ByteKV: Reusable KV Cache Compression via Bit Allocation and Model Adaptation

Anonymous authors
Paper under double-blind review

Abstract

The key–value (KV) cache accelerates autoregressive decoding in large language models by avoiding redundant context computation, but storing long contexts incurs substantial memory overhead. Reusing a compressed cache across questions requires preserving context information within a tight byte budget before downstream queries are revealed. Here we present ByteKV, a framework combining existing query-blind selection rules with bit allocation and model adaptation. Lowering KV precision enables ByteKV to retain more context positions per byte, while a shared low-rank adapter (LoRA) adapts the model to the sparse, quantized cache through self-distillation from a frozen teacher using a full 16-bit KV cache (all context positions). Across four selection rules evaluated on Llama and Qwen models, 4-bit ByteKV outperforms matched frozen 16-bit baselines on RULER-13, delivering absolute accuracy gains of 17.7 to 58.1 percentage points at a nominal 3% budget relative to full 16-bit KV storage. Integrated with custom fused packed-KV attention and vLLM memory management, ByteKV delivers 1.91× the peak output throughput of full-cache serving with the same adapted weights under concurrent serving. Our code is available at https://anonymous.4open.science/r/ByteKV.

1 Introduction

Large language models (LLMs) process an input context during a *prefill* phase, then generate output tokens sequentially. Prefill stores key–value (KV) vectors at every attention layer for reuse during generation (Kim et al., 2025). The cache grows with context length, and concurrent requests compete for GPU memory. Compressing these caches can make room for more requests.

In multi-turn scenarios—such as a document assistant answering sequential questions about a contract—reusing the initial prefill cache avoids redundant computation (Li et al., 2025). While query-aware compression methods prune the cache for a specific, known question (Li et al., 2024; Cai et al., 2025), they risk discarding context required by subsequent queries. To enable persistent cache reuse, we focus on *query-blind* (query-agnostic) compression: reducing the cache using only the context itself, before downstream questions are revealed (Kim et al., 2025).

Given a fixed memory budget, context selection rules determine which positions are preserved. How many positions fit also depends on numerical precision. Since LLMs can operate with quantized KV vectors (Liu et al., 2024b), lowering per-entry precision makes room for more positions under the same budget (Yang et al., 2024; Zhang et al., 2026; 2025a), retaining context that selection at higher precision would discard. However, quantization perturbs the retained vectors: key errors change attention weights, while value errors change the vectors being combined. The model must therefore use the additional context through a sparse, numerically altered cache. This motivates allocating bytes to retain more context while adapting the model to produce and use the resulting representation.

We propose ByteKV, a framework that combines bit allocation and model adaptation. With the selection and quantization rules fixed, we train a shared low-rank adapter (LoRA) (Hu et al., 2022) via self-distillation to match the next-token distributions of the frozen base model operating with a full 16-bit KV cache. The adapted model learns to produce and use this sparse, quantized cache, with the same adapter active during prefill and autoregressive decoding. To make the storage savings available during serving, we implement custom fused packed-KV attention kernels and vLLM page management tailored to the compressed footprint.

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.

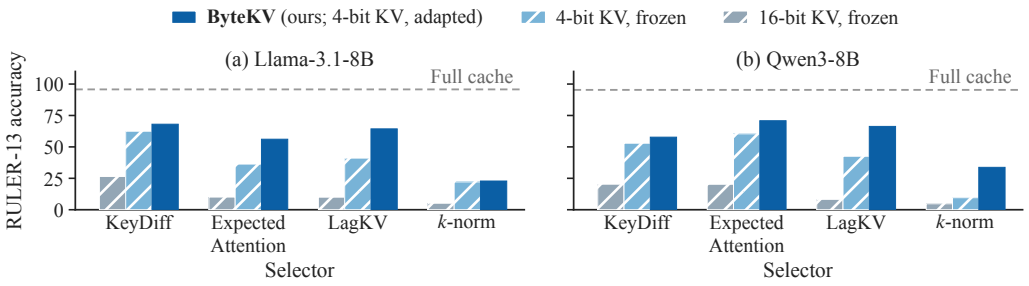


Figure 1: **ByteKV combines bit allocation and model adaptation under an existing selector.** At a 3% byte budget, groups show frozen 16-bit KV, frozen 4-bit KV, and ByteKV with adapted 4-bit KV. Both allocation and adaptation improve full RULER-13 accuracy within each group. KeyDiff's adapted bars average four seeds; others use one. Dashed lines show frozen full-cache accuracy. Full-cache measurements and other budgets: Table 8 and Appendix F.1.

Our contributions are:

- **Reusable cache compression.** We introduce **ByteKV**, combining byte allocation and shared model adaptation under existing query-blind selection rules to build reusable context caches.
- **Accuracy recovery.** At a nominal budget of 3% of full 16-bit KV storage, 4-bit ByteKV improves RULER-13 (Hsieh et al., 2024) accuracy by 17.7 to 58.1 percentage points across four selectors on Llama and Qwen, relative to each selector's matched frozen 16-bit baseline (Figure 1). In a control with fixed retained positions, adaptation still improves accuracy (Section 5.3).
- **Packed serving and throughput.** We implement fused attention over packed KV, with vLLM (Kwon et al., 2023) pages sized for the compressed cache. For Llama-3.1-8B on one A100-40GB with 8192 input and 1024 output tokens, this reduces per-request KV storage by 7.05× and delivers 1.91× the peak output throughput of full-cache serving with the same adapted weights (Section 5.8).

2 Related Work

Selection for reusable contexts. A selector's inputs determine whether its cache can be prepared before future questions arrive. Query-conditioned eviction uses the current question (Li et al., 2024; Cai et al., 2025; Ahn et al., 2026); online policies update retention from observed attention (Zhang et al., 2023; Oren et al., 2024). Query-blind construction can use initial and recent positions (Xiao et al., 2024), context-derived KV statistics (Park et al., 2025; Liang et al., 2025; Devoto et al., 2024), context reconstruction (Kim et al., 2025), or expected attention under a modeled distribution of future queries (Devoto et al., 2025). With a query-blind selection rule fixed, ByteKV studies how to represent and use the retained context under a byte budget.

Precision and coverage. Quantization lowers the storage cost of each entry (Liu et al., 2024b; Hooper et al., 2024), making room for more retained positions. At fixed memory, Quantized Pruning (Zhang et al., 2025a) studies this exchange under fixed pruning rules. MiKV (Yang et al., 2024) retains less important entries at low precision, and RDKV (Zhang et al., 2026) jointly chooses eviction and precision. KVzip (Kim et al., 2025) also combines query-agnostic selection with 4-bit quantization. This trade-off preserves more context, but the model receives it through perturbed keys and values.

Adapting to compressed representations. Model training can address the changes introduced by compression. KV-CAT (Gelberg et al., 2026) trains models to produce compressible KV. For quantized caches that retain all positions, KVLinC (Saxena & Roy, 2025) learns linear attention corrections for quantized keys, while independent concurrent work on low-rank attention adaptation (Abdellatif, 2026) trains query, key, and value projections through a fixed quantizer. RestoreKV (Baek et al., 2026), another independent concurrent study, addresses eviction by self-distilling a LoRA adapter to generate supplementary KV entries alongside a fixed evictor. Its adapter is disabled during question

processing and decoding. ByteKV couples byte allocation with adaptation to a reusable cache that is both sparse and quantized. A shared adapter changes the KV vectors produced during context prefill and their use during autoregressive decoding, without adding supplementary entries. Appendix G and Table 25 expand the comparison.

3 PRELIMINARIES

3.1 KV CACHING AND QUERY-BLIND REUSE

For a context of T tokens, the prefill phase computes a key and a value vector for every position in each layer and KV head. These vectors form the context's KV cache. A *full cache* retains all context positions at the original 16-bit precision, without eviction or quantization. During decoding, the model reads from this cached context to generate subsequent tokens, appending new KV entries as generation proceeds. For one attention head, let q be the current query vector and let K, V collect the available keys and values. Attention reads the cache as (Vaswani et al., 2017)

$$\mathrm{Attn}(q, K, V) = \mathrm{softmax}\left(\frac{qK^\top}{\sqrt{D}}\right)V,$$

where D is the head dimension. Query–key matches determine attention weights, which specify how to combine the cached values for the current token. Quantizing keys perturbs these weights; quantizing values perturbs the vectors being combined. Here q is an attention vector produced while processing a token. At fixed architecture and precision, KV storage grows linearly with the number of cached positions.

Given a model, a context, and a byte budget, our task is to construct a reusable compressed cache before future questions are known. A *query-blind* selection rule uses the context alone (Kim et al., 2025). For each question, the model reads this same context cache, prefills the question, and generates an answer by autoregressive decoding. Question and generated tokens add uncompressed KV entries; the context cache is reused without recomputation or reselection. Query blindness thus restricts cache construction, while attention still uses query vectors to read the cache. Related reuse settings include precomputed prompts (Gim et al., 2024), cache composition (Hu et al., 2025; Yang et al., 2025b; Yao et al., 2025), and cache transfer (Liu et al., 2024a).

3.2 SELECTION, BIT ALLOCATION, AND MODEL ADAPTATION

ByteKV combines three operations on this representation. *Selection* retains KV entries at positions chosen by an existing scoring rule. *Bit allocation* sets their precision and retained count, exchanging fewer high-precision entries for more low-precision entries within the same byte budget. A selected 16-bit cache applies eviction alone; a quantization-only cache retains all positions at lower precision. *Model adaptation* fits the model that produces and reads the compressed cache. We quantize KV entries while keeping model weights unquantized. We call the model without LoRA *frozen* and the model using LoRA *adapted*; either can use a full or compressed cache.

Byte accounting. We denote the nominal KV payload budget for the context by r, expressed as a fraction of its full 16-bit storage. This budget covers stored key and value scalars. Metadata and the uncompressed KV added by questions and generated tokens are accounted for separately. At $r = 3\%$, 16-bit storage retains 3% of positions, while 4-bit storage retains 12% before metadata. We call this exchange of precision for coverage *reallocation*. We choose bytes as the comparison axis, and this choice is not neutral: lower precision buys additional positions. Matching position counts removes that reallocation, and controls with nearly equal retained counts show a much stronger frozen 16-bit baseline (Appendix A.2). Section 5.1 reports realized-byte matching; Appendix A.2 gives the payload formula and metadata costs.

4 METHOD

ByteKV allocates bytes between retained positions and their precision (Section 4.1), then trains a shared LoRA adapter through that cache (Section 4.2; Figure 2). Packed attention and page allocation support inference with compressed KV in GPU memory (Section 4.3).

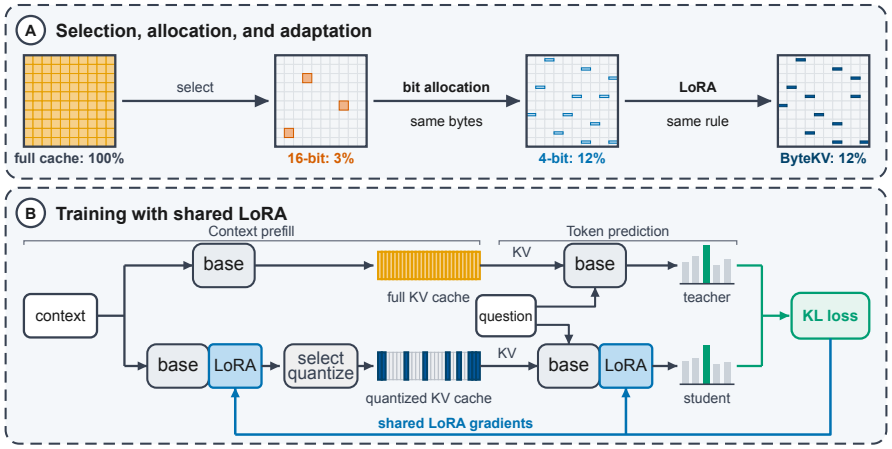


Figure 2: **ByteKV: bit allocation and model adaptation.** (A) Percentages denote retained context positions. The three compressed configurations share a nominal 3% byte budget relative to the full cache. LoRA changes the KV vectors and how the model uses them; with KeyDiff, the retained positions can also change. (B) The teacher uses a full cache; the student shares LoRA across context prefill and token prediction with selected, quantized KV. Both branches keep base weights frozen and unquantized; the loss updates only LoRA. The question and cached KV are separate inputs. Prediction uses teacher-forced answer tokens (Section 4.2). Dark cache endpoints in (B) denote protected positions.

4.1 Compression under a byte budget

Given a query-blind selection rule and a bit-width b per KV scalar, ByteKV sets the retained count to $n \approx (16r/b)T$ per layer and KV head, capped at the context length T, before metadata and rounding. Initial and recent positions are always retained within this count (*protection*); the rule fills the remaining slots. Selected entries preserve sequence order. Question and generated tokens retain their original position indices, starting after the uncompressed context.

Each key and value channel (vector coordinate) is quantized across groups of retained positions, using a separate scale and offset (zero point) per group. Training dequantizes these entries before attention. A straight-through estimator (Bengio et al., 2013) treats quantization as the identity during backpropagation. Gradients flow through the selected KV vectors without differentiating the discrete selection indices. Training and evaluation use the same compression rules (Appendix A.1).

The selection rule and bit-width are fixed configuration choices. Training optimizes only the LoRA parameters. KeyDiff (Park et al., 2025) scores the adapted model's keys, so LoRA can change the retained positions, or *keep set*, under the same scoring rule. Appendix C.1 measures these changes.

4.2 Self-distillation from the full cache

Each training example provides a context, a question, and a reference answer. The *teacher* prefills the context with the frozen base model and retains its full cache. The *student* uses the same frozen base weights with LoRA and compresses its own context cache using the operator above. LoRA adapts every layer's query (Q), key (K), value (V), and output (O) projection weights. Each projection has its own LoRA parameters, shared across context prefill and subsequent token prediction.[1]

Using their respective caches, both models predict each answer token from the same question and preceding tokens from the reference answer (*teacher forcing*). At answer position t, let p_t^{teacher} and $p_{t,\theta}^{\text{student}}$ denote their next-token distributions over the vocabulary, with θ collecting all student LoRA parameters.

[1]The Qwen3 recipe also uses an optional full-cache consistency term that matches the student's uncompressed output to the same teacher (Appendix A.1).

To preserve the teacher's predictions through the compressed cache, we minimize Kullback–Leibler (KL) divergence over positions $\mathcal{A}$ in the reference answer:

$$\mathcal{L}_{\text{comp}} = \frac{1}{|\mathcal{A}|} \sum_{t \in \mathcal{A}} \mathrm{KL}\left(p_t^{\text{teacher}} \,\|\, p_{t,\theta}^{\text{student}}\right). \tag{1}$$

The targets are teacher distributions, without a gold-token cross-entropy term. The trained adapter is reused across new contexts and questions.

4.3 Packed-cache inference

Training reconstructs quantized entries before attention; serving must keep the cache compressed in GPU memory. Our Triton (Tillet et al., 2019) kernel follows FlashAttention's tiled computation with online softmax (Dao et al., 2022; Milakov & Gimelshein, 2018). It reads selected context KV (*prefix*) directly from packed 4-bit pages, dequantizes entries in registers, and incorporates the 16-bit KV of question and generated tokens (*tail*) in the same attention computation. This fuses unpacking, dequantization, and attention across both formats, avoiding a full 16-bit prefix buffer and separate prefix and tail attention launches.

The vLLM backend sizes pages for this layout, so compressed storage translates into capacity for more resident requests. It accounts for both the packed prefix and uncompressed tail. Merging LoRA into the base weights removes separate adapter allocation and execution. Appendix E.1 details the layout; Section 5.8 measures the resulting serving capacity and throughput.

