KV-KAIZEN: LEARNING CONTEXT-ADAPTIVE CACHE COMPRESSION CHOICES

Anonymous authors
Paper under double-blind review

ABSTRACT

As the context size of text processed with an LLM grows, the size of KV caches can outstrip the memory allocated for the original model weights. This impacts LLM throughput negatively, since decoding is memory-bound and decode cost grows with cache size. Recent work has proposed to alleviate this bottleneck by discarding some of these tokens, keeping the most relevant bits and dropping the rest. *Eviction* introduces a tension, since a one-off decision to discard content may prove detrimental later. Instead, we focus on alternative choices that can lead to cache compression **without** evicting tokens. We achieve this by learning a selector that is able to produce, based on context, a per-layer cache configuration towards an overall compression budget. The selector operates along three axes: sharing one cache across layers (*depth*), caching at fewer bits (*precision*), or truncating the low-rank latent cache representations (*rank*). We call the resulting method KV-Kaizen, for the many small per-layer choices it compounds. We observe that these interventions taken independently and uniformly over all layers limit achievable compression because they degrade accuracy. Crucially, we observe that composing them *locally* and *adaptively* to the context can instead preserve accuracy while achieving large memory savings. At train time, a lightweight selector module is given access to the embeddings of the context sequence, and trained to pick *at each layer* a combination of these three axes, with a language modeling loss that is regularized by a budget constraint satisfaction loss. At inference, the selector runs once, before pre-fill. In evaluations on instruction following and reasoning tasks, our selectors reach the Pareto frontier of accuracy against cache size, against learning-free and *post-hoc* baselines. On long-context tasks, KV-Kaizen improves on eviction and can be composed with it, reaching a $32\times$ smaller decode-time cache on a 14B model while preserving accuracy. A $4\times$ cache size reduction incurs no accuracy degradation from 7B parameters up, and a compressed model is more accurate than a smaller uncompressed one with the same cache size. Together, these findings support pre-training large models and compressing them only afterwards.

1 INTRODUCTION

Attention-based LLMs trade memory for compute: a KV cache stores the key and value (KV) vectors of every prefilled or generated token, so that they are not recomputed at each decoding step. The memory this takes grows with the length and number of the sequences served. `Qwen2.5-14B` (Qwen Team et al., 2025), for instance, uses a cache memory of 192 KiB per token, so serving 32 sequences of 8k tokens fills up 48 GiB, more than the model weights in `bf16`. Naturally, that memory cache introduces a bottleneck: every decoded token reads all of the KV cache, so decoding is limited by memory bandwidth rather than by compute. A smaller cache therefore lets a model serve more and longer sequences, and decode at a higher throughput. The challenge thus becomes caching more efficiently without incurring information loss with respect to the original cache.

Smaller KVs. To alleviate these issues, recent works shrink KV caches by keeping positions that are deemed relevant, and drop the rest (Zhang et al., 2023; Cai et al., 2024; Qin et al., 2025). When such *eviction* strategies are query-independent, they run the risk of sacrificing tokens that could have proved useful for other queries. This failure can be mitigated by dropping KV at inference conditionally on a specific query (Li et al., 2024), or by guiding eviction using a wide-reaching generic prompt (Zhu et al., 2025). Eyuboglu et al. (2026); Zweiger et al. (2026); Monteiro et al. (2026) have

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.

