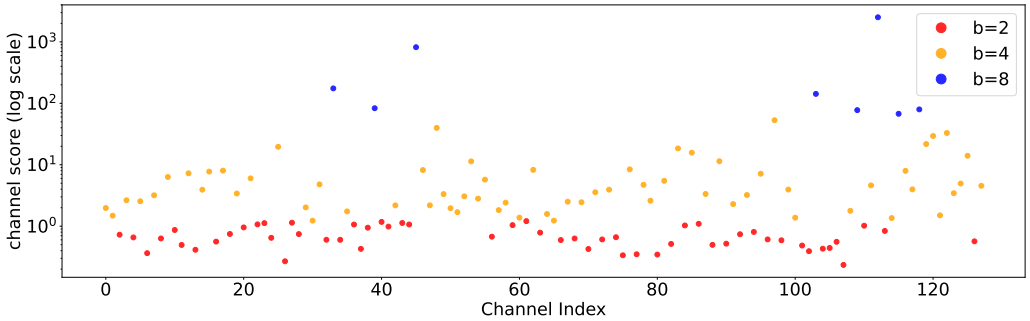


Figure 10: Per-channel K-cache bit allocation: layer 15, head 1.

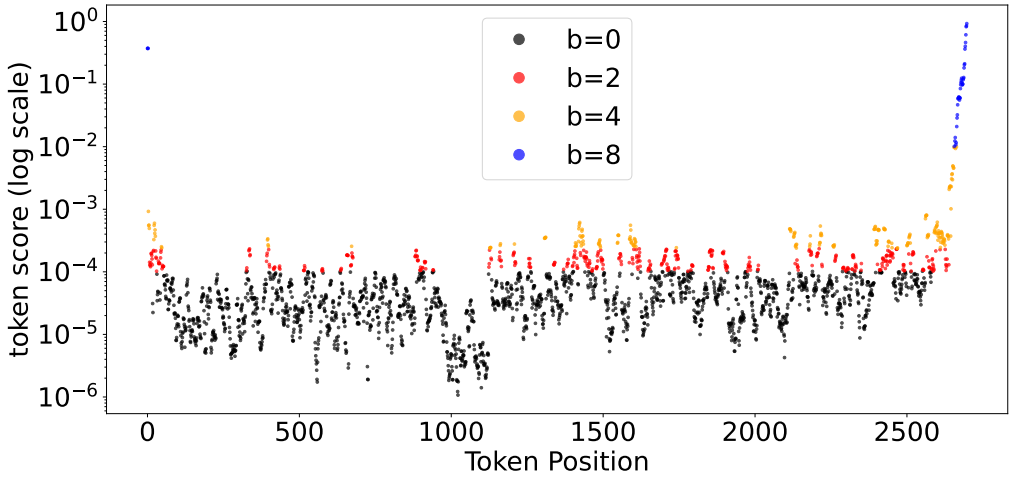


Figure 11: Per-token V-cache bit allocation: layer 31, head 0.

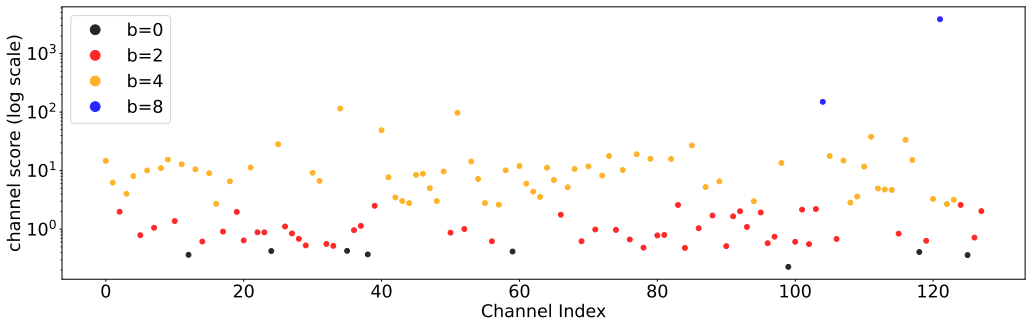


Figure 12: Per-channel K-cache bit allocation: layer 31, head 0.