5 Experiments

In this section, we first describe the evaluation setup (Section 5.1), then address three questions: **Q1:** Does combining bit allocation with model adaptation improve accuracy over frozen 16-bit selection under the same rule and byte budget? (Section 5.2). **Q2:** How do the cache representation and the adapter's training data affect accuracy gains from adaptation? (Sections 5.3–5.7). **Q3:** Can the compressed cache reduce per-request KV storage and improve peak output throughput in concurrent serving? (Section 5.8).

5.1 Experimental setup

Models and data. We evaluate Llama-3.1-8B (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025a) on all 13 RULER tasks (Hsieh et al., 2024), with 500 examples per task. Macro accuracy averages task scores; differences are in percentage points. The default adapters use 701 synthetic retrieval examples with 4K-token contexts and random identifiers as answers, and 616 independently generated development examples. K denotes 1024 tokens. The main configuration uses KeyDiff with 4-bit KV, denoted ByteKV-KeyDiff. Its results average four training runs (random seeds 0–3); the Mistral-7B (Jiang et al., 2023) extension uses two seeds. Projection and length studies use a four-task needle-retrieval subset (Appendices A.2 and D.1).

Comparison configurations. We compare frozen 16-bit KV, frozen 4-bit KV, and adapted 4-bit KV (ByteKV) using the same scoring rule and nominal byte budget. All three protect the same initial and recent context positions. The other retained positions can differ with precision and adaptation. We call this 16-bit configuration the *matched 16-bit control*; baselines labeled *original protection* retain their default policies. The frozen pair measures reallocation, and the 4-bit pair measures adaptation. KeyDiff provides the main comparison and bit-width sweep; Section 5.3 extends the comparison to other selection rules. All compressors use the query-blind setting. In the KVPress evaluation toolkit, SnapKV disables question conditioning and is labeled *prefix-only* (Appendix G).

We train the default adapter at a 3% byte budget and evaluate the same adapter at 3%, 5%, and 10%. The two larger budgets test its performance when more context positions are retained than during training.

Checkpoint selection. Among checkpoints from three training epochs, we select the one with highest compressed-cache development accuracy whose absolute change in full-cache accuracy from

Table 1: **ByteKV-KeyDiff and its matched frozen controls.** Full RULER-13 macro, 500 examples per task, at nominal byte budgets.

Model	Budget	Full cache frozen	16-bit KV frozen	4-bit KV frozen	ByteKV 4-bit, adapted
Llama-3.1-8B	10%	95.75	61.17	80.97	**82.64** $\pm$ 2.47
	5%		44.22	68.63	**73.38** $\pm$ 1.66
	3%		26.47	62.45	**68.06** $\pm$ 1.29
Qwen3-8B	10%	95.36	48.21	75.95	**78.58** $\pm$ 0.50
	5%		30.68	63.70	**65.94** $\pm$ 0.47
	3%		20.30	52.91	**57.75** $\pm$ 0.55

The 16-bit control matches the scoring and protection rules (Section 5.1). Full cache uses the frozen base model. ByteKV-KeyDiff is the four-seed mean $\pm$ population standard deviation (Table 4). Frozen columns are deterministic. The separately trained 16-bit diagnostic appears in Appendix B.4.

the frozen model stays within a predeclared tolerance (Table 3). If none qualifies, no checkpoint is selected. Both full and compressed test evaluations use that checkpoint. The tolerance governs development-set eligibility; test-set full-cache changes provide a severity scale, not a second eligibility test (Appendices A.2 and F).

Nominal and realized bytes. For full groups, scales and zero points raise storage from 4 to 4.25 effective bits per scalar. Rounding retained counts and storing indices bring realized bytes to between 106.6% and 111.3% of the nominally matched frozen 16-bit control's. Exact-byte controls use measured context lengths and charge index metadata to both configurations (Appendix A.2).

Accuracy tests use floating-point reconstructions of quantized KV for standard attention, retaining quantization error. Byte budgets describe the compressed representation. We measure actual storage and throughput with packed serving (Section 5.8); Table 23 compares the two implementations' accuracy.

5.2 Allocation and adaptation under fixed selection

Among eight frozen KVPress baselines (Devoto et al., 2025), the highest scores at a 3% 16-bit payload are 34.1 on Llama-3.1-8B and 19.4 on Qwen3-8B (Table 9). With original protection, each nonrandom baseline in this set loses at least 44% of its score as the byte budget tightens from 10% to 3% on both architectures (Appendix B.2).

Table 1 addresses Q1 with three KeyDiff configurations at each nominal byte budget: frozen 16-bit KV, frozen 4-bit KV, and adapted 4-bit KV (ByteKV-KeyDiff). Reallocation expands coverage and adds 19.8 to 36.0 macro points; adaptation adds 1.7 to 5.6 with the resulting representation. Both mean gains are positive in all six model–budget cells. At 3%, reallocation raises Llama accuracy from 26.47 to 62.45, and adaptation raises it to 68.06. Qwen3's corresponding scores are 20.30, 52.91, and 57.75. With exact byte matching, reallocation adds 18.4 to 34.5 points and adaptation adds 2.1 to 7.0 (Appendix A.2). Task-level results and the ten-task retrieval subset appear in Appendices C.2 and C.3.

These results reuse an adapter trained at 3% at the two milder budgets. Its mean gain exceeds the four-seed standard deviation in five of six cells. For Llama at 10%, the gain is 1.67 with standard deviation 2.47, and two of four seeds score below frozen 4-bit KV (Table 4). Thus positive mean recovery at that budget is not consistent across seeds. Mistral-7B adds a third architecture: both training seeds improve on frozen 4-bit KV at all three budgets, with mean gains of 6.13 to 7.97 points. At 3%, accuracy rises from 56.97 to 64.94 (Appendix D.1).

5.3 Adaptation across selection rules

We test whether the gains extend beyond KeyDiff by repeating the comparisons under other selection rules (Table 2). Each comparison fixes the scoring function, protection, and sequence ordering; we report within-selector gains without ranking selectors. Reallocation improves all 18 added cells with matched 16-bit controls for Expected Attention, LagKV, and k-norm across two models and three budgets, alongside KeyDiff's six main cells. Against 16-bit controls with baseline protection policies, it improves 27 of 30 cells across five selectors and all 10 cells at 3%; KVzip accounts for the three

Table 2: **Within-selector comparisons on full RULER-13.** Each budget reports absolute accuracy for frozen 16-bit KV, frozen 4-bit KV, and ByteKV with adapted 4-bit KV.

(a) Llama-3.1-8B

	10% budget			5% budget			3% budget		
Fixed selector	16-bit	4-bit	ByteKV	16-bit	4-bit	ByteKV	16-bit	4-bit	ByteKV
KeyDiff	61.2	81.0	82.6	44.2	68.6	73.4	26.5	62.5	68.1
Expected Attention	30.9	77.7	78.8	15.2	56.1	65.2	10.1	36.3	56.1
LagKV	37.3	78.9	80.9‡	15.3	56.8	72.6‡	10.0	41.1	64.3‡
k-norm	20.0	43.4	43.2†	7.5	28.2	31.6†	5.2	22.1	22.9†

(b) Qwen3-8B

	10% budget			5% budget			3% budget		
Fixed selector	16-bit	4-bit	ByteKV	16-bit	4-bit	ByteKV	16-bit	4-bit	ByteKV
KeyDiff	48.2	76.0	78.6	30.7	63.7	65.9	20.3	52.9	57.8
Expected Attention	58.8	83.5	92.4	39.7	66.7	79.5	20.2	60.4	70.8
LagKV	34.9	75.4	86.5	12.8	59.5	77.2	8.2	42.5	66.3
k-norm	8.9	52.1	65.9	5.8	18.1	46.7	4.7	9.6	33.7

All pairs fix scoring and protection. KeyDiff's adapted results average four seeds, other adapted results use one. Table 8 gives two-decimal values and the KVzip and positional configurations, which lack matched 16-bit controls. For Llama's adapted cells: †full-cache loss of 26.5 points on the four-needle test slice. ‡Unchanged full-cache accuracy on that slice, but a loss on full RULER-13 (Appendix F.1).

exceptions at milder budgets (Table 8). Adaptation improves 23 of 24 non-KeyDiff 4-bit pairs. The exception is k-norm on Llama at the 10% budget, where adaptation reduces accuracy from 43.39 to 43.18.

A fixed scoring function need not retain the same positions after adaptation. KeyDiff, k-norm, LagKV, and Expected Attention score the KV vectors produced by the current model, so LoRA can change both these vectors and their ranking. Our KVzip control instead shares frozen-model scores between the two 4-bit configurations, while positional streaming selects by location. Their positions stay fixed at a given budget, but LoRA remains active during prefill and decoding, changing the retained KV vectors and their subsequent use.

KVzip improves accuracy even with the retained positions fixed (Appendix B.3), establishing that recovery does not require changing which positions survive. Fixed positions alone do not ensure recovery: at 3%, positional streaming discards answer-bearing positions on almost every four-needle example; Qwen3's frozen and adapted 4-bit configurations both score 12.95. On full RULER-13, adaptation raises accuracy from 25.44 to 30.55 with the same checkpoint. Its benefit therefore depends on what the retained representation makes usable for each task. Three Llama KVzip runs span 7.21 points at 10% (Appendix B.2); most non-KeyDiff configurations use one seed.

5.4 What adaptation changes

With recovery possible at fixed positions, we examine how projection updates change KV representations and, under KeyDiff, selection in later layers.

Effects across layers. LoRA changes the hidden representations passed from one attention layer to the next. Under KeyDiff, V-only and Q/O-only adaptation leave first-layer selection unchanged, yet both change the positions selected in later layers (Appendix C.1). Even an unchanged key projection can therefore produce different keys when its input changes. Placement specifies which weights are updated directly, rather than separating cache construction from cache use.

Projection placement. We train each projection configuration separately with random seed 0 and evaluate it on the four-needle subset (Table 11). Adapting only Q and O retains 91% of the default Q/K/V/O adapter's gain on both architectures. Under grouped-query attention (GQA) (Ainslie et al., 2023), multiple query heads share each KV head. In these models, the Q/O adapter has 3.2× as many trainable parameters as either the K-only or V-only adapter. The V-only and K-only adapters have identical parameter counts and layer coverage because the key and value projections have the same shape. V-only adaptation recovers 100% of the default adapter's gain on Llama and 61% on Qwen3, compared with 50% and 11% for K-only adaptation. V is thus an effective adaptation target under

this rule, even though KeyDiff scores keys. Over four seeds, Llama's V-only adapter recovers 84% of the default gain on average, ranging from 58% to 100%.

Answer positions and usable context. Answer-token survival is the retained fraction of context tokens that spell the reference answer, excluding protected positions and averaging over layers and KV heads. On Llama's harder multi-key retrieval task, adaptation raises this fraction by 4.3 percentage points at 3%, while accuracy remains near zero. An easy single-needle task retains accuracy despite slightly lower survival (Table 12). Together with the fixed-position KVzip control, these observations motivate examining the content and use of retained vectors, beyond counting literal answer positions. Prefill constructs contextual representations before eviction; a retained vector need not represent its own token in isolation.

5.5 Dependence on bit-width

The preceding comparisons fix 4-bit precision. We now vary precision, which changes both the retained count and numerical error at fixed bytes, and train separate adapters at 16, 8, 4, and 2 bits. The adapted sweep shows model-dependent precision preferences (Figure 4); paired frozen controls test what adaptation adds at a given precision and byte budget.

At 2 bits and 3%, accuracy rises from 12.63 to 49.55 on Llama and from 40.50 to 61.95 on Qwen3, with improvements on 12 of 13 tasks for each model. Macro accuracy improves at all three budgets on both models. At 8 bits, Qwen3 again improves throughout, while Llama improves at 3% and declines at the milder budgets (Table 10). Each pair uses the same standalone evaluator and one training seed. The 16-bit diagnostic also trains its own adapter, testing adaptation when fewer positions survive without quantization; Llama falls below its matched frozen control at 10% (Appendix B.4). Adaptation improves accuracy on both models with 2-bit KV, while its effect at higher precision varies with the model and byte budget.

5.6 Training distribution and task transfer

Precision changes the cache representation; training data specify which predictions the adapter is trained to preserve. We evaluate whether its accuracy gains transfer to new tasks on LongBench (Bai et al., 2024) and SCBench (Li et al., 2025), using the same adapters without downstream fine-tuning. On LongBench at 3%, adaptation changes the frozen 4-bit score from 41.74 to 31.16 on Llama and 32.46 to 34.38 on Qwen3. Both directions persist at the milder budgets. The complete 4-bit ByteKV-KeyDiff configuration exceeds matched 16-bit KeyDiff on 7 of 12 tasks on Llama and 10 on Qwen3, with macro gains of 0.18 and 7.32 (Appendix D.2). On SCBench's English book question-answering task at 32K, it exceeds the 16-bit control throughout but matches or falls below frozen 4-bit KV.

The default training task retrieves a random identifier from an explicit key. LongBench additionally requires semantic matching, evidence combination, summarization, and code completion. Even its passage-retrieval task matches an abstract to a paragraph rather than looking up a literal key. Multi-format training expands answers to include words, names, and numbers, but retains the same synthetic lookup task and background styles. At 3%, Llama's LongBench score rises from 31.16 to 35.90, still below the frozen 4-bit score of 41.74. Qwen3's score decreases from 34.38 to 33.73. On SCBench, multi-format training improves all six adapted scores and exceeds frozen 4-bit KV in four.

Distillation matches the teacher on the training questions, so it can favor representations useful for those questions without preserving every downstream use of the cache. This offers an explanation for task-dependent transfer. The corpus comparison changes answer formats, training exposure, and development data together; their individual effects remain unresolved (Appendices D.2 and D.3).

5.7 Longer contexts

We extend adapter training and evaluation to 8K and 16K on the disjoint four-needle test slice at 3%, using one seed and the same checkpoint-selection protocol. Each adapter is trained at its evaluation length. Llama's gains over frozen 4-bit KV decrease with length: 14.9 points at 4K, 10.4 at 8K, and 4.1 at 16K. At 8K, adaptation raises accuracy from 51.5 to 61.9 on Llama and from 41.7 to 51.3 on Qwen3. Llama's 16K score rises from 51.0 to 55.1; the corresponding Qwen3 run is absent because

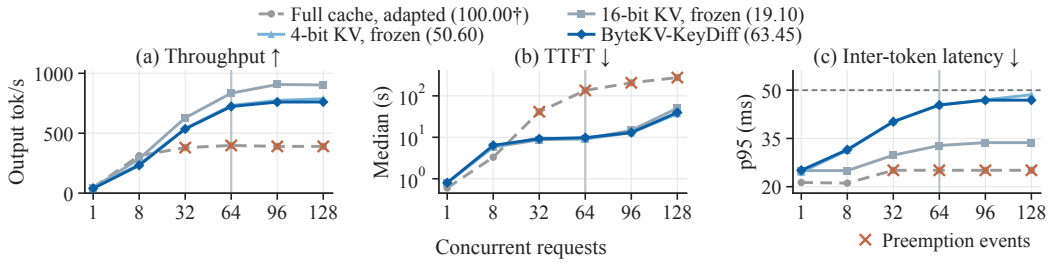


Figure 3: **KeyDiff configurations under concurrent serving.** Llama-3.1-8B, dedicated A100-40GB, 8K context, 1024 generated tokens, nominal 2.68% byte budget for 4-bit KV, CUDA graphs disabled. All compressed configurations use KeyDiff; ByteKV-KeyDiff is the adapted 4-bit configuration. It shares merged weights and a timing sweep with the adapted full-cache reference; the frozen controls use a separate sweep. Parenthesized accuracies (%) come from separate 4K four-needle evaluations: offline for the frozen controls and served for ByteKV-KeyDiff. †The full-cache score is the adapted checkpoint's offline reference before merging; merged full-cache serving accuracy is unmeasured. Crosses indicate preemption and recomputation. The horizontal line marks 50 ms ITL; the vertical line marks 64 requests. Appendix E details the protocol and output-consistency checks.

of resource limits (Appendix A.1). These results show positive recovery after training at longer contexts, with a narrowing Llama margin.