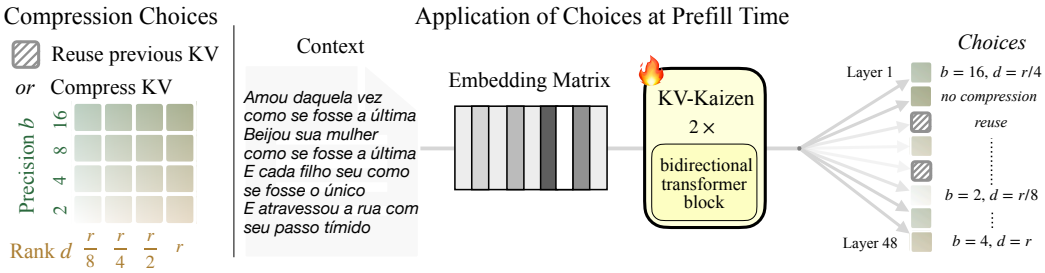

Figure 1: **Per-layer cache selection and composition of compression axes.** The selector chooses among 16 width–precision pairs or, alternatively, a inherit/reuse action per layer. Producing a cache costs $(d + d_R)b$ bits per token; inheritance costs zero. The configuration is fixed during generation, including at prefill time. In theory, the maximal compression factor that could be attained for a 48 layers model such as `Qwen2.5-14B` could be in practice $8\times$ (precision) times 10.7 (lowest rank), and depth 48, by reusing the first KV layer at all subsequent layers (Table 5). This would yield a maximal compression of $4108\times$, though all of our experiments target far lower factors.

also proposed to learn *compact* KV cache representations of a fixed context one attends to repeatedly, by distilling the LLM's use of KV caches against a wide set of synthesized QAs relevant to the targeted context. A natural counterpart to these *compaction/eviction* approaches is cache *compression* without token selection. For instance, one can store the KV cache in low precision to reduce the amount of information that must be moved at every cache read (Hooper et al., 2024; Liu et al., 2024d), or target lower dimensional representations of KV states, as proposed with multi-head latent attention (MLA) (Liu et al., 2024a). Since the KV cache encodes tokens at every head and every layer, cache redundancies have been exploited so that groups of layers share the same KV states rather than producing their own cache (Brandon et al., 2024; Sun et al., 2024; Filippova et al., 2026), as recently showcased in `Gemma-4` (Team et al., 2026) and `DeepSeek-V4.1-Flash` (DeepSeek-AI, 2026). However, applying these structural changes aggressively quickly results in performance degradation.

Learned Adaptive KV Compression Strategies. In this work, we compose three *compression* approaches: MLA's dimension or *rank*, the cache *precision*, and the effective *depth* determined by how layers are grouped to share a cache. We refer to this method as *KV-Kaizen*, in reference to the "kaizen" management method that leverages minor, gradual adjustments to improve an existing pipeline. The savings of these three approaches compound, combining multiple mild compression that yield high compression ratios that would prove too lossy if carried by a single direction. We learn by how much to push along each of precision, rank, and depth during supervised fine-tuning: a small *selector* reads the prompt and picks one option per layer. During inference, the selector runs once at the start of pre-fill, and its choice then holds for every generated token. Since we train selector and model together, the model adapts to the cache it will be served under. We evaluate KV-Kaizen on a range of models covering 1.5B to 32B parameters and different model families, on instruction following, reasoning tasks and long-context retrieval and question answering, against learning-free, *post-hoc* and manually designed baselines at matched cache sizes. Our contributions are:

- We formulate depth, rank, and precision as a single per-layer choice under a shared budget, with all actions expressed in a common cost of bits per token. The budget can also be an input, so that one selector serves several (Section 3.5).
- *Cache size reductions due to rank, precision, and depth compose.* We show that mild settings of the three cache reduction approaches we consider reach compression ratios none reaches alone. For instance, a $4\times$ cache size reduction gives accuracy comparable to the corresponding backbone in the evaluated models of at least 7B parameters. Trained selectors outperform the static and random configurations as shown in Figure 3 (Sections 3.2 and 3.3).
- *Token eviction composes with the non-eviction approaches we consider.* As we do not evict tokens, eviction can be applied to the same cache and the two ratios multiply. On `Qwen2.5-14B` at 16k, a $4\times$ compression due to our cache configuration plus eviction to one position in eight gives a $32\times$ smaller cache at decode time, with no significant accuracy difference (Section 3.4).
- *Cache size reductions realize gains on a real device.* Benchmarking the three approaches on device, we observe that composing them keeps peak memory nearly flat as sequences grow, and that most combinations decode faster than an uncompressed cache (Section 4).