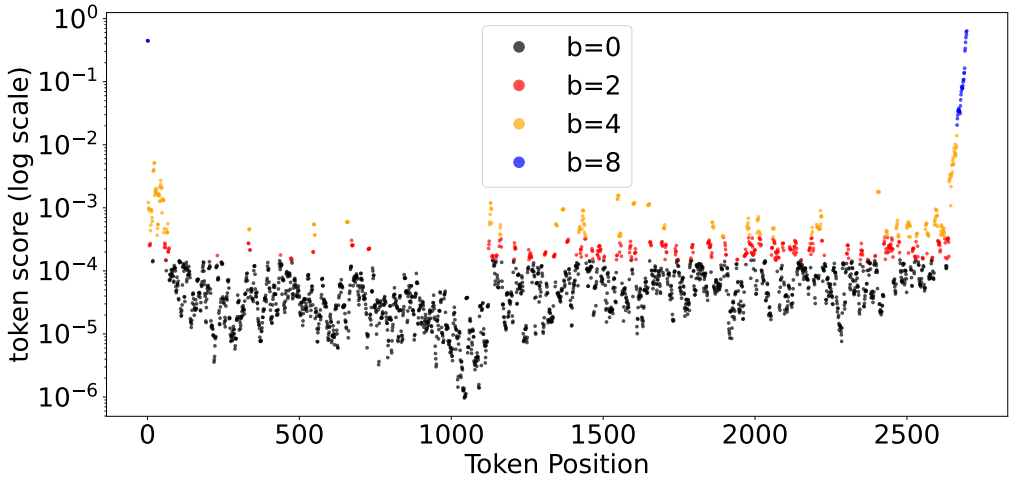


Figure 13: Per-token V-cache bit allocation: layer 31, head 1.

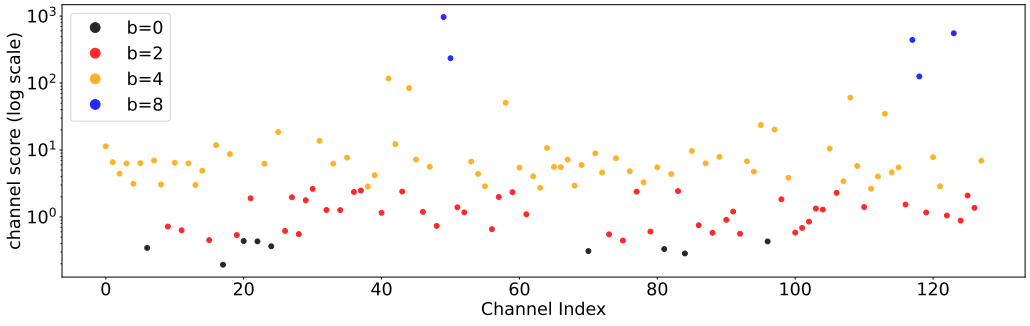


Figure 14: Per-channel K-cache bit allocation: layer 31, head 1.

M CALIBRATED DISTORTION TABLES

Table 19 reports the calibrated per-coordinate distortion $\varepsilon^K(b)$ (K cache, per-channel NMSE) and $\varepsilon^V(b)$ (V cache, per-token NMSE) for each evaluated model and bit-width $b \in \{2, 4, 8\}$. Each entry is averaged across all layers, heads, and 32 calibration sequences (4 k tokens, LongBench prefill prefixes; see Sec. B for calibration details). These layer-averaged values are the distortion tables consumed by the MCKP solver (Prop. A.2). All models share $\varepsilon(0) = 1$ (eviction) and $\varepsilon(16) = 0$ (full precision).

V cache distortion is notably consistent across architectures at every bit-width, whereas K cache distortion shows greater cross-model variance—chiefly because Qwen2.5-72B's embedding-adjacent layers exhibit elevated per-channel 8-bit quantization error.

Table 19: Calibrated normalized distortion $\varepsilon(b)$ per model. K cache: per-channel NMSE; V cache: per-token NMSE.

	$\varepsilon^K(b)$ (per-channel)			$\varepsilon^V(b)$ (per-token)		
Model	b=2	b=4	b=8	b=2	b=4	b=8
LLaMA-3.1-8B	0.149	0.0062	2.2×10^{-5}	0.313	0.0140	4.9×10^{-5}
LLaMA-2-13B	0.288	0.0124	5.5×10^{-5}	0.272	0.0122	4.4×10^{-5}
Mistral-7B	0.280	0.0116	6.9×10^{-5}	0.296	0.0130	4.5×10^{-5}
Qwen2.5-72B	0.296	0.0164	4.4×10^{-3}	0.281	0.0126	4.4×10^{-5}
Qwen3-4B	0.347	0.0147	1.5×10^{-4}	0.313	0.0139	4.8×10^{-5}

N LIMITATIONS AND FUTURE WORK

Limitations. RDKV targets the prefill-dominated regime (long prompt, short generation), and its default configuration fixes the allocation after prefill, as do SnapKV, AdaKV, and PyramidKV. We evaluate a dynamic variant that re-runs the allocation every 256 generated tokens (Sec. K): it matches static RDKV on LongBench (49.47 vs. 49.47) while increasing decode latency at 128k context from 18.70 to 24.73 ms/token. In this regime the generated tokens are a negligible fraction of the cache, so re-budgeting has little to act on. Its value in long-output settings, such as extended reasoning traces where the decode-phase cache becomes comparable to the prompt, remains untested. In addition, the current dynamic variant re-solves the allocation over the full cache at each interval, so its per-interval cost is comparable to the one-time prefill allocation.

Future work. Two directions follow from these limitations. First, re-budgeting can be made incremental: because the water level λ changes little between consecutive intervals, the MCKP can be warm-started from the previous solution and only units near the bit-width thresholds, together with the newly generated block, need to be re-evaluated and repacked. Second, evaluating RDKV on long-output workloads would determine whether decode-phase re-allocation becomes beneficial once generated tokens constitute a substantial share of the cache.