5.8 Serving cost and cache capacity

For Q3, Figure 3 compares four serving configurations: *full cache, adapted*; *16-bit KV, frozen*; *4-bit KV, frozen*; and *ByteKV-KeyDiff* with adapted 4-bit KV. All three compressed configurations use KeyDiff. The full-cache reference and ByteKV-KeyDiff share merged adapted weights; the two frozen controls use the base model. We report output throughput, time to first token (TTFT), and inter-token latency (ITL); p95 denotes the 95th percentile.

Including the generated-token tail, ByteKV-KeyDiff reduces per-request paged KV storage by 7.05× relative to the adapted full-cache reference; the measured pools fit 89 versus 12 sequences. In the concurrency sweep, ByteKV-KeyDiff reaches 759.8 tokens/s at 96 requests, 1.911× the adapted full-cache peak of 397.5 at 64, with 46.9 ms p95 ITL.

At 64 concurrent requests, Table 22 compares all four configurations. The frozen 16-bit control, frozen 4-bit control, and ByteKV-KeyDiff produce 834.3, 731.3, and 723.0 output tokens/s, respectively. Their separate 4K four-needle accuracy references are 19.10, 50.60, and 63.45. The 16-bit control attends to 233 positions versus 878 for both 4-bit configurations: its higher raw throughput accompanies less coverage.

At 64 requests, all three compressed configurations have a median time to first token between 9.1 and 9.8 s, compared with 135.4 s for the full-cache reference. Across the sweep, the full-cache reference incurs 146 preemption events and all three compressed configurations incur none; every request completes. All four stay below the 50 ms p95 inter-token bound, but queueing under sustained concurrent requests makes them exceed the 2 s p95 first-token bound above one request.

6 Conclusion and limitations

ByteKV combines bit allocation and model adaptation for query-blind KV compression. With KeyDiff fixed at a nominal 3% byte budget, reallocation improves RULER-13 accuracy by 36.0 and 32.6 points on Llama and Qwen3; adaptation adds 5.6 and 4.8. Both gains remain positive with exact byte matching. Packed attention and page allocation reduce per-request KV storage, allowing more requests to fit in GPU memory and raising peak output throughput with merged adapted weights.

Our current training uses synthetic retrieval tasks, leaving transfer to broader tasks as an open challenge. Future work can explore richer training data and jointly optimize task and compression-aware objectives during model fine-tuning. The fixed compression configurations also leave room for adaptive choices of KV precision and retention across models, byte budgets, and expected task distributions, while preserving query-blind cache construction. Evaluating these extensions beyond the 7B–8B models studied here would test their accuracy and computational cost at larger scales.

Ethical Considerations

We use public model checkpoints and benchmarks. The main training and development corpora contain random identifiers and templated filler. The training distribution ablation compares single-format and multi-format synthetic corpora (Table 17). The study involves no human subjects or personally identifiable information.

Reproducibility Statement

Section 4 and Appendix A.1 specify the compression operator, objective, and training settings. Section 5.1 and Appendix A.2 describe the data splits, checkpoint selection, evaluation implementations, and byte accounting. Table 4 reports all four main seeds. Full RULER results use 500 examples per task; retrieval-slice ablations, the disjoint 50-example per-family specialist comparison, and complete-split capability evaluations are identified separately.

The repository documents the training and evaluation commands, model and software revisions, corpus generation, and saved predictions. RULER uses a fixed dataset revision; LongBench does not. Each adapter's records identify its training and development corpora.

Use of Large Language Models

LLM-based coding and writing assistants helped write and refactor experiment and analysis code, propose controls and interpretations, and organize prior work from source papers. They also assisted with LaTeX formatting, prose revision, and manuscript organization. A seeded program generated the synthetic corpora. The authors are responsible for the research decisions, verification of AI-assisted work, and the contents of the submission.

References

Seifeldin Abdellatif. Quality recovery for quantized KV caches via low-rank attention adaptation. *arXiv preprint arXiv:2609.04263*, 2026.

Jinwoo Ahn, Ingyu Seong, Akhil Kedia, Junhan Kim, Hyemi Jang, Kangwook Lee, and Yongkweon Jeon. LookaheadKV: Fast and accurate KV cache eviction by glimpsing into the future without generation. In *International Conference on Learning Representations*, 2026.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, 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. doi: 10.18653/v1/2023.emnlp-main.298.

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 *International Conference on Learning Representations*, 2026.

Changwoo Baek, Seungjun Shin, and Kyeongbo Kong. RestoreKV: Recovering full-cache behavior under aggressive query-agnostic KV cache eviction. *arXiv preprint arXiv:2608.01247*, 2026.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Adithya Bhaskar, Alexander Wettig, Tianyu Gao, Yihe Dong, and Danqi Chen. Cache me if you can: How many KVs do you need for effective long-context LMs? *arXiv preprint arXiv:2506.17121*, 2025.

Ngoc Bui, Shubham Sharma, Simran Lamba, Saumitra Mishra, and Rex Ying. Cache what lasts: Token retention for memory-bounded KV cache in LLMs. In *International Conference on Learning Representations*, pp. 120316–120342, 2026.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. In *Second Conference on Language Modeling*, 2025.

Inesh Chakrabarti, David Limpus, Aditi Ghai Rana, Bowen Bao, Spandan Tiwari, Thiago Crepaldi, and Ashish Sirasao. UltraQuant: 4-bit KV caching for context-heavy agents. *arXiv preprint arXiv:2606.20474*, 2026.

Vivek Chari and Benjamin Van Durme. Compactor: Calibrated query-agnostic KV cache compression with approximate leverage scores. *arXiv preprint arXiv:2507.08143*, 2025.

Vivek Chari, Guanghui Qin, and Benjamin Van Durme. KV-Distill: Nearly lossless learnable context compression for LLMs. *arXiv preprint arXiv:2503.10337*, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Ashwin Gerard Colaco and Nada Lahjouji. What to keep, what to forget: A rate–distortion view of memory compaction in LLMs and agents. *arXiv preprint arXiv:2607.08032*, 2026.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective L_2 norm-based strategy for KV cache compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18476–18499. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.1027.

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.

Chuheng Du, Junyi Chen, Hanlin Tang, Kan Liu, Tao Lan, Lin Qu, Chaoyue Niu, Shengzhong Liu, Guihai Chen, and Fan Wu. C^2KV: Compressed and composable KV cache reuse for efficient LLM inference. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 991–1002. ACM, 2026. doi: 10.1145/3770855.3817715.

Shahrzad Esmat, Dhawal Shah, and Ali Jannesari. VarRate: Training-free variable-rate KV cache compression for long-context LLMs. *arXiv preprint arXiv:2607.15498*, 2026.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations*, pp. 42642–42687, 2026.

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. In *Advances in Neural Information Processing Systems*, 2025.

Yuan Feng, Haoyu Guo, Junlin Lv, S. Kevin Zhou, and Xike Xie. DefensiveKV: Taming the fragility of KV cache eviction in LLM inference. In *International Conference on Learning Representations*, 2026.

Yoav Gelberg, Yam Eitan, Michael Bronstein, Yarin Gal, and Haggai Maron. Training transformers for KV cache compressibility. *arXiv preprint arXiv:2605.05971*, 2026.

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

Nathan Godey, Alessio Devoto, Yu Zhao, Simone Scardapane, Pasquale Minervini, Éric de la Clergerie, and Benoît Sagot. Q-Filters: Leveraging QK geometry for efficient KV cache compression. *arXiv preprint arXiv:2503.02812*, 2025.

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.

Mohsen Hariri, Alan Luo, Weicong Chen, Tianyi Zhang, Qifan Wang, Xiaotian Han, and Vipin Chaudhary. Quantize what counts: More for keys, less for values. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 26393–26420, 2026. doi: 10.18653/v1/2026.findings-acl.1314.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. ZipCache: Accurate and efficient KV cache quantization with salient token identification. In *Advances in Neural Information Processing Systems*, volume 37, pp. 68287–68307, 2024. doi: 10.52202/079017-2181.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, volume 37, pp. 1270–1303, 2024. doi: 10.52202/079017-0040.

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

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

Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 24391–24402. PMLR, 2025.

Yuxiang Huang, Binhang Yuan, Xu Han, Chaojun Xiao, and Zhiyuan Liu. Locret: Enhancing eviction in long-context LLM inference with trained retaining heads on consumer-grade devices. *Transactions on Machine Learning Research*, 2025.

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. *arXiv preprint arXiv:2310.06825*, 2023.

Bo Jiang and Sian Jin. KVSculpt: KV cache compression as distillation. *arXiv preprint arXiv:2603.27819*, 2026.

Jushi Kai, Yixuan Wang, Boyi Zeng, Haoli Bai, Bo Jiang, Ziwei He, and Zhouhan Lin. FreqKV: Key-value compression in frequency domain for context window extension. In *International Conference on Learning Representations*, 2026.

Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. GEAR: An efficient KV cache compression recipe for near-lossless generative inference of LLM. *arXiv preprint arXiv:2403.05527*, 2024.

Jang-Hyun Kim, Junyoung Yeom, Sangdoo Yun, and Hyun Oh Song. Compressed context memory for online language model interaction. In *International Conference on Learning Representations*, 2024.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. In *Advances in Neural Information Processing Systems*, volume 38, pp. 167563–167591, 2025. doi: 10.52202/085713-5585.

Junhyuck Kim, Ethan Ewer, Taehong Moon, Jongho Park, and Dimitris Papailiopoulos. Not all bits are equal: Scale-dependent memory optimization strategies for reasoning models. In *International Conference on Learning Representations*, 2026a.

Minsoo Kim, Arnav Kundu, Han-Byul Kim, Richa Dixit, and Minsik Cho. EpiCache: Episodic KV cache management for long-term conversation on resource-constrained environments. In *International Conference on Machine Learning*, 2026b.

Anna Kuzina, Maciej Pióro, and Babak Ehteshami Bejnordi. KaVa: Latent reasoning via compressed KV-cache distillation. In *International Conference on Learning Representations*, 2026.

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

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods. In *International Conference on Learning Representations*, pp. 66063–66093, 2025.

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. In *Advances in Neural Information Processing Systems*, volume 37, pp. 22947–22970, 2024. doi: 10.52202/079017-0722.

Manlai Liang, JiaMing Zhang, Xiong Li, and Jinlong Li. LagKV: Lag-relative information of the KV cache tells which tokens are important. *arXiv preprint arXiv:2504.04704*, 2025.

Chengxi Liao and Zeyi Wen. Channel-aware mixed-precision quantization for efficient long-context inference. In *International Conference on Learning Representations*, 2026.

Tengxuan Liu, Shiyao Li, Jiayi Yang, Tianchen Zhao, Feng Zhou, Xiaohui Song, Guohao Dai, Shengen Yan, Huazhong Yang, and Yu Wang. PM-KVQ: Progressive mixed-precision KV cache quantization for Long-CoT LLMs. In *International Conference on Learning Representations*, pp. 107881–107900, 2026.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: KV cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pp. 38–56. ACM, 2024a. doi: 10.1145/3651890.3672274.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 32332–32344. PMLR, 2024b.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2017.

Maxim Milakov and Natalia Gimelshein. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*, 2018.

Mistral AI. Mistral-7B-Instruct-v0.3 model card, 2024. URL https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 37396–37412. PMLR, 2024.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized KV cache compaction in a single forward pass. *arXiv preprint arXiv:2606.07878*, 2026.

Matanel Oren, Michael Hassid, Nir Yarden, Yossi Adi, and Roy Schwartz. Transformers are multi-state RNNs. *arXiv preprint arXiv:2401.06104*, 2024.

Junyoung Park, Dalton Jones, Matthew Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. KeyDiff: Key similarity-based KV cache eviction for long-context LLM inference in resource-constrained environments. In *Advances in Neural Information Processing Systems*, volume 38, pp. 5983–6019, 2025. doi: 10.52202/085713-0210.

Wonpyo Park and Seung-won Hwang. TaskPress: Query-agnostic KV cache compression via task-guided pruning. *arXiv preprint arXiv:2608.03276*, 2026.

Junjie Peng, You Wu, Haoyi Wu, Jialong Han, Xiaohua Xie, Kewei Tu, and Jianhuang Lai. GRKV: Global regression for training-free KV cache compression in long-context LLMs. *arXiv preprint arXiv:2605.31105*, 2026.

Akshat Ramachandran, Marina Neseem, Charbel Sakr, Rangharajan Venkatesan, Brucek Khailany, and Tushar Krishna. ThinKV: Thought-adaptive KV cache compression for efficient reasoning models. In *International Conference on Learning Representations*, 2026.

Utkarsh Saxena and Kaushik Roy. KVLinC: KV cache quantization with Hadamard rotation and linear correction. *arXiv preprint arXiv:2510.05373*, 2025.

Konrad Staniszewski and Adrian Łańcucki. KV cache transform coding for compact storage in LLM inference. In *International Conference on Learning Representations*, pp. 37768–37815, 2026.

Chenxia Tang, Jianchun Liu, Hongli Xu, and Liusheng Huang. Adaptive KV-cache compression without manually setting budget. *arXiv preprint arXiv:2509.03136*, 2025.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 47901–47911. PMLR, 2024.

Haolin Tian, Yuzhe Liu, and Tonghan Wang. Every cache entry earns its place: Global allocation of resolution and coverage for KV cache compression. *arXiv preprint arXiv:2608.07001*, 2026.

Philippe Tillet, H. T. Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, 2019. doi: 10.1145/3315508.3329973.

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

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, pp. 21875–21895, 2024.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*, pp. 37228–37253, 2025.

Hui Xie, Peng Xiao, Yutong Deng, Shuoran Dou, Jian Yang, and Jinyang Guo. SemPIC: Learning semantic position-independent KV caches. *arXiv preprint arXiv:2607.28069*, 2026.

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*, 2025a.

Jingbo Yang, Bairu Hou, Wei Wei, Yujia Bao, and Shiyu Chang. KVLink: Accelerating large language models via efficient KV cache reuse. In *Advances in Neural Information Processing Systems*, volume 38, pp. 133797–133824, 2025b. doi: 10.52202/085713-4461.

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.

Xintong Yang, Hao Gu, Binxing Xu, Lujun Li, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Yike Guo, and Sirui Han. IndexMem: Learned KV-Cache eviction with latent memory for long-context LLM inference. In *International Conference on Machine Learning*, 2026.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*, pp. 94–109. ACM, 2025. doi: 10.1145/3689031.3696098.

Zhiyuan Yu, Shijian Xiao, Zhangyue Yin, Xiaoran Liu, Lekai Xing, Wenzhong Li, Cam-Tu Nguyen, and Sanglu Lu. ProtoKV: Long-context knowledges are already well-organized before your query. In *International Conference on Learning Representations*, 2026.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Yuhang Zhan, Lisi Chen, and Shuo Shang. ResKV: Reconstructing omitted attention contributions for fixed-budget KV cache compression. *arXiv preprint arXiv:2607.29591*, 2026.

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. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 8092–8105, 2025a. doi: 10.18653/v1/2025.findings-emnlp.429.

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

Tao Zhang, Ziqian Zeng, Hao Peng, Huiping Zhuang, and Cen Chen. MixKVQ: Query-aware mixed-precision KV cache quantization for long-context reasoning. *arXiv preprint arXiv:2512.19206*, 2025b.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710, 2023. doi: 10.52202/075280-1506.

Table 3: Training configuration. The parameter column counts trainable LoRA weights in millions (M). The full-cache loss weight is λ; tolerance bounds the absolute change in full-cache accuracy from the frozen model, in percentage points (pp), for development-set checkpoint eligibility.