2 COMPOSING COMPRESSION AXES UNDER ONE BUDGET

2.1 BACKGROUND AND NOTATION

We consider a model of L layers and denote by l the layer index. Under grouped-query attention (GQA) (Ainslie et al., 2023), layer l stores K_l and V_l for n_{kv} key-value heads of dimension d_h, so its cache holds $2n_{\text{kv}}d_h$ scalars per token. We consider three cache compression approaches, none of which drops input tokens. **Depth.** Under cross-layer attention (Brandon et al., 2024; Filippova et al., 2026), a layer uses its own query but reads another layer's keys and values, where depth is the number of layers producing their own KV cache. In other words, each layer either produces its own cache, in which case we call it an *anchor*, or inherits from the nearest anchor before it. **Rank.** Multi-head latent attention (MLA) (Liu et al., 2024a) stores one latent representation $c_l^{KV} \in \mathbb{R}^r$ shared across heads, and a decoupled key $k_l^R \in \mathbb{R}^{d_R}$ reserved for RoPE (Su et al., 2024), carrying positional information. A layer holds $r + d_R$ values per token instead of $2n_{\text{kv}}d_h$. Given our focus on methods that enable cache reduction after pre-training, we convert non-MLA models into MLA ones using CARE (Zhou et al., 2026). The *rank* r is the dimension of the MLA latent representation of the KV states, and we can control the cache size by truncating c_l^{KV} to its first $d \leq r$ coordinates, giving an effective latent rank d. This truncation approach works well to the extent that the leading coordinates carry the most information. Training nested representations encourages this ordering (Kusupati et al., 2022), but a converted model does not necessarily do so (Appendix C.2). This can be accounted for during fine-tuning. **Precision.** A cached tensor can be held at $b \leq 16$ bits, by symmetric uniform quantization with one scale per token (Appendix B). During fine-tuning, the cache is quantized and dequantized in the forward pass with the gradient passed straight through, so the model trains against and learns to cope with a b-bit representation of KV states.

2.2 THE SELECTOR ACTION SPACE AND THE CACHE CONFIGURATION COST

We treat depth, rank, and precision as a single per-layer choice (Figure 1), since all three reduce to one quantity: the number of bits a layer writes per token. The selector influences the cache cost via the following *actions*: *inherit* the cache, in which case the layer will simply re-use the most recently produced cache, incurring a cost of 0. If instead a layer produces a cache, then actions correspond to deciding a pair (d, b) of a kept width and a bit-width, with cost

$$c(a) = \begin{cases} 0 & a = \text{inherit}, \\ (d + d_R) \cdot b & a = (d, b), \end{cases} \tag{1}$$

in bits per token, where d and d_R are MLA's kept latent width and positional key width, as discussed in Section 2.1. As a non-MLA backbone has no latent to truncate and no separate key channel, the cost in that case follows with $d = 2n_{\text{kv}}d_h$ and $d_R = 0$, the rank axis is unavailable, and a layer chooses only its bit-width and whether it produces or inherits a cache.

A *plan* $a_{1:L}$, one action per layer, then has one scalar cost, which we report as a compression ratio against the unmodified BF16 non-MLA cache,

$$\rho(a_{1:L}) = \frac{C_0}{\sum_{l=1}^{L} c(a_l)}, \qquad C_0 = L \cdot 2\, n_{\text{kv}} d_h \cdot 16, \tag{2}$$

with n_{kv} KV heads of dimension d_h. Layer 1 must own a cache, so no plan can cost nothing. A GQA layer at $b = 16$ costs exactly C_0/L. An MLA layer at full width already costs less, since it represents the cache in smaller dimensions, and so ρ in that case starts between $1.60\times$ and $1.88\times$ depending on the underlying model before the selector chooses anything, as detailed in Table 5. That part of the compression comes from the conversion and not from the cache configuration and, as such, one must be extra careful when comparing MLA vs. non-MLA cache configurations.