Model	Parameters	Learning rate	λ	Tolerance
Llama-3.1-8B	54.5M	2×10^{-4}	0	4 pp
Qwen3-8B	61.3M	5×10^{-5}	1	8 pp

A Training and evaluation

The matched comparisons evaluate allocation and adaptation within the same selection rule. This section specifies the training and scoring protocols, checks the realized byte budgets, and reports uncertainty in the RULER-13 results. Training and compressor controls then examine which choices affect the adapted configuration.

A.1 Training recipe

We use the instruction-tuned Llama-3.1-8B and Mistral-7B-Instruct-v0.3 (Mistral AI, 2024) checkpoints, and Qwen3-8B. LoRA adapts the Q, K, V, and O projections in every layer at rank 64. Table 3 gives the architecture-specific learning rates and full-cache anchor weights. For Llama and Qwen3, training seeds 0–3 use the same 701 training examples and 616 development examples, with a checkpoint selected from three epochs at a 3% byte budget. The full-cache accuracy criterion determines eligibility on development data. Test-set distortion describes behavior after selection.

Unless stated otherwise, compression uses KeyDiff with 4-bit KV. In each layer and KV head, the first four context positions (attention sinks) and the last 32 are always retained within the position budget. Each key and value channel is quantized in groups of 128 retained positions. Scoring, protection, sequence ordering, and quantization rules are identical during training and evaluation.

Full training objective. The compressed-cache loss is defined in Eq. 1. To also constrain the adapter's behavior when compression is disabled, we can add a full-cache consistency term, or *anchor*. Let $p^{\text{full}}_{t,\theta}$ denote the student's uncompressed answer distribution. The objective is

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{comp}} + \lambda \mathcal{L}_{\text{full}}, \\ \mathcal{L}_{\text{full}} &= \frac{1}{|\mathcal{A}|} \sum_{t \in \mathcal{A}} \mathrm{KL}\big(p^{\text{teacher}}_{t} \,\|\, p^{\text{full}}_{t,\theta}\big)\,. \end{aligned} \tag{2}$$

We use $\lambda = 1$ for Qwen3 and $\lambda = 0$ for Llama. Setting $\lambda = 0$ removes the full-cache student term; the frozen full-cache teacher still supplies the targets for $\mathcal{L}_{\text{comp}}$. Both terms use the same corpus-supplied answer span.

Training cost. Training costs 16.7 GPU-hours per Llama adapter and 43.6 per Qwen3 adapter, using two and four A100-40GB GPUs, respectively.

Training at longer contexts. With the full-cache anchor, Qwen3 training retains activations from both full- and compressed-cache prefill for backpropagation. This recipe does not fit a 16K context on our available A100-40GB GPUs, even with feed-forward gradient checkpointing. The 16K control therefore uses Llama, whose $\lambda = 0$ objective backpropagates only through the compressed student. The frozen full-cache teacher still supplies the training targets.

A.2 Evaluation protocol and byte accounting

Qwen3 disables its optional reasoning mode in training and evaluation. Greedy decoding follows each RULER task's generation limit. Question answering (QA) accepts any valid answer alias; other categories score the fraction of reference answer spans recovered. The four-needle slice contains two single-needle tasks (`niah_single_2`, `niah_single_3`) and two multikey tasks (`niah_multikey_2`, `niah_multikey_3`). These needle-in-a-haystack (NIAH) tasks retrieve specified information among distractors. At 4 bits, byte budgets of 10%, 5%, and 3% retain 40%, 20%, and 12% of positions, respectively, before metadata.

Generation limits. Before decoding, we check that each task's generation limit can fit a full-credit answer under the model's tokenizer. The limits are 128 tokens for needle tasks, 120 for common-word extraction, 50 for frequent-word extraction, 32 for QA, and 30 for variable tracking. We sum the token lengths of all distinct required spans, or use the shortest accepted QA alias, so that multi-span answers fit as a whole.

Numerical precision and evaluators. In training and standalone evaluation, base weights and unquantized KV use float16 (fp16) for Llama and bfloat16 (bf16) for Qwen3 and Mistral; both formats occupy two bytes per scalar. Trainable LoRA parameters use float32. Main RULER comparisons use KVPress with its RULER dataset mirror. Additional controls use a standalone evaluator and are identified below. Adaptation comparisons pair frozen and adapted 4-bit results within the same evaluator. Repository documentation records the exact model, dataset, and software revisions alongside the commands and saved predictions.

Re-scoring seed 0 from Table 1 with the standalone evaluator gives 12 cross-evaluator comparisons: frozen and adapted 4-bit configurations on two architectures at three budgets. The mean absolute difference is 0.21, the maximum is 0.71, and the signed mean is -0.04. Full-cache scores agree within 0.04.

Logical payload. For L layers, H KV heads per layer, head dimension D, n retained positions per head, and b bits per scalar, the idealized key/value payload in bytes is (Liu et al., 2024b; Hooper et al., 2024)

$$B_{\text{logical}} = \frac{2LHnDb}{8}. \tag{3}$$

Relative to the original context length T, the retained fraction is $n/T = \min\{1, 16r/b\}$ at byte budget r, before metadata and rounding. The budget applies to the context KV representation; model and adapter weights are shared across requests and reported separately in Appendix E.1.

Realized-byte matching. The logical payload in Eq. 3 omits group metadata. With one 16-bit scale and zero point per group of 128 positions per channel, the effective width is $4 + 32\lceil n/128 \rceil / n$ bits for n retained positions. Whole-position rounding and retained indices also affect the footprint. For KeyDiff across all 6500 examples per model, nominal matching gives the 4-bit configuration between 106.6% and 111.3% of the matched 16-bit control's realized bytes, with medians ranging from 107.6% to 108.4%.

For an exact match, we charge both configurations two bytes per retained index and choose the largest retained count whose codes, scales, zero points, and indices fit the 16-bit reference on every example. Short contexts impose the tightest constraint because they amortize group metadata over fewer positions. The resulting nominal byte budgets, in descending order, are 9.20%, 4.55%, and 2.70% for Llama, and 9.20%, 4.55%, and 2.69% for Qwen3. At these budgets, frozen 4-bit KVPress scores are 79.59, 67.51, and 60.93 on Llama, and 74.47, 61.41, and 49.72 on Qwen3. Reallocation adds 18.4 to 34.5 points over the matched 16-bit control.

We pair adapted and frozen runs within the standalone evaluator to measure adaptation at these exact budgets. Llama's four-seed adapted means are 81.52, 72.50, and 66.81, against frozen scores of 79.42, 67.48, and 61.29. The corresponding gains are 2.11, 5.02, and 5.51 points. Qwen3 seed 0 scores 77.17, 64.80, and 56.05, against frozen scores of 74.29, 61.47, and 49.08, for gains of 2.88, 3.33, and 6.97 points. Gains are computed before rounding. Relative to nominal runs in this same evaluator, exact matching costs the adapted configurations 1.07 to 1.86 points. Qwen3's other seeds lack per-example nominal counterparts and therefore do not enter that paired correction.

What changes under token matching. At 10%, matched 16-bit selection retains 10% of positions and scores 61.17 on Llama and 48.21 on Qwen3. Frozen 4-bit selection at 3% retains 12% and scores 62.45 and 52.91, respectively. Once retained counts are close, so are the scores. This explains why the byte axis matters: it exposes the opportunity to spend lower precision on coverage, whereas a position-matched comparison removes that opportunity.

Table 4: Four trained seeds for 4-bit ByteKV-KeyDiff. Full RULER-13 macro, 500 examples per task; these runs underlie Table 1.

Model and byte budget	Seed 0	Seed 1	Seed 2	Seed 3
Llama, 10%	83.6	80.2	86.2	80.6
Llama, 5%	73.3	70.9	75.6	73.8
Llama, 3%	67.8	66.1	69.3	69.0
Qwen3, 10%	78.9	78.5	79.1	77.8
Qwen3, 5%	66.2	66.2	66.3	65.1
Qwen3, 3%	58.2	58.0	58.1	56.8

A.3 Training-seed and test-sample uncertainty

Table 4 reports all four main seeds. The population standard deviation ranges from 1.3 to 2.5 points on Llama and is about 0.5 on Qwen3. Every seed loses accuracy as the budget tightens. Adaptation's mean gain exceeds its seed standard deviation in five of six cells; Llama at 10% is the exception, with two seeds below the frozen 4-bit control. Standalone re-evaluation of all four Llama adapters differs from KVPress by at most 0.60 per seed and 0.20 in the mean at each budget.

Paired bootstrap. We resample examples within each task for 10,000 replicates, using the same indices for the frozen and adapted configurations across all four seeds. Tasks remain fixed, so these intervals measure sampling uncertainty in the 13-task mean, conditional on the trained adapters. At byte budgets of 10%, 5%, and 3%, Llama's mean adaptation differences and 95% intervals are $+1.69$ $[+1.08, +2.29]$, $+4.75$ $[+4.22, +5.30]$, and $+5.61$ $[+5.07, +6.15]$; Qwen3's are $+2.63$ $[+2.10, +3.15]$, $+2.24$ $[+1.69, +2.80]$, and $+4.83$ $[+4.25, +5.42]$. All exclude zero, while the mild-budget Llama result remains sensitive to retraining. Means from the stored predictions reproduce Table 1 within 0.02.

A.4 Training and compressor controls

These controls test whether recovery under 4-bit compression depends on training at the tightest budget, the distillation objective, or particular quantization and protection settings.

Training-budget specialization. We compare the default adapter trained at a 3% byte budget with separate adapters trained at each budget and one adapter trained jointly on all three, using the same disjoint four-needle test slice. At budgets of 10% and 5%, the default Llama adapter scores 91.9 and 74.2, respectively. Mixed-budget training scores 86.2 and 73.3, and per-budget training scores 76.9 and 71.6. Thus matching or mixing the training budgets does not improve these milder-budget results. The separate per-budget runs score 69.6 on Llama and 53.50 on Qwen3 at 3%, versus the default adapters' 65.6 and 52.15. At this budget, both sets already train at 3%: the comparison measures variation between training runs with the budget held fixed. The alternative recipes use one seed.

Objective and full-cache anchor. Adding cross-entropy on the reference answer tokens to KL reduces Llama's four-needle score by 1.55 points (Table 5). The teacher's answer distribution already supplies useful supervision.

The full-cache anchor asks the adapted model to preserve the frozen model's predictions when no compression is applied. This adds a constraint on uncompressed use to the objective that trains recovery from sparse, quantized KV. Removing the anchor improves Qwen3's compressed accuracy by 1.90 points with no full-cache loss on the four-needle slice. The full RULER-13 control gives 58.84 with $\lambda = 0$ versus 57.86 with $\lambda = 1$ at nominal 3%, using seed 0 and the same standalone evaluator. With compression disabled, the $\lambda = 0$ adapter scores 95.72 against the same frozen reference of 95.36. The selected Qwen3 adapter thus retains full-cache retrieval accuracy without the consistency term. Together with Llama's anchor-free recipe, this shows that compressed-cache recovery in these experiments does not require the extra full-cache objective.

Quantization granularity and protected positions. We change one inference-time setting with the Llama adapter fixed (Table 6). Group sizes 32 and 256 change realized bytes by $+17.6\%$ and -2.9%, but change adapted accuracy by only $+0.45$ and -0.40 points. Removing the recent window

Table 5: Objective and anchor ablations at 3%, four needle families, one seed. Frozen uses the same compressor without adaptation; Δ compares with the corresponding ByteKV-KeyDiff objective.

Model	Variant	Frozen	Adapted	Δ vs. ByteKV-KeyDiff
Llama-3.1-8B	ByteKV-KeyDiff (KL only)	50.65	65.55	
	+ next-token cross-entropy	50.65	64.00	−1.55
Qwen3-8B	ByteKV-KeyDiff (λ=1)	43.45	52.15	
	λ=0	43.45	54.05	+1.90

Table 6: Inference-time compressor changes with the Llama ByteKV-KeyDiff adapter fixed, four-needle slice at 3%. Gain is adapted minus frozen; bold marks its largest change from the default row. Effective bits for full groups include 16-bit scale and zero point: $4 + 32/\text{group size}$.

Inference-time change	Effective bits	Frozen	Adapted	Gain
default (group 128, initial 4, recent 32)	4.250	50.65	65.55	+14.90
group size 32	5.000	50.55	66.00	+15.45
group size 256	4.125	50.65	65.15	+14.50
recent 0	4.250	50.75	66.15	+15.40
recent 128	4.250	51.30	63.05	+**11.75**

instead improves accuracy by 0.60, while expanding it from 32 to 128 positions costs 2.50. The larger window improves the frozen 4-bit control by 0.65 points, reducing the adaptation gain by 3.15. A protected position consumes a slot that the scoring rule could otherwise assign to the context. Frozen and adapted models can give positions different KeyDiff scores, so increasing protection can alter their retained sets differently. Adaptation still recovers accuracy without a recent window.

The group-size result falls within the preregistered ±5-point band. Removing the recent window contradicts the predicted loss of more than five points. Nearly all variation comes from `niah_multikey_2`: the single-needle tasks stay at ceiling and `niah_multikey_3` changes by one scored example.

Corpus size and update count. The single- and multi-format corpora differ in both breadth and size (Table 17). To examine exposure separately, we measure full-cache accuracy on five families absent from the training formats: common-word extraction, frequent-word extraction, variable tracking, and single- and multi-hop QA. The single-format adapter changes accuracy by −9.40 points. Subsampling the same narrow corpus to 236 examples gives −2.61 and −1.60 across two independent draws. At 236 examples, restoring the original update count with nine epochs (1888 steps) gives −5.70. Repeated exposure therefore accounts for part of the loss even when the examples are fixed. These controls concern uncompressed behavior. The corpus comparison under compression remains a joint change in breadth and size (Appendix D.2).

A.5 Adapter rank

Table 7 varies capacity at fixed KeyDiff selection and 3%. On Llama, ranks 16 and 32 retain accuracy close to rank 64, whose four-seed scores on this slice range from 65.4 to 67.6. Rank 8 falls to 60.3. Qwen3 instead loses accuracy monotonically as rank decreases, by 2.1 points at rank 32 and 4.1 at rank 8. The smaller-rank runs use one seed, and the main experiments keep rank 64 on both models.

B Allocation and adaptation under fixed selection

The configuration grid groups the complete method and its partial configurations by selection rule. Matched 16-bit and frozen 4-bit controls measure allocation; frozen and adapted 4-bit pairs measure model adaptation. Repeating these comparisons tests how recovery depends on the rule.

B.1 Matched controls and the configuration grid

Table 8 groups matched 16-bit controls, frozen 4-bit controls, and ByteKV with adapted 4-bit KV.

For KeyDiff, KVzip, Expected Attention, LagKV, and k-norm, reallocation improves 27 of 30 selector–architecture–budget cells relative to 16-bit controls with each rule's baseline protection

Table 7: LoRA rank at fixed KeyDiff selection and 3%. Four-needle macro, 500 examples per family, one seed per cell. Rank 64 is the default configuration; the shared frozen scores are 50.65 for Llama and 43.45 for Qwen3.

Model	Rank 64	Rank 32	Rank 16	Rank 8
Llama-3.1-8B	65.55	67.80	65.65	60.25
Qwen3-8B	52.15	50.05	49.80	48.10

policy, including 10 of 10 at 3%. The exceptions are KVzip at 10% on both models and at 5% on Llama. Its crossover lies between 5% and 3% on Llama, and between 10% and 5% on Qwen3. Thus the budget at which the original-baseline comparison favors 4-bit coverage depends on the model.

Comparisons with original baselines also change protection. The 4-bit operator always retains initial and recent positions and restores sequence order. The matched 16-bit controls use these same conventions. For KeyDiff, matching protection changes the 16-bit reference by -7.58 points on Llama at 3% and by $+10.11$ on Qwen3 at 10%. An original reference can therefore misstate the allocation gain in either direction.

Matched controls cover KeyDiff, Expected Attention, LagKV, and k-norm on two models at three budgets. Reallocation gains are positive in all 18 added cells beyond the six KeyDiff main cells, ranging from 4.8 to 46.8 points. Adaptation improves 23 of 24 cells for the four content-based selectors other than KeyDiff. Its sole exception is Llama k-norm at 10%. KVzip contributes to the adaptation tally but lacks a matched 16-bit control. Positional streaming is outside both tallies.

B.2 Frozen baselines and sensitivity to budget

Table 9 supplies original-baseline context for the matched comparisons. Across three architectures, each published nonrandom frozen baseline loses at least 44% of its score as the byte budget tightens from 10% to 3%. The budget dependence also appears in the ten-task retrieval subset (Appendix C.3), so it is not driven solely by aggregation and multi-hop QA.

The additional KVzip-adapted row is sensitive to training seed. Three Llama seeds span 7.21, 6.08, and 7.96 points at the three budgets, with means of 84.77, 76.49, and 62.32. The table retains seed 0 for consistency with the other selector variants. Frozen baselines and the main KeyDiff evaluation use KVPress, while the KVzip-adapted runs use the standalone evaluator. All use task-specific generation budgets and the same per-category metric.

B.3 How the selection rule constrains adaptation

The four-needle results in Table 8 extend the selector comparison to individual retrieval settings. At 3%, adaptation improves every tested content-based selector, including those whose frozen 4-bit accuracy is near zero. The full-suite comparisons appear in Table 2. A second Llama k-norm training seed produced no development-eligible checkpoint among its three epochs and therefore has no test score.

The scoring function and its input have distinct roles. KeyDiff, k-norm, LagKV, and Expected Attention compute scores from the current model's KV vectors; LoRA can change their rankings without changing the scoring function. Initial and recent protected positions remain fixed. KVzip fixes the retained positions in our control by scoring them with the frozen model. Its Llama four-needle score rises from 15.50 to 61.50 with adaptation. The adapter can still change the KV vectors at those positions and how subsequent attention uses them. This control establishes recovery without changing the keep set. The fixed scores are an experimental choice: recomputing KVzip's scores with the adapted model would permit selection to change.

In our implementation, KVzip's reconstruction scorer materializes attention weights and requires 2.5 to 3 MiB of peak memory per context token, exceeding a 40 GiB card beyond roughly 7K. Its evaluation is consequently limited to RULER.

Positional selection. Positional streaming retains only initial and recent positions and cannot move them toward an answer. No Llama epoch satisfies development eligibility; Qwen3 selects a checkpoint but remains at 12.95 on the four-needle slice with or without adaptation. On full RULER-13, adaptation changes accuracy from 54.00 to 53.04 at the largest budget, from 37.20 to

Table 8: **Configuration grid at fixed rules and byte budgets.** RULER-13 and its four-needle slice each use 500 examples per task. Matched frozen 16-bit KV, frozen 4-bit KV, and ByteKV with adapted 4-bit KV form the within-selector comparisons. Budget columns give the percentage of full 16-bit KV bytes.

Configuration	Llama-3.1-8B				Qwen3-8B			
	RULER-13			Four-needle	RULER-13			Four-needle
	10%	5%	3%	3%	10%	5%	3%	3%
16-bit KV, frozen, matched protection								
KeyDiff	61.17	44.22	26.47	—	48.21	30.68	20.30	—
Expected Attention	30.86	15.21	10.07	0.95	58.83	39.69	20.23	5.90
LagKV	37.28	15.30	9.97	—	34.91	12.78	8.24	—
k-norm	20.01	7.53	5.15	—	8.90	5.78	4.74	—
4-bit KV, frozen								
KeyDiff	80.97	68.63	62.45	50.65	75.95	63.70	52.91	43.45
KVzip	79.86	54.77	37.66	15.50	74.56	42.82	29.97	1.35
Expected Attention	77.65	56.08	36.33	10.00	83.49	66.69	60.40	27.55
LagKV	78.92	56.81	41.07	14.55	75.38	59.55	42.53	20.60
k-norm	43.39	28.18	22.14	1.10	52.10	18.14	9.58	1.55
streaming (positional)	—	—	—	—	54.00	37.20	25.44	12.95
***ByteKV: 4-bit KV, adapted**, fixed selector per row*								
KeyDiff	82.64	73.38	68.06	65.55	78.58	65.94	57.75	52.15
KVzip	86.97‡	78.35‡	63.99‡	61.50‡	91.74	66.15	40.72	13.35
Expected Attention	78.85	65.22	56.12	63.95	92.40	79.49	70.83	53.40
LagKV	80.94‡	72.64‡	64.35‡	66.80‡	86.50	77.24	66.30	48.10
k-norm	43.18†	31.57†	22.85†	15.20†	65.86	46.73	33.74	23.80
streaming (positional)	—	—	—	—	53.04	37.97	30.55	12.95
ByteKV with the bit-width varied (KeyDiff held fixed)								
16 bits	50.69	44.98	39.77	—	53.73	43.83	34.44	—
8 bits	**67.28**	**59.29**	**54.62**	—	66.98	54.97	44.50	—
2 bits	52.75	53.12	49.08	—	**87.67**	**72.79**	**61.87**	—
ByteKV with the training corpus varied (KVzip held fixed)								
multi-format synthetic	90.00	74.87	58.58	42.20	81.08	48.60	33.02	3.85
frozen full cache	*95.75*	*95.75*	*95.75*	*99.95*	*95.36*	*95.36*	*95.36*	*100.00*

The 16-bit controls match the protection and ordering of the 4-bit operator (Section 5.1). Expected Attention (Devoto et al., 2025) uses its value-norm component in all three configurations. KVzip and positional streaming lack matched 16-bit controls. Bold compares the listed alternative bit-widths under fixed KeyDiff; its 4-bit reference appears above. 4-bit adapted KeyDiff scores on RULER-13 average four seeds; other adapted entries use one. The corpus block uses the standalone evaluator. Dashes denote unmeasured results. No Llama streaming checkpoint meets development eligibility, so neither its frozen nor adapted test configuration was evaluated. The following markers apply to Llama only: †four-needle full-cache change of -26.5 points. ‡four-needle change of $+0.00$, with RULER-13 changes of -10.46 (LagKV) and -5.06 (KVzip). The corresponding Qwen3 changes remain within the development tolerance on both test sets. These are post-selection measurements (Appendix F.1).

37.97 at the middle budget, and from 25.44 to 30.55 at the tightest budget. The two task sets therefore respond differently to the same fixed-position cache and adapter. This pattern is consistent with their different demands on the retained context. Since KV vectors are contextual representations, absence of a literal answer position alone does not establish that its information is absent from every retained vector.

B.4 Precision and coverage

We examine both the precision–coverage trade-off among trained configurations and the effect of adaptation at a fixed bit-width. The latter comparison pairs the same selector, quantizer, nominal byte budget, test examples, and generation limits with and without the adapter.

Figure 4 examines precision and coverage as part of the trained configuration. KeyDiff and protection remain fixed, and each width has its own adapter trained at 3%. Lower precision buys more positions while changing the representation used during training and inference. The resulting trade-off differs by architecture. 4-bit precision gives Llama's highest observed scores. Qwen3's 2-bit configuration scores highest at every tested budget, including 87.67 versus 78.58 at 10% and a 4.12-point margin

Table 9: **Frozen references across byte budgets.** Full RULER-13, 500 examples per task, identical examples across configurations. Columns give the remaining byte budget. Frozen controls retain their baseline protection policies; SnapKV is prefix-only. The ByteKV rows are reference systems, with four seeds for KeyDiff and one for KVzip. Frozen baselines and ByteKV-KeyDiff use KVPress; ByteKV-KVzip uses the standalone evaluator. The paired Mistral KeyDiff extension appears in Appendix D.1.

	Llama-3.1-8B			Qwen3-8B			Mistral-7B		
Method	10%	5%	3%	10%	5%	3%	10%	5%	3%
KeyDiff	61.13	48.64	34.05	38.09	21.36	14.13	54.63	39.13	25.61
KVzip	90.50	63.47	27.05	87.22	38.03	19.40	86.89	62.52	43.06
Expected Attention	30.47	12.72	10.36	65.75	37.10	16.50	28.95	14.73	7.11
LagKV	37.25	13.18	8.05	34.73	12.42	8.25	28.51	16.28	8.50
SnapKV (prefix-only)	22.96	14.28	8.12	19.46	13.66	8.84	20.05	11.62	7.92
StreamingLLM	24.45	14.86	7.90	22.14	9.86	5.82	25.18	13.83	6.13
k-norm	18.32	7.45	4.27	5.72	1.68	1.46	11.31	6.67	4.72
Random	0.13	0.10	0.05	3.67	2.88	1.97	4.99	4.99	3.60
ByteKV-KeyDiff	82.64	73.38	68.06	78.58	65.94	57.75	—	—	—
ByteKV-KVzip	86.97	78.35	63.99	91.74	66.15	40.72	—	—	—

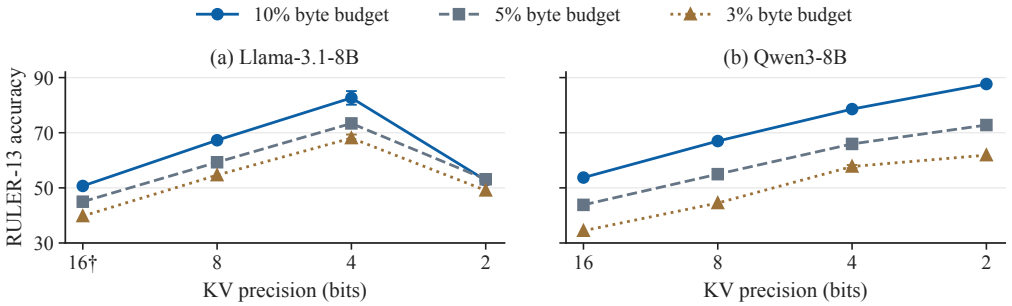


Figure 4: **The precision–coverage trade-off depends on the model.** Full RULER-13 with KeyDiff and protection fixed. Each width has a separately trained adapter at 3%, evaluated at all three budgets. 4-bit points show four-seed means ± population standard deviation; other widths use one seed. †Llama's 16-bit adapter loses 27.4 full-cache points on the four-needle slice. Qwen3's 2-bit result is the best tested endpoint. Exact scores: Table 8.

at 3%. The lowest tested width is 2 bits, so an interior optimum has not been established. Only the 4-bit reference averages multiple seeds.

The additional 3-bit control sharpens this contrast. Qwen3 scores 61.4 at 3%, between 4 bits (57.8) and 2 bits (61.9). Llama's 3-bit adapted score falls below its own frozen compressed baseline. Thus a successful precision choice on Qwen3 does not transfer directly to Llama with the tested training recipes. Its full-cache loss, and that of the 16-bit diagnostic, are reported with the other uncompressed evaluations in Appendix F.1.

Recovery at fixed precision. Table 10 supplies frozen and adapted controls at 2 and 8 bits using the standalone evaluator. Each pair is scored together; Figure 4 retains the KVPress evaluation of the adapted sweep. At 2 bits, both models improve at every budget. Llama's frozen scores are 0.12 and 0.21 at the two milder budgets, while the adapted model scores 52.98 and 53.44. At 3%, both architectures improve on 12 of 13 tasks. Llama's exception is variable tracking, which falls from 72.68 to 26.56; Qwen3's is common-word extraction. The recovery therefore spans task families while retaining task-specific failures.

At 8 bits, Llama's adapted score declines by 2.13 and 1.89 points at the milder budgets and improves by 2.56 at 3%. Qwen3 improves at all three. The adapter is fitted at the tightest budget, and its gains depend on the compression setting encountered at evaluation. The 2-bit results show that low-precision KV vectors can remain useful after training the model through them; the 8-bit controls show that fitting one such operator need not improve every budget.

Table 10: **Adaptation with precision and byte budget fixed.** Full RULER-13 in the standalone evaluator, 500 examples per task, seed 0. Each bit-width has its own KeyDiff adapter trained at 3%. Frozen and adapted use the same scoring, protection, and quantization rules.

		10% budget		5% budget		3% budget		Full-cache
Model	Bits	Frozen	Adapted	Frozen	Adapted	Frozen	Adapted	Δ
Llama-3.1-8B	8	69.41	67.28	61.12	59.23	51.91	54.47	−10.76
	2	0.12	52.98	0.21	53.44	12.63	49.55	−13.94
Qwen3-8B	8	64.19	66.93	47.45	55.06	34.64	44.64	+0.29
	2	70.43	87.61	54.78	72.43	40.50	61.95	+0.05

The last column evaluates the same adapter with compression disabled, relative to its frozen full-cache reference (95.75 for Llama, 95.36 for Qwen3). All four checkpoints satisfy the development eligibility criterion. Differences are computed before rounding. Test-set full-cache changes describe a separate usage condition (Appendix F.1).

Table 11: **Recovery by adapted projection.** Four-needle accuracy gain over the frozen 4-bit control at 3%, training seed 0; parentheses give the fraction of the default Q/K/V/O adapter's gain. Full-cache accuracy stays near 99.9. Projection placement specifies the direct update; prefill hidden states carry its effects to later layers.

LoRA on	Direct output	Llama Δ (% default)	Qwen3 Δ (% default)
Q, K, V, O (default)	all attention projections	+14.90 (100%)	+8.70 (100%)
Q, O	queries and attention output	+13.55 (91%)	+7.95 (91%)
K, V	cached keys and values	+12.70 (85%)	+7.35 (84%)
V	cached values	+14.90 (100%)	+5.30 (61%)
K	cached keys	+7.40 (50%)	+0.95 (11%)

C Representing and using the retained context

Allocation determines how many positions survive and at what precision. Adaptation changes the model that constructs and uses the retained KV vectors. Projection ablations and keep-set measurements examine how these changes interact with fixed selection; task-level results and specialists examine which kinds of information remain usable at the same budget.

C.1 Projection placement and keep-set changes

Table 11 varies which attention projections receive LoRA while holding the training recipe and compressor fixed. The default Q/K/V/O reference reuses the seed-0 checkpoint from the four-seed main experiment (Table 4), evaluated here on the four-needle subset. Each restricted-projection adapter is trained separately with the same seed. Q/O-only adaptation recovers 91% of the default adapter's gain on both architectures, but under GQA it has 3.2× as many parameters as one cache-writing projection. The equal-capacity comparison is V against K: V-only recovers more accuracy on both models. Since KeyDiff scores keys, recovery from adapting V raises a further question: does it also change which positions survive selection?

Why a fixed scoring rule can select different positions. LoRA is active during prefill. An adapted projection changes the hidden states that later layers use to write keys and values, even when those later projections are unadapted. First-layer K/V remain equal to the frozen model's in the Q/O-only configuration. On four needle families with 500 examples each, first-layer keep-set overlap is exactly 1.000 for V-only and Q/O-only adaptation, while both change selection in later layers.

Jaccard similarity is the size of the keep-set intersection divided by its union. Across layers, similarities between score-selected keep sets and the frozen model's are 0.70, 0.62, and 0.62 for K-only, V-only, and Q/O-only on Llama, and 0.88, 0.79, and 0.80 on Qwen3. K-only moves selection least. Together with its smaller seed-0 recovery, this shows why parameter placement cannot be read as an isolated intervention on either cache writing or cache reading: changes in intermediate token representations propagate between layers.