If $\mathcal{B}$ is the bit-widths the precision axis can aim for, $\{2, 4, 8, 16\}$, and $\mathcal{D}$ the kept widths the rank axis can reach, i.e., the model's own r successively halved, $\{r/8, r/4, r/2, r\}$, then the action space is

$$\mathcal{A} = \{\text{inherit}\} \cup \{\,(d, b) : d \in \mathcal{D},\ b \in \mathcal{B}\,\}, \qquad |\mathcal{A}| = |\mathcal{D}| \cdot |\mathcal{B}| + 1. \tag{3}$$

That gives the selector 17 actions per layer as illustrated in Figure 1. Restricting the action space to a subset of these three approaches recovers the corresponding single-axis methods since *precision* is per-layer cache quantization, *rank* is per-layer rank truncation, and *depth* is learned cross-layer sharing. Selectors operating on pairs of these approaches can also be defined by simply setting $\mathcal{B} = \{16\}$ or $\mathcal{D} = \{r\}$. Table 5 shows the cache reduction factors each such subset of $\mathcal{A}$ can reach.

2.3 LEARNING THE CACHE CONFIGURATION AGAINST A BUDGET

A selector $s : \mathcal{V}^T \to \mathcal{A}^L$ maps T input tokens from the vocabulary $\mathcal{V}$ into per-layer cache configurations, and learns to do so while the backbone trains. We parameterize the selector with one linear head per layer, each mapping the shared feature to logits over the action space $\mathcal{A}$. When depth is not within the action space of the selector being trained, its logit is forced to $-\infty$, which includes layer 1. Selection uses Gumbel-softmax (Jang et al., 2016; Maddison et al., 2016) with a straight-through estimator (Bengio et al., 2013), hard $\arg\max$ forward and softmax backward at a temperature annealed from 2.0 to 0.1, and a plain $\arg\max$ at evaluation. The heads are initialized so that every layer's $\arg\max$ is the least-compressed action, so training starts from the uncompressed model and departs from it as training progresses.

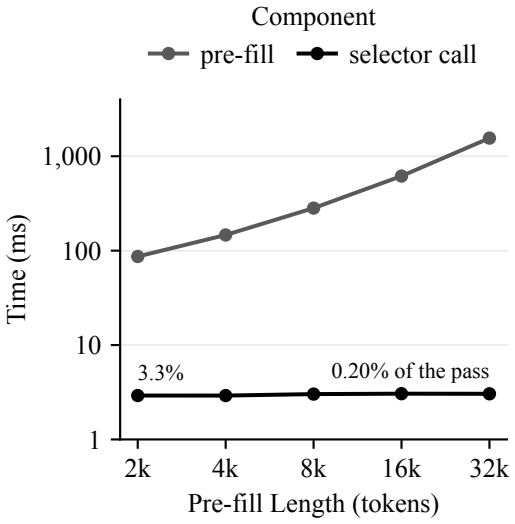

Figure 2: **Selector and pre-fill latency versus input length.** Measurements use `Qwen2.5-7B` at $4\times$ compression, batch size 1, on one H100. The selector takes about 3 ms, or 3.3%–0.20% of pre-fill latency.

The shared feature comes from a small model that takes the prompt as an input. It is made of two bidirectional transformer blocks that ingest the backbone's own token embeddings. The representations are later averaged along the time axis. The evaluated selectors range from 2.6 to 3.7M parameters, which represents less than 0.05% of the model from 7B upward. This sequence model shared across layers runs once at pre-fill, as depicted in Figure 2 where the selector takes about 3 ms, from 2k to 32k input tokens, representing a share ranging from 3.3% to 0.20% in time to first token, while the cache reduction applies at every memory-bound decode step. The plan is then held for the whole generation, so the cache shape is stable across decode steps and the selector costs nothing per decoded token. In our implementation, training with the selector takes 1.0 to $3.1\times$ as long as a plain fine-tune (Appendix F.10). The selector is trained jointly with the backbone under the language modeling loss plus a term pushing realized cost toward a target compression $\rho_\star$:

$$\mathcal{L}_{\text{rate}} = \beta \cdot \mathbb{E}_i \left| \frac{1}{\rho_i} - \frac{1}{\rho_\star} \right|, \qquad \frac{1}{\rho_i} = \frac{1}{C_0} \sum_l \sum_{a \in \mathcal{A}} w_{l,a}^{(i)}\, c(a), \tag{4}$$

$$\mathcal{A} = \underbrace{\{\text{reuse}\}}_{\text{depth}} \cup \Big(\underbrace{\{\tfrac{1}{8}, \tfrac{1}{4}, \tfrac{1}{2}, 1\}}_{\text{rank (fraction kept)}} \times \underbrace{\{2, 4, 8, 16\}}_{\text{precision (bits)}} \Big)$$

for input sequence i, where $w_l^{(i)}$ is layer l's straight-through one-hot selection vector and $c(a)$ is a lookup cost, so that $1/\rho_i$ evaluates the total cost of the plan cached. We introduce a hyperparameter β that must be tuned, and its best value depends both on the backbone and $\rho_\star$. Sweeping it for every configuration is expensive, so we adapt it during training instead. We raise β while the requested compression is missed and relax it once it is met. This works well enough that one setting reaches every fixed target we train for, which is what lets us compare a grid of configurations. We discuss the β adaptive rule in full in Appendix D, and compares it against a swept constant.

In our evaluations discussed in Section 3, we compare learned selectors with static cache configurations *at a matched budget and over the same action space* (Appendix B). Similarly, we consider picking the cache configuration at random out of a set of options that yield the desired target cache size. In all cases, models are fine-tuned with the cache configuration already in place, under exactly the recipe of the learned configurations, since applying a configuration to a model that never trained under it degrades it systematically (Table 10).

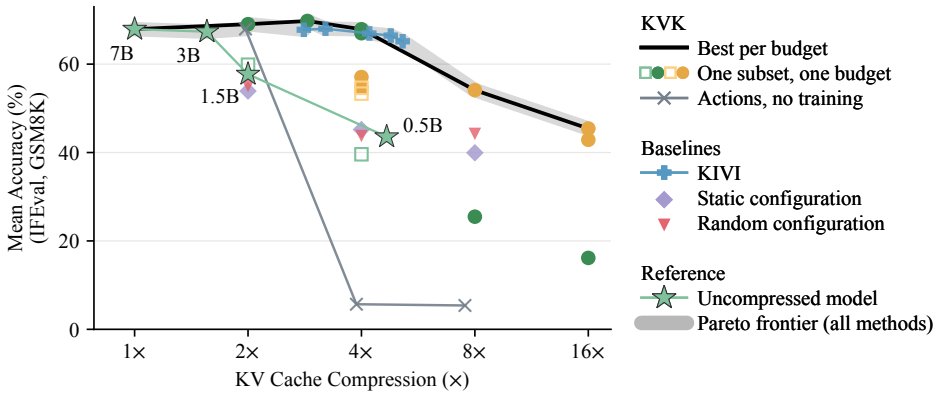

Figure 3: **Accuracy versus cache compression on `Qwen2.5-7B`.** Accuracy is the mean of IFEval and GSM8K; compression is relative to the full 7B BF16 cache. The starred curve shows uncompressed models at four sizes. The black curve is the best KVK run per budget; the pale band gives the Pareto frontier of all methods shown. Static and random configurations are fine-tuned. KIVI: grouped, BF16 residual.

3 Evaluation

3.1 Setup

We reduce the cache size of already trained models by fine-tuning them along with the KV-Kaizen selector. The base models we start from are listed in Table 4 in Appendix B along their respective attention patterns. For configurations that use the rank axis we rely on MLA backbones, converted from the original model with CARE (Zhou et al., 2026). For all models we use the same SFT recipe: the Dolci instruction mixture (Olmo Team et al., 2025) at length 4096 for 270,000 steps, with the selector and the backbone trained together. Appendix B gives the optimizer settings, the conversion settings and the selector architecture in full. We report accuracy on an instruction following task and a reasoning task, IFEval (Zhou et al., 2023) and GSM8K (Cobbe et al., 2021), both through the lm-evaluation-harness (Gao et al., 2024), and we evaluate longer contexts on RULER (Hsieh et al., 2024) at 16k. We implement all baselines and compare realized cache reduction factors, matching cache sizes where available and stating both factors otherwise. The cost model in Equations (1) and (2) excludes quantization metadata; Appendix C reports its storage overhead.