Which answer positions survive. For the default adapter, Table 12 measures keep-set overlap and answer-token survival outside the protected initial and recent positions, before decoding. Keep sets change most on the hard multikey families. Llama's `niah_multikey_2` gains 0.084 in answer-token survival at 3%, and 0.18 at 10%. Yet `niah_multikey_3` gains 0.043 at the tighter budget

Table 12: **Keep-set changes and answer-token survival.** Adapter on versus off at 3% byte budget. Jaccard measures overlap of KeyDiff keep sets; Δ survival measures the change in the fraction of answer-token positions retained, excluding protected initial and recent positions. Four-seed means over layers and KV heads, 500 examples per family.

	Llama-3.1-8B		Qwen3-8B	
Family	Jaccard	Δ survival	Jaccard	Δ survival
`niah_multikey_2`	0.43	+0.084	0.64	+0.033
`niah_multikey_3`	0.37	+0.043	0.56	+0.018
`niah_single_2`	0.65	+0.047	0.88	−0.008
`niah_single_3`	0.66	−0.037	0.88	−0.034

while its accuracy remains near zero. Retaining an answer-bearing token does not by itself ensure that the model can recover the answer.

The easy `niah_single_3` family retains accuracy despite slightly lower answer-token survival. This pattern is consistent with adaptation changing the content and use of retained KV vectors beyond preserving literal answer positions. These statistics average four seeds, with standard deviations below 0.02 in every cell. Mistral repeats the hard-versus-easy contrast: survival rises by about 0.02 on multikey tasks and falls by 0.09 to 0.17 on single-needle tasks.

Representations at protected positions. To test whether adaptation relies on rewriting the always-retained initial and recent positions, we replace their adapted K/V with the frozen model's. The reference answer's negative log-likelihood under teacher forcing is unchanged on Qwen3 and the easy single-needle families, and slightly improves on hard Llama families. Thus these adapted entries provide no negative-log-likelihood advantage in this control, which measures prediction of the reference answer rather than free-generation accuracy.

Variation across training seeds. Projection recovery also varies with training seed. Llama V-only recovers 58% to 100% of the default adapter's gain over four seeds, with a mean of 84%; Q/O-only recovers 79% to 112%, with a mean of 93%. Qwen3's three Q/O-only seeds gain 7.95, 7.65, and 7.90 points, averaging 90% of the default adapter's gain. The seed-0 V-only and K/V-only gains are 61% and 84%. Several parameter placements can therefore recover accuracy, with effects that propagate through successive layers to selection and retained representations.

C.2 Task-level recovery

Table 13 gives the four-seed task means underlying the RULER-13 averages in Table 1. Single-needle, multiple-query, and multiple-value tasks retain high accuracy at the 3% budget. Hard multikey tasks, common-word extraction, and QA account for most of the decline.

The adaptation gain has a different concentration on the two models. At seed 0 and 3%, `niah_multikey_2` contributes 4.32 of Llama's 5.39 macro-point gain (80%). Variable tracking and single-hop QA add 1.35 and 1.26, offset by a combined 2.25-point loss on common/frequent-word extraction and multi-query/value tasks. On Qwen3, ten of thirteen tasks improve, `cwe` is the only loss, and `niah_multikey_2` contributes 29% of the gain. The Llama macro therefore chiefly reflects recovery on distractor-heavy multikey retrieval; Qwen3's recovery is more distributed.

C.3 Localized retrieval and aggregation

The ten-task subset in Table 14 contains eight NIAH families, variable tracking, and single-hop QA. It separates localized retrieval from common-word extraction, frequent-word extraction, and multi-hop QA. At 3%, 4-bit ByteKV-KeyDiff scores 78.3 on Llama and 68.0 on Qwen3, above their respective RULER-13 averages. Removing the three families also raises every frozen baseline's score, while preserving its decline as the budget tightens. Thus aggregation and multi-hop requirements deepen the accuracy loss but do not explain it on their own. Full RULER-13 remains the primary measure.

Table 13: Per-family accuracy of 4-bit ByteKV-KeyDiff, four-seed mean. These 4096-token, 500-example task results underlie Table 1. Columns give the remaining byte budget.

		Llama-3.1-8B			Qwen3-8B		
Group	Family	10%	5%	3%	10%	5%	3%
single	niah_single_1	100.0	100.0	100.0	100.0	100.0	100.0
	niah_single_2	100.0	100.0	100.0	100.0	97.9	90.7
	niah_single_3	100.0	100.0	100.0	100.0	99.3	97.6
multi-key	niah_multikey_1	100.0	100.0	100.0	97.5	89.7	78.1
	niah_multikey_2	100.0	90.5	64.3	87.1	44.5	20.7
	niah_multikey_3	60.4	6.5	0.6	20.0	0.8	0.0
multi-query/value	niah_multiquery	96.7	95.9	96.3	99.5	94.8	87.3
	niah_multivalue	96.4	95.6	93.0	99.8	97.5	90.2
other tasks	vt	91.7	90.8	89.9	99.9	98.8	87.3
	cwe	40.8	20.1	7.4	57.0	22.3	8.5
	fwe	78.2	72.8	67.5	76.2	55.6	41.3
	qa_1	66.4	48.0	38.6	53.7	34.3	28.2
	qa_2	43.8	33.9	27.4	30.9	21.9	21.1

NIAH denotes needle-in-a-haystack retrieval. The other identifiers are variable tracking (vt), common-word extraction (cwe), frequent-word extraction (fwe), single-hop QA (qa_1), and multi-hop QA (qa_2).

Table 14: Ten-task retrieval macro: eight NIAH families, variable tracking, and single-hop QA, 500 examples per task. ByteKV-KeyDiff averages four seeds. Best frozen is the highest score among eight 16-bit baselines with original protection in each cell. Its selector can differ by cell, so it provides budget-sensitivity context rather than a matched 16-bit control.

Model	Configuration	10%	5%	3%
Llama-3.1-8B	ByteKV-KeyDiff	91.2	82.7	78.3
	Best frozen	96.3	72.6	37.7
Qwen3-8B	ByteKV-KeyDiff	85.8	75.7	68.0
	Best frozen	91.7	40.5	22.8

C.4 Family-specific training

A task-specific adapter can recover accuracy lost by the default adapter at the same compression budget. We train rank-8 specialists on 150 contexts per family and evaluate each on 50 disjoint examples (Table 15). On Llama, specialization raises niah_multikey_3 from 0.0 to 10.0 and cwe from 4.2 to 55.4, compared with frozen 4-bit scores of 0.0 and 21.4. On Qwen3, the same recipe leaves multikey accuracy at zero and gives 25.6 on cwe, close to the frozen 25.4. Some losses therefore reflect the training distribution, while the recoverable portion depends on the model.

Evidence span and coverage. A tokenizer-only diagnostic counts context positions protected by an oracle using the gold answers. The oracle protects a few positions for localized retrieval and hundreds to thousands for aggregation, approximately 400 for Llama cwe and 1800 for fwe, with the same scale contrast on Qwen3 and Mistral. Broadly distributed evidence consumes more of a fixed keep set and competes with other useful context. These counts describe the oracle's protected occurrences, rather than a minimum sufficient representation: Llama partially recovers cwe, while Qwen3 fails on the localized multikey task. Span size helps explain the coverage pressure, but does not determine whether adaptation succeeds.

Each specialist comparison uses one seed. For tasks scored as correct or incorrect, one example changes accuracy by 2.0 points; cwe gives partial credit in 0.2-point steps. The diagnostic assumes that task type is known at cache construction and does not evaluate a routing policy.

D Generalization across models and tasks

Mistral tests adaptation through a fixed 4-bit compressor on another architecture. LongBench and SCBench compare matched 16-bit controls, frozen 4-bit controls, and ByteKV outside the synthetic

Table 15: **Family-specific training at a fixed compression budget.** KeyDiff at 3%, 50 held-out examples per family, one seed. Default is the standard ByteKV-KeyDiff adapter; Specialist uses rank 8 trained on that family. Best frozen is the highest score among eight 16-bit baselines with original protection. Bold compares the three fixed-KeyDiff configurations.

Model	Family	Best frozen	Frozen 4-bit	Default	Specialist
Llama-3.1-8B	`niah_multikey_3`	0.0	0.0	0.0	**10.0**
	`cwe`	4.0	21.4	4.2	**55.4**
Qwen3-8B	`niah_multikey_3`	0.0	0.0	0.0	0.0
	`cwe`	7.0	25.4	6.8	**25.6**

Table 16: **Adaptation on three architectures.** Fixed KeyDiff and 4-bit storage, 500 examples per task. (a) Four-needle macro at 3%, seed 0 per model. (b) Full RULER-13 on Mistral with two independently trained adapters. Full cache evaluates the same weights without compression.

(a) Four-needle slice, 3% budget

Model	4-bit KV, frozen	ByteKV	Δ
Llama-3.1-8B	50.65	65.55	+14.90
Mistral-7B	46.35	57.00	+10.65
Qwen3-8B	43.45	52.15	+8.70

(b) Mistral-7B, full RULER-13

Configuration	10% budget	5%	3%	Full cache
4-bit KV, frozen	75.14	65.67	56.97	92.06
ByteKV, seed 0	80.60	72.43	65.74	91.31
ByteKV, seed 1	81.93	71.88	64.13	88.57

training distribution. Two training corpora test how the adapted configuration's transfer changes with the data used to fit it.

D.1 A THIRD ARCHITECTURE

Mistral-7B adds an architecture beyond the two 8B main models. Both Mistral training seeds improve 4-bit RULER-13 accuracy at every budget, with mean gains of 6.13, 6.48, and 7.97 points at 10%, 5%, and 3%, respectively (Table 16). Both train at 3% with learning rate 5×10^{-5} and anchor weight $\lambda = 1$, using the same development procedure as the main models. With compression disabled, the adapters change accuracy by -0.75 and -3.49 points relative to the frozen full-cache model.

The four-needle comparison reports seed 0 for each architecture and shows recovery on all three at 3%. The corresponding four-seed mean gains for the main models are 15.78 on Llama and 8.71 on Qwen3.

D.2 LONGBENCH TRANSFER AND TRAINING DISTRIBUTION

LongBench tests transfer beyond synthetic retrieval with four configurations: frozen 16-bit KV, frozen 4-bit KV, and ByteKV trained on either corpus in Table 17. All use KeyDiff with four initial and 32 recent protected positions. At nominal 3%, the 16-bit control retains 3% of positions; the other configurations retain 12% at 4 bits. The corpora are synthetic and disjoint from LongBench sources, differing in answer formats, sample counts, and development sets. We use each task's benchmark metric on a 0–100 scale and average these scores equally across tasks.

Both corpora ask the model to look up an access code from an explicit key. Both use synthetic log, prose, document, and code backgrounds; these are background styles, not separate reasoning or code-generation objectives. The single-format corpus uses random identifiers as answers. The multi-format corpus adds alphanumeric strings, numbers, adjective–noun words, and person names while preserving the lookup question and answer-prefix templates. The family counts in Table 17 count combinations of retrieval structure, answer format, and background style, rather than distinct downstream tasks.

Table 18 reports the twelve tasks whose contexts accommodate the 36 protected positions at this budget. Both adapted Llama configurations remain below frozen 4-bit KV, whereas both Qwen3

Table 17: Synthetic training corpora with the same LoRA rank, projection placement, and compressor. Single-format training produces the default ByteKV-KeyDiff adapter. Corpora are prepared separately for the two model tokenizers. All runs allow three epochs; sample counts and development sets differ.

Variant	Model	Families	Train	Development
single-format **(default)**	both	18	701	616
multi-format synthetic	Llama	81	236	26
multi-format synthetic	Qwen3	80	230	25

configurations exceed it. Multi-format training narrows Llama's gap by raising its macro from 31.16 to 35.90, against a frozen 4-bit score of 41.74. On Qwen3, it slightly lowers the score from 34.38 to 33.73.

Table 18: **LongBench transfer at 3%.** The 16-bit and 4-bit columns use frozen KeyDiff with matched protection. Single and multi are 4-bit ByteKV-KeyDiff trained on the two synthetic corpora, one seed each. All four configurations use the same twelve tasks and 2650 examples.

	Llama-3.1-8B				Qwen3-8B			
task	16-bit	4-bit	single	multi	16-bit	4-bit	single	multi
gov_report	22.36	26.72	22.55	23.75	16.57	20.29	20.69	20.14
hotpotqa	33.27	44.14	27.80	37.22	25.02	27.68	30.76	31.23
multifieldqa_en	19.13	35.55	35.40	36.05	22.27	25.47	27.94	26.27
musique	7.63	17.43	11.76	16.53	8.63	10.23	12.06	11.66
narrativeqa	20.73	27.72	21.07	24.71	13.90	17.43	18.46	18.18
passage_retrieval_en	36.50	94.50	35.60	55.50	11.75	53.33	59.27	54.35
qasper	12.15	19.56	26.65	24.77	10.47	12.53	19.79	15.40
qmsum	18.80	22.14	22.56	22.98	17.85	19.07	20.19	19.66
repobench-p	50.59	47.31	31.95	39.03	64.96	63.60	64.90	63.95
samsum	37.00	40.16	38.59	39.55	35.11	36.12	35.58	35.31
trec	23.00	36.50	21.50	22.50	9.00	14.25	14.25	20.75
triviaqa	90.64	89.21	78.53	88.26	89.19	89.49	88.61	87.90
macro	30.98	41.74	31.16	35.90	27.06	32.46	34.38	33.73
tasks above frozen 16-bit	—	10/12	7/12	9/12	—	11/12	10/12	10/12
tasks above frozen 4-bit	—	—	2/12	3/12	—	—	9/12	9/12

The twelve tasks accommodate 36 protected positions at this budget. Short contexts exclude `2wikimqa` and `lcc`; `multi_news` is infeasible even at 4 bits.

Where adaptation transfers. At 3%, Llama's single-format adapter falls below the frozen 4-bit control on ten tasks and improves on `qasper` and `qmsum`. Its largest loss occurs on passage retrieval, where accuracy falls from 94.50 to 35.60; multi-format training raises it to 55.50. Qwen3 improves on nine tasks, ties on `trec`, and declines on `samsum` and `triviaqa`. LongBench passage retrieval asks which paragraph matches an abstract, whereas training asks for a value associated with an explicit key. The shared label "retrieval" therefore covers different operations. The improvement from multi-format training on that task is partial, and Llama's gains on `qasper` and `qmsum` show that question answering and summarization can benefit even as passage retrieval declines. The task-level pattern is consistent with sensitivity to how training questions exercise the cached context.

Transfer across budgets. Every adapter in Table 19 is trained at 3% and reused at the two milder budgets. Mean task scores increase for all configurations. Both adapted recipes remain below the frozen 4-bit control on Llama and above it on Qwen3 at every budget. Llama's single-format loss ranges from 10.58 to 11.87 points; Qwen3's gain ranges from 0.31 to 1.92. Multi-format training improves Llama at every budget but leaves a gap of 4.97 to 5.84 points to that 4-bit control. The model-dependent transfer pattern is therefore present at the training budget as well as at unseen budgets.

Training distribution and retrieval specialization. The broader corpus also changes in-domain recovery. In single-seed standalone evaluations at 3%, Qwen3's full RULER-13 score with KeyDiff decreases from 57.86 to 53.86. With KVzip fixed, multi-format training lowers accuracy from 63.99 to 58.58 on Llama and from 40.72 to 33.02 on Qwen3 (Table 8). Across the three budgets, multi-format training raises Llama's KVzip score at 10% and lowers it at the two tighter budgets;

Table 19: **LongBench transfer across byte budgets.** Macro over the same twelve tasks as Table 18, one seed per adapted configuration. Every adapter is trained at 3%; the milder budgets are unseen during training. The 16-bit and 4-bit columns are frozen controls. The task counts compare with the matched 16-bit control.