3.2 Reducing the cache size by more than about 5× needs training

Figure 3 compares mean IFEval and GSM8K accuracy against cache compression on `Qwen2.5-7B`. For example, a factor of 2× corresponds to half the cache size of the uncompressed model. Remarkably, at a cache reduction factor of 4×, the KVK selector reaches the same accuracy as the uncompressed control. **Simple baselines**. The static configuration (the same action applied uniformly to every layer) and the random configuration (one action drawn at random per layer) both degrade accuracy, and fall below the frontier in Figure 3 at the same cache size. In Table 15 we separately compare *learned*, *evenly spaced*, and *random anchor* placement at a fixed cache size. In Appendix E we compare the learned configuration with 24 alternatives at similar cache cost, while holding model weights fixed. **Search baseline**. A cache configuration can also be found by search rather than learned. In Appendix E we compare an offline dynamic program over the same action space against the uniform and random configurations, on held-out calibration data at a similar cache reduction factor of 8×. The program's configuration raises the language modeling loss by 0.030, while the uniform configuration raises it by 4.00 and the best out of 64 random configurations raises it by 1.25. At fixed cache size, the loss depends on the configuration chosen.

The model must be train under the configuration that will be used for serving at inference time. Applying the configuration of a learned selector to a frozen backbone collapses accuracy: we see those measurements sitting at the bottom of Figure 3. *Post-hoc* quantization can work with a more

Table 1: **Mean IFEval and GSM8K accuracy differences at $4\times$ cache reduction.** The 3B, 7B, and 14B columns use `Qwen2.5`; Ref. identifies the size-matched control (MLA for rank, non-MLA otherwise). The first row gives absolute accuracies; subsequent rows give differences from Ref., with higher values better. Bold marks differences ≥ -0.053, ie the largest observed seed range (Table 16).

		Mean accuracy / difference ↑				
		Qwen2.5			*Other families*	
Approaches	Ref.	3B	7B	14B	Mistral-7B	OLMo-3-7B
Uncompressed control	–	0.673	0.679	0.723	0.349	0.508
Precision	Non-MLA	**−0.041**	**−0.009**	**+0.040**	**+0.079**	**+0.011**
Depth	Non-MLA	−0.354	−0.283	−0.147	−0.092	−0.126
Precision + depth	Non-MLA	−0.063	**−0.000**	**+0.031**	**+0.068**	**+0.015**
MLA conversion only	Non-MLA	−0.178	−0.105	−0.091	**+0.005**	**−0.040**
Rank	MLA	−0.219	**−0.028**	**−0.005**	**−0.007**	**+0.006**
Precision + rank	MLA	−0.167	**−0.003**	**+0.043**	**−0.001**	**+0.009**
Depth + rank	MLA	−0.224	**−0.041**	**−0.008**	**−0.033**	**+0.020**
All three	MLA	−0.169	**−0.028**	**+0.021**	**+0.027**	**+0.000**

accurate quantizer: a KIVI-style scheme (Liu et al., 2024d) applied after training stays on the frontier and reaches a cache reduction factor of about $5\times$, with a limit of two bits per stored entry. Beyond that compression factor of $5\times$, all configurations on the frontier are the learned ones. We report the individual numbers in Table 10 in Appendix F.

We compare compression by serving smaller models (Figure 3). The starred curve shows `Qwen2.5` models at 0.5B, 1.5B, 3B and 7B, reporting their *uncompressed* KV-cache (of lower dimension) against the 7B reference. Past a factor of $2\times$ these models score below the Pareto front, making a compressed KV-cache of a 7B model more accurate than the full KV-cache of smaller alternatives.