				ByteKV, single		ByteKV, multi	
Model	Byte budget	16-bit	4-bit	Macro	Tasks above 16-bit	Macro	Tasks above 16-bit
Llama-3.1-8B	10%	40.79	49.52	37.65	5/12	44.54	9/12
	5%	34.64	44.89	33.88	6/12	39.91	9/12
	3%	30.98	41.74	31.16	7/12	35.90	9/12
Qwen3-8B	10%	32.39	44.65	44.96	10/12	45.87	10/12
	5%	28.58	37.12	38.00	10/12	38.83	11/12
	3%	27.06	32.46	34.38	10/12	33.73	10/12

Table 20: **SCBench semantic transfer at 32K.** The same 351 question–answer pairs per configuration on `scbench_qa_eng`. The 16-bit and 4-bit columns use frozen KeyDiff with matched protection. Single and multi use 4-bit ByteKV-KeyDiff with the corresponding adapters, one seed each. Frozen full-cache scores are 27.92 on Llama and 26.78 on Qwen3.

				ByteKV	
Model	Byte budget	16-bit	4-bit	single	multi
Llama-3.1-8B	10%	18.52	24.79	23.93	24.50
	5%	16.81	21.65	21.65	22.22
	3%	14.81	18.23	17.66	18.80
Qwen3-8B	10%	11.11	22.22	19.37	23.08
	5%	8.55	15.10	14.81	16.24
	3%	8.83	13.68	10.83	12.82

Qwen3's scores decrease throughout. Thus a corpus change that improves some transfer results can reduce retrieval specialization, even with the compressor held fixed.

Distillation constrains the student's predictions on the training questions; it does not reconstruct every original KV vector or train on all possible uses of the context. Answer-format breadth therefore probes one aspect of transfer while leaving the lookup objective unchanged. The corpus runs also change sample counts, training exposure, and the development examples used to choose a checkpoint. The exposure controls in Appendix A.4 show that update count affects full-cache behavior, but do not isolate its effect on compressed LongBench transfer. The present comparison establishes sensitivity to the training recipe rather than a causal effect of answer-format diversity alone.

D.3 SCBench at 32K

SCBench adds context length and reuse beyond the 4K training setting. Its English question-answering task asks multiple questions about a book using one context cache. Single-format ByteKV-KeyDiff exceeds the matched 16-bit control at all three budgets on both models, but matches or falls below frozen 4-bit KV (Table 20). Multi-format training improves all six adapted scores and exceeds the frozen 4-bit control in four cells. Neither synthetic corpus trains book question answering. The multi-format results show that changing the training recipe can improve transfer even without adding that downstream task; format breadth, exposure, and development data change together in this comparison.

Input truncation in the retrieval sub-task. The separate identifier-retrieval task encounters a constraint before KV compression: keeping only the beginning of each input removes many target keys. The target key remains in only 27% of Llama queries and 20% of Qwen3 queries, despite appearing in every untruncated context. The answer is a random 36-character identifier, so this availability constrains exact retrieval. Llama's frozen full-cache score reaches 95% of that ceiling; Qwen3 scores 2.4 against its 20-point ceiling even with its optional reasoning mode disabled. This configuration gives little room for a compressor comparison on Llama and also encounters a base-model limitation on Qwen3. For the semantic sub-task, verbatim answers occur in 43% of truncated contexts and 57% of untruncated contexts. Since semantic answers need not be copied, these fractions are not accuracy ceilings.

Table 21: **Resident prefix KV memory.** Full fp16 versus packed 4-bit ByteKV-KeyDiff at 3%, Llama, batch one, A100-40GB. The generated tail is excluded. Qwen3 and Mistral also reduce storage by a factor of 31.1 to 31.3. Separate batch-one timing uses five warm-ups and twenty decodes.

Context	Full fp16 KV	Packed 4-bit KV	Reduction
4K	0.45 GiB	0.014 GiB	31.1×
8K	0.96 GiB	0.031 GiB	31.2×
16K	1.91 GiB	0.061 GiB	31.3×

E Resident memory and serving cost

At a fixed byte budget, lower precision retains more positions for attention to read. Packed attention and page allocation keep those positions compressed in GPU memory; merging LoRA removes the separate adapter execution path. We measure prefix storage and complete-request capacity separately because question and generated tokens occupy a private, uncompressed tail.

E.1 Packed representation and execution

The serving backend implements the grouped 4-bit quantizer used in training. Within each key or value channel, groups of 128 retained scalars have separate scales and offsets for reconstruction from integer codes. Each compressed page stores packed codes with fp16 scales and zero points. Question and generated tokens occupy separate fp16 tail pages. The prefix therefore keeps its compressed representation as the answer grows. Longer generation budgets require more private tail capacity.

A fused Triton kernel loads the packed prefix, dequantizes tiles in registers, and accumulates attention over both prefix and tail with one online softmax. This removes separate tail gathers, attention launches, and partial-softmax merging without materializing the prefix in fp16. The grouped-query variant reuses loaded KV data across query heads that share a KV head. The implementation retains the quantizer's grouping, selection order, and original-context position convention; the accuracy comparison in Table 23 tests agreement with the simulated operator.

The page allocator and attention backend use the same page layout. A request is charged for its compressed prefix pages and fp16 tail capacity. Charging only a smaller page size while keeping the original position count would miss the capacity gain from selection. Accounting for both makes the storage reduction available to the scheduler. The backend is an external vLLM plugin; the measured feature settings are specified below.

Adapter storage and cache reuse. At inference, unmerged fp16 adapters occupy 104 to 117 MiB for 54.5 to 61.3 million parameters. For the served Llama checkpoint, merging folds LoRA into the base weights and stores the result in fp16, avoiding a separate allocation and execution path for a fixed adapter. Because adaptation participates in prefill, the reusable cache is tied to the weights that produced it. Switching adapters or merged checkpoints requires re-prefill and the corresponding model identity in the cache key.

E.2 Prefix storage and attention work

Packed 4-bit storage at 3% reduces resident prefix KV by a factor of 31.1 to 31.3 across the three architectures (Table 21). Group scales explain the difference from the nominal 33× ratio. At batch one, decode speed ratios range from 0.997 to 1.002 across 4K–16K, consistent with weight operations and kernel launches dominating this workload.

E.3 Serving configurations and capacity

Figure 3 uses a dedicated A100 and closed-loop arrivals, with 8192 context tokens, 1024 generated tokens, greedy decoding, and end-of-sequence stopping disabled. Closed-loop arrivals maintain a target number C of outstanding requests. For $C \in \{1, 8, 32, 64, 96, 128\}$, the client submits $16, 24, 96, 192, 288, 288$ requests, respectively. TTFT includes admission queueing. The adapted full-cache reference and ByteKV-KeyDiff share merged weights and a timing run. The 16-bit and 4-bit controls use the frozen base model and a separate run on another idle A100. The full-cache reference uses standard vLLM attention, chunked prefill, and prefix caching. The custom compressed

Table 22: **The four serving configurations at 64 concurrent requests.** Llama-3.1-8B, 8K context and 1024 generated tokens, as in Figure 3. TTFT is the median; ITL is the 95th percentile. Accuracy references use a separate 4K four-needle evaluation.

Configuration	Ref. accuracy	Output tok/s	TTFT (s)	ITL (ms)	Preemptions
Full cache, adapted	100.00†	397.5	135.4	25.1	29
16-bit KV, frozen	19.10	834.3	9.2	32.8	0
4-bit KV, frozen	50.60	731.3	9.1	45.4	0
ByteKV-KeyDiff	63.45	723.0	9.8	45.4	0

The two adapted configurations share merged weights; the frozen controls were timed in a separate sweep. †Offline full-cache reference of the adapted checkpoint before merging. The corresponding merged full-cache vLLM accuracy is unmeasured. Other accuracy references are offline for the frozen 16-bit and 4-bit controls, and served for ByteKV-KeyDiff. Preemptions count recomputation events, not dropped requests.

backend disables the latter two. All four configurations disable CUDA graph capture. Each curve is one sweep. Table 22 places their measurements at $C = 64$ alongside the separate accuracy references.

ByteKV-KeyDiff reaches 759.8 tokens/s at $C = 96$ and 760.6 at $C = 128$, forming a plateau. The reported $1.911\times$ ratio compares the former point with the adapted full-cache peak of 397.5 at $C = 64$.

Capacity explains the load-dependent benefit. A compressed request occupies seven prefix pages and 32 private fp16-tail pages per layer. With 139264 bytes per page and 32 layers, it uses 165.75 MiB; the full-cache request uses 73 pages of 524288 bytes per layer, or 1168 MiB. The $7.05\times$ request-level reduction is smaller than the prefix-only reduction because the generated tail remains uncompressed. Measured pools hold 3506 compressed or 931 full-size pages per layer, admitting 89 versus 12 concurrent sequences.

Across all six settings, the adapted full-cache reference incurs 146 preemption events; the frozen 16-bit and 4-bit controls and ByteKV-KeyDiff incur none. Preemption releases cache blocks and schedules recomputation while retaining generated tokens. Every configuration completes all 904 requests. At $C = 64$, the full-cache reference completes 192 sequences and 196608 output tokens despite 29 preemptions. All four configurations stay below 50 ms p95 ITL throughout, but exceed the 2 s p95 TTFT bound when $C > 1$. The additional capacity reduces queueing and recomputation without making this closed-loop workload satisfy the full latency target.

Cost at the same prefix byte allowance. The frozen 16-bit control retains 233 positions; the frozen 4-bit control and ByteKV-KeyDiff retain 878 at 4 bits after accounting for group scales. Despite attending to fewer positions, the 16-bit request uses eleven full-size pages per layer, or 176 MiB, 6.2% more than the 4-bit request. Its higher throughput in Table 22 therefore accompanies less attention work, while the 4-bit configurations provide higher accuracy with broader coverage. Because the frozen and adapted configurations were timed in separate runs, their small timing differences do not isolate the execution cost of adaptation.

Merging versus run-time LoRA. An additional implementation comparison runs ByteKV-KeyDiff with a separate run-time adapter. At $C = 8$, p95 ITL is 61.0 ms with run-time LoRA and 31.5 ms with merged weights, measured in separate sweeps. Their 4K four-needle accuracies are 63.65 and 63.45, respectively. This comparison concerns how the same learned update is executed; it is separate from the four configurations in Figure 3.

Longer generation. At 2048 generated tokens, ByteKV-KeyDiff with run-time LoRA reaches 491.9 tokens/s against 402.4 for the frozen full-cache model, a ratio of 1.222 versus 1.370 at length 1024. The private tail grows to give 70 total pages per layer for the compressed request, reducing the request-storage ratio from 7.05 to 4.36. Both configurations now incur preemptions, 161 for frozen full cache and 51 for ByteKV-KeyDiff with run-time LoRA. This longer-generation comparison has not been run with merged weights.

Sharing the compressed prefix. Prefix sharing is measured as pool occupancy at a 7652-token context. With seven shared prefix pages and 32 private tail pages, 32 queries use 1031 pages instead of 1248, a $1.210\times$ reduction. All twelve measurements across six reuse counts match the allocator

prediction. The asymptotic ratio is only $39/32 = 1.219$: after aggressive prefix compression, the private generated tail dominates occupancy.

Batch-output consistency. We compare generated token sequences for 16 prompts with the same configuration's $C = 1$ outputs. At $C \in \{32, 64, 96, 128\}$, 9 of 16 ByteKV-KeyDiff sequences differ, with median first divergence at the second generated token. The frozen 4-bit control has 10 of 16 differing sequences at $C \in \{32, 96, 128\}$, with median first divergence between the 59th and 63rd generated tokens. Repeated $C = 1$ runs reproduce all 16 outputs for both configurations. These observations trigger two conditions of the same consistency check: ByteKV-KeyDiff exceeds 25% differing sequences with median divergence in the first three tokens, while the frozen 4-bit control exceeds 60% differing sequences regardless of where divergence begins.

A separate 256-token diagnostic fixes the GQA kernel across batch sizes. Enabling vLLM's batch-invariant execution mode makes all sixteen compared sequences match their $C = 1$ outputs for both the adapted full-cache reference and ByteKV-KeyDiff at $C = 32$. Kernel controls also agree across batch sizes with a fixed tile configuration. This supports batch-dependent floating-point execution as the source of the observed divergence. Under greedy decoding, small numerical differences can change the next token and thereby change the input to later predictions. Task accuracy at the larger concurrency settings remains unmeasured. The accuracy study below uses a separate workload.

E.4 Agreement of offline and served accuracy

Table 23 follows the implementation from simulated quantization to packed attention and vLLM. All accuracy measurements use a separate 4K slice of four needle families, with 500 examples each. The paired simulated/packed comparison shares the quantizer, selection rule, adapter, and examples. Across five budgets, macro accuracy differs by at most 0.10, with 1998 to 2000 of the 2000 examples scored identically. Its 64-token decode cap removes only post-answer text on this slice (Appendix F).

At the serving budget 2.68%, run-time LoRA scores 63.65 and merged weights 63.45. Offline evaluation with merged weights also scores 63.45, equal task by task. The 0.20 difference from run-time LoRA lies entirely in `niah_multikey_2`. At 3%, vLLM and offline evaluation with fp16 LoRA both score 65.40; the earlier 65.55 reference uses 32-bit floating-point (fp32) LoRA. The other served budget points agree with their offline references within 0.25. These runs measure accuracy through vLLM with compressed storage allocated directly by the attention backend. Paged-memory timing uses a separate workload.

The full-cache references recorded for this checkpoint are 100.00 with LoRA and 99.95 for the frozen model. Figure 3 and Table 22 mark 100.00 as an offline reference before merging; the merged full-cache vLLM accuracy remains unmeasured.

F Full-cache behavior and scoring scope

The complete method is trained and evaluated with compressed caches. This section characterizes the same adapters when compression is disabled and examines whether changes in answer length affect the measured gains under compression.

F.1 Behavior without compression

Table 24 evaluates the 4-bit ByteKV-KeyDiff adapters with compression disabled. Qwen3's changes stay below two percentage points on each accuracy benchmark. Llama improves on ARC and GSM8K, declines on MMLU and HellaSwag, and has higher WikiText perplexity. On full-cache RULER-13, the changes are -4.17 points on Llama and $+0.13$ on Qwen3. The Qwen3 results here use $\lambda = 1$; its $\lambda = 0$ control covers RULER-13 (Appendix A.4).

Task coverage of the full-cache check. The criterion in Appendix A.1 determines checkpoint eligibility on development data. Test-set distortion uses the tolerance as a severity scale, rather than a second eligibility check. The selector controls illustrate why these evaluations serve different purposes. Llama k-norm passes development eligibility but loses 26.5 full-cache points on the

Table 23: **From simulated quantization to served accuracy.** Llama, 4K context, four needle families with 500 examples each. Upper: paired simulated and packed operators; Equal scores counts examples scored identically out of 2000. Lower: serving configurations and budget checks. The frozen 16-bit and 4-bit controls are offline; other lower rows use vLLM. These accuracy runs are separate from the 8K timing workload. MK2/MK3 denote `niah_multikey_2`/`niah_multikey_3`; SN2/SN3 denote `niah_single_2`/`niah_single_3`.

packed against simulated, paired on the same 2000 *examples*				
Byte budget	Simulated	Packed	Δ	Equal scores
9.375%	89.40	89.40	0.00	2000
4.6875%	73.15	73.20	+0.05	1999
3%	65.55	65.55	0.00	2000
2.734375%	63.60	63.60	0.00	2000
2.68%	63.65	63.75	+0.10	1998

4K accuracy references for the serving configurations						
Configuration	Byte budget	MK2	MK3	SN2	SN3	Macro
16-bit KV, frozen, offline	2.84%	0.8	0.0	73.2	2.4	19.10
4-bit KV, frozen, offline	2.68%	2.8	0.0	99.8	99.8	50.60
ByteKV-KeyDiff, run-time LoRA	2.68%	54.6	0.0	100.0	100.0	63.65
ByteKV-KeyDiff, merged LoRA	2.68%	53.8	0.0	100.0	100.0	63.45
run-time LoRA at additional budgets, with offline references						
served, offline fp16 reference 65.40	3%	61.4	0.2	100.0	100.0	65.40
served, offline reference 74.20	5%	90.0	6.4	100.0	100.0	74.10
served, offline reference 91.90	10%	100.0	68.6	100.0	100.0	92.15

Table 24: **Uncompressed use of the 4-bit ByteKV-KeyDiff adapters.** Frozen-to-adapted scores on each complete standard split, seed 0. WikiText is perplexity (lower is better); other suites report accuracy.

Suite	Llama-3.1-8B	Qwen3-8B
	Frozen→Adapted	Frozen→Adapted
WikiText↓ (Merity et al., 2017)	7.97 → 10.38	10.91 → 10.47
MMLU↑ (Hendrycks et al., 2021)	46.6 → 45.4	49.0 → 49.1
ARC-Easy↑ (Clark et al., 2018)	77.2 → 78.9	79.5 → 79.9
HellaSwag↑ (Zellers et al., 2019)	73.3 → 69.2	68.6 → 68.9
GSM8K↑ (Cobbe et al., 2021)	45.0 → 50.6	14.2 → 12.3

four-needle test. LagKV and KVzip show $+0.00$ on that slice but lose 10.46 and 5.06 on full RULER-13.

Changing the trained precision can also affect uncompressed behavior. The 3-bit Llama adapter changes by $+0.00$ on four needles and -15.37 on RULER-13. With the 16-bit Llama diagnostic, four-needle accuracy falls from 99.95 to 72.55. The 27.40-point loss comes entirely from the two multikey families; Qwen3's corresponding change is -0.2. The 2-bit and 8-bit Llama adapters keep their full-cache development score unchanged while losing accuracy on full RULER-13 (Table 10). The table pairs those losses with the same checkpoints' compressed results, separating retrieval through the trained representation from use with compression disabled.

The Llama adapters trained at 8K and 16K show the same task-coverage issue. Their four-needle full-cache changes are $+0.05$ and $+0.00$ points, respectively, while their RULER-13 changes are -25.04 and -22.93. A near-ceiling retrieval slice can detect declines in its own families, but cannot represent unmeasured tasks.

F.2 Answer length and containment scoring

RULER scores recovered spans or accepted QA aliases without penalizing output length. With 4-bit KV at seed 0 and 3%, mean generated length changes from 23.65 to 31.93 tokens on Llama and from 42.37 to 34.68 on Qwen3 after adaptation. The fraction of outputs reaching the token limit rises from 2.9% to 18.5% on Llama but falls from 12.6% to 6.0% on Qwen3. Qwen3's gain therefore accompanies shorter outputs. A minority of long responses raises Llama's mean: its median falls from 21 to 19 tokens, and its task most often reaching the cap gains 0.00.

Truncating outputs at a reference-based length. For each task, we compute the 95th percentile of the token count needed for full credit from the reference answers alone, then truncate both frozen and adapted outputs at twice this count. On the four-needle slice, this removes no credit from the adapted outputs on any of the three architectures. Llama's margin stays at $+14.90$; Qwen3's changes from $+8.70$ to $+9.05$ because some credit is removed from the frozen outputs. The recovery on this slice survives the cap.

On full RULER-13, the cap removes credited content from both configurations. For Llama at 3%, it removes 8.02 points from the frozen 4-bit control and 2.02 from the adapted model, increasing the observed gap from 5.39 to 11.39. Truncation therefore changes which correct answers receive credit. This full-suite control mixes answer placement with retrieval accuracy.

Variation in the capped scores across training seeds. To measure variation under truncation, we rescore all four independently trained adapters with the same reference-based caps. At 3%, the highest and lowest capped RULER-13 scores differ by 1.22 points on Llama and 1.25 on Qwen3. The preregistered consistency check requires this range to be no greater than twice the standard deviation across seeds. The recorded analysis uses two references: the sample standard deviation recomputed after truncation, giving limits of 1.10 and 1.22 points, and the published population standard deviation before truncation, giving 2.58 and 1.10 points, respectively. Llama exceeds the first limit, while Qwen3 exceeds both, so this check remains unmet. Together with the cap's removal of correctly retrieved spans, these results leave length-adjusted gains on full RULER-13 unresolved.

G Related work by comparison axis

Table 25 organizes prior work by the component changed, the budget unit, information used in cache construction, and learning. ByteKV combines an existing query-blind selection rule with byte allocation and adaptation through the resulting sparse, quantized representation.

Table 25: **Comparison axes for cache compression.** ByteKV combines bit allocation and model adaptation with an existing selection rule. The input column describes information used by the cache-construction rule. *Context* denotes context-derived information without a separate question-conditioned score. *Context probes* are query vectors drawn before the downstream question arrives; *prompt probes* may include question tokens. *Synthetic probes* are generated query vectors. *Question* denotes explicit question conditioning, and *online* denotes current or accumulated query signals during decoding. For policy-dependent methods, inputs follow the chosen compression rule. For reuse systems, the column describes stored-module construction; online composition may use the arriving request. Training refers to model or compression-module weights.

Method family	Operation	Budget	Inputs	Training
Context-based KV scoring (Park et al., 2025; Liang et al., 2025; Devoto et al., 2024)	selection	positions	context	none
Sink and recent-position retention (Xiao et al., 2024)	selection	positions	context	none
Online attention-based eviction (Zhang et al., 2023; Oren et al., 2024)	selection	positions	*online*	none
Statistical leverage scoring (base) (Chari & Van Durme, 2025)	selection	positions	context	none
Task-guided scoring (Park & Hwang, 2026)	selection	positions	*task guide*	none
Context-reconstruction scoring (Kim et al., 2025)	selection	positions	context	none
Expected-attention scoring (Devoto et al., 2025)	selection	positions	context	none
Layer-output reconstruction scoring (An et al., 2026)	selection	positions	*prompt probes*	none
Key-space clustering (Yu et al., 2026)	selection	positions	*prompt probes*	none
Retained-cache regression (Peng et al., 2026)	merging	positions	*context probes*	none
Main and residual cache (Zhan et al., 2026)	selection, synthesis	positions	*prompt probes*	none
Query–key geometry (Godey et al., 2025)	selection	positions	context	none

Continued on next page

Table 25 continued

Method family	Operation	Budget	Inputs	Training
Head-budget allocation (Feng et al., 2025)	selection	positions	*prompt probes*	none
Per-input retained count (Tang et al., 2025)	selection	positions	*prompt probes*	none
Risk-aware layer budgets (Feng et al., 2026)	selection	positions	*context probes*	none
Learned retention (Huang et al., 2025; Bui et al., 2026)	selection	positions	context	*yes*
Query-conditioned eviction (Li et al., 2024; Cai et al., 2025)	selection	positions	*question*	none
Query-aware sparse access (Tang et al., 2024)	access selection	pages read; full cache stored	*online*	none
Learned retention with latent memory (Yang et al., 2026)	selection, adaptation	positions + latent vectors	*prompt probes*	*yes*
Learned future-response retention (Ahn et al., 2026)	selection	positions	*question*	*yes*
Quantization (Liu et al., 2024b; Hooper et al., 2024; Chakrabarti et al., 2026)	allocation	bits	context	none
Saliency-based quantization (He et al., 2024)	allocation	bits	*prompt probes*	none
Key-over-value precision (Hariri et al., 2026)	allocation	bits	context	none
Graded per-token fidelity (Esmat et al., 2026)	allocation	*bytes*	*prompt probes*	none
Importance-aware mixed precision (Yang et al., 2024)	allocation	bits	*policy-dependent*	none
Joint retention and precision (Zhang et al., 2026)	selection, allocation	bits	*prompt probes*	none
Per-block bit allocation (Liu et al., 2026)	allocation	bits	context	none
Per-channel bit allocation (Liao & Wen, 2026)	allocation	bits	context	none
Query-driven precision (Zhang et al., 2025b)	allocation	bits	*online*	none
Transform coding (Staniszewski & Łańcucki, 2026)	allocation	*bytes*	context	none
Delta and entropy coding (Liu et al., 2024a)	allocation	*bytes*	context	none
Frequency-domain compression (Kai et al., 2026)	compression, adaptation	positions	context	*yes*
Composite compression (Kang et al., 2024)	allocation	bits + rank + outliers	context	none
Prototype resolution and coverage (Tian et al., 2026)	selection, merging	positions	*question*	none
Synthesised entries (Jiang & Jin, 2026)	synthesis, allocation	positions	*prompt and synthetic probes*	none
Learned selection and rewriting (Chari et al., 2025)	selection, adaptation	positions	context	*yes*
Learned synthetic memories (O'Neill et al., 2026; Kim et al., 2024)	synthesis	positions	context	*yes*
Learned online merging (Nawrot et al., 2024)	merging	positions	context	*yes*
Trained compressibility (Gelberg et al., 2026)	adaptation	positions	context	*yes*
Linear quantization correction (Saxena & Roy, 2025)	adaptation	bits	context	*yes*
Learned restore entries (Baek et al., 2026)	synthesis, adaptation	positions	context	*LoRA*
Adapter for a fixed quantizer (Abdellatif, 2026)	adaptation	bits	context	*LoRA*
Learned position-independent cache reuse (Xie et al., 2026)	adaptation	*uncompressed*	context	*yes*
Learned compressed-cache producer (Du et al., 2026)	adaptation	positions	context	*yes*
Reusable modular caching (Gim et al., 2024; Hu et al., 2025; Yao et al., 2025)	—	*uncompressed*	context	none

Continued on next page

Table 25 continued

Method family	Operation	Budget	Inputs	Training
Trained cache concatenation (Yang et al., 2025b)	—	*uncompressed*	context	*yes*
Rate–distortion survey (Colaco & Lahjouji, 2026)	—	*bytes*	n/a	n/a
Eviction versus quantization at scale (Kim et al., 2026a)	selection vs. allocation	*bytes*	*policy-dependent*	n/a
Compressed cache as a distillation target (Kuzina et al., 2026)	—	positions	*question*	*yes*
Footprint accounting (Bhaskar et al., 2025)	—	positions × lifespan	n/a	n/a
ByteKV (this work)	selection, allocation, adaptation	*bytes*	context	*LoRA*

Selection and allocation. Query access depends on the inputs used for scoring, including any probe tokens. ProtoKV (Yu et al., 2026) forms key-space prototypes and scores their clusters using a recent prompt window; ReST-KV (An et al., 2026) scores eviction by layer-output reconstruction error over a recent prompt window. Such windows can include question tokens once they have been processed. GRKV (Peng et al., 2026) and DefensiveKV (Feng et al., 2026) instead draw their probes from the context before introducing the downstream question. ResKV (Zhan et al., 2026) evaluates both context-only and question-conditioned construction. KVSculpt (Jiang & Jin, 2026) fits its synthetic cache to retained prompt queries and generated future-query probes. TaskPress (Park & Hwang, 2026) supplies an offline task description as a proxy query. Compactor (Chari & Van Durme, 2025) uses statistical leverage for query-agnostic scoring; the table lists its base variant, while the work also includes calibrated and optional LoRA variants. Our within-selector comparisons keep the scoring rule and its access to the question fixed.

Allocation can act across blocks (Liu et al., 2026), heads (Feng et al., 2025; Xiao et al., 2025), channels (Liao & Wen, 2026), or risk-aware layer budgets (Feng et al., 2026). MiKV (Yang et al., 2024) keeps less important entries at low precision instead of discarding them. RDKV (Zhang et al., 2026) jointly assigns eviction and nonzero precisions. It derives its allocation from prefill attention and uses a recent observation window. A query-blind application requires forming these statistics from context positions before the future question is available. GraceKV (Tian et al., 2026) instead allocates prototype resolution and coverage, using probes from the question suffix as well as context regions. ThinKV (Ramachandran et al., 2026) allocates retention and precision during query-conditioned reasoning. SpectralKV (Hariri et al., 2026) studies asymmetric K/V precision, a different intervention from our equal-width, capacity-matched K-versus-V adapter ablation. VarRate (Esmat et al., 2026) uses graded per-token fidelity, while FreqKV (Kai et al., 2026) compresses the cache through a query-blind frequency transform and fine-tunes the reader for that representation.

What is learned. Learning can change the compression module, the cache-producing model, or the model that reads the cache. KV-Distill (Chari et al., 2025) and Still (O'Neill et al., 2026) learn compression modules with a frozen base model. SemPIC (Xie et al., 2026) trains a reusable-cache writer with full-cache KL supervision, and C^2KV (Du et al., 2026) trains an auxiliary cache producer for a frozen reader. KV-CAT (Gelberg et al., 2026) trains the model for compressibility. ByteKV keeps the compressor parameter-free and updates the attention projections used both to produce and to attend to sparse, quantized KV vectors.

KVLinC (Saxena & Roy, 2025) combines key quantization-error feature maps with Hadamard rotation of values. Its linear correction is trained with cross-entropy against full-precision attention weights while retaining the token history. ByteKV instead adapts the attention projections used in prefill and decoding through sparse selection and quantization, using full-cache answer distributions as supervision.

Two independent concurrent approaches study closely related adaptation problems. Low-rank attention adaptation to a fixed quantizer (Abdellatif, 2026) distills a full-cache teacher into a student with quantized KV while retaining every position. It evaluates perplexity and long-context retrieval under the fixed quantized format. RestoreKV (Baek et al., 2026) fixes an evictor and learns supplementary, context-conditioned KV entries within the retained-position budget. LoRA is enabled while gener-

Table 26: GEAR's tested native operating point, full RULER-13, 500 examples per task, one run per model. Its total footprint lies outside the main comparisons' byte-budget range of 3% to 10%.

Model	Byte budget	RULER-13
Llama-3.1-8B	36%	95.5
Qwen3-8B	36%	95.1

ating those entries; context prefill, question processing, and decoding use the base model. ByteKV instead trains the model used for both prefill and decoding through sparse selection and quantization, allocating bytes between position count and precision without adding restore entries.

Reuse and serving. Prompt Cache (Gim et al., 2024) reuses full-size module caches at schema-defined positions; EPIC (Hu et al., 2025) and CacheBlend (Yao et al., 2025) compose full-size chunk caches with selective recomputation. KVLink (Yang et al., 2025b) trains a model and link tokens to read concatenated precomputed caches; an optional extension also compresses those caches. SCBench (Li et al., 2025), EpiCache (Kim et al., 2026b), and Cartridges (Eyuboglu et al., 2026) motivate persistent context representations. CacheGen (Liu et al., 2024a) reduces transfer bytes through delta and entropy coding while retaining positions. UltraQuant (Chakrabarti et al., 2026) demonstrates packed 4-bit serving at 4.25 effective bits per element. KIVI (Liu et al., 2024b) fuses dequantization with matrix multiplication, while ThinKV (Ramachandran et al., 2026) extends paged attention to avoid cache compaction during reasoning. Our backend supports the grouped quantizer used in training, fuses the sparse prefix and generated tail in attention, and allocates pages by their compressed footprint. The serving study measures how the private tail limits both request capacity and the additional benefit of sharing.

Budget scope. Footprint accounting (Bhaskar et al., 2025) and byte-based comparison (Colaco & Lahjouji, 2026) motivate measuring more than a retained-token ratio. Kim et al. (2026a) allocate total memory between model weights and KV for long-output reasoning, finding that eviction-versus-quantization preferences depend on model scale. Our models and weight precision stay fixed. The budget applies to a long-input, query-blind KV prefix.

G.1 GEAR AT ITS NATIVE BUDGET

GEAR (Kang et al., 2024) combines quantization, low-rank error correction, and sparse outlier storage. In our evaluation, its smallest tested representation occupies about 36% of full-cache bytes, outside our range of 3% to 10%. Under the same RULER-13 protocol it is nearly lossless (Table 26), providing a reference at a milder compression level.