3.3 Scaling the base model size

We fix the base model at 7B in Figure 3. We sweep the size of the base model, and the set of cache reduction approaches the selector is allowed to use. We also add two other model families, `Mistral-7B` (Jiang et al., 2023) and `OLMo-3-7B` (Olmo Team et al., 2025), so that the results do not rest on `Qwen2.5` alone. In Table 1 we look at the accuracy gap against the uncompressed control in the same attention class (higher is better ↑). That is, MLA models are compared against an MLA control fine-tuned with the same recipe and no compression, and non-MLA models against a non-MLA control. The gap closes as the base model grows. At 3B every configuration sits behind its control, and at 7B and 14B depth on its own is the only one that is not level with its control or ahead of it. Table 17 in Appendix F.6 carries the same comparison out to 32B, and also reads it against the model we start from rather than against our own control, which gives the same picture.

We also note that combining approaches does not add their costs together. On the non-MLA backbone, precision on its own and precision combined with depth reach similar gaps at every size, while depth on its own, which has to let three layers in four inherit to reach $4\times$, is far behind both. At 14B this is a comparison at one cache size: with only depth available the selector loses accuracy at a factor of $4\times$, where precision and depth together at that same cache size land above the control. Allowing several approaches lets the selector meet the budget without relying on aggressive depth compression. Table 11 in Appendix F breaks the same configurations down per task at 7B, and depth is observed to cost more on reasoning than on instruction following. Figure 4 reports the same configurations at 14B, per task. Beyond $8\times$, selector settings unable to act on rank lose reasoning accuracy first. Depth on its own at $16\times$ still reaches 0.444 on instruction following, but only 0.056 on reasoning.

3.4 Long context evaluations and composition with eviction methods

In Table 2 we reports RULER at 16k on `Qwen2.5-14B`. We use this setting for two things: to compare the approaches we consider against token eviction methods such as SnapKV (Li et al., 2024) and PyramidKV (Cai et al., 2024), and to combine the two. The retrieval tasks can not discriminate

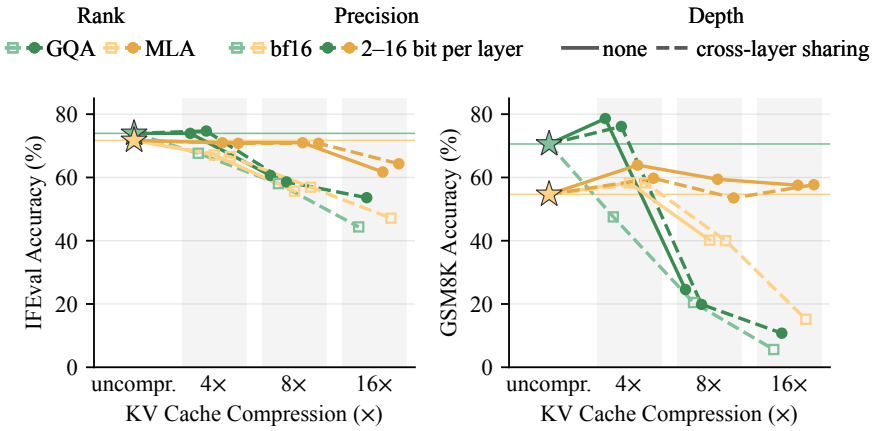

Figure 4: **IFEval and GSM8K accuracy versus realized cache compression on `Qwen2.5-14B`.** Each curve uses a different subset of compression axes: hue marks rank, shade and marker mark precision, and a dashed line marks depth. Horizontal lines mark the GQA and MLA controls.

Table 2: **RULER accuracy at 16k on `Qwen2.5-14B`.** NIAH multi-key and QA report retrieval and question-answering accuracy over 275 documents each. ρ is the decode-time cache reduction relative to the uncompressed GQA model; keep is the retained token fraction, and G the quantization group size. Parentheses give QA differences from the SFT control. Single-needle accuracy is 1.000 throughout and therefore omitted from the table.

		Accuracy ↑	
Configuration	ρ ↑	NIAH multi-key	QA (SQuAD)
Untuned, before our recipe	1.0	1.000	0.592 (+0.049)
Uncompressed control	1.0	0.956	0.543
KVK, 4× (precision + depth)	4.0	0.967	0.614 (+0.071)
+ **SnapKV**, keep 0.25	16.0	0.967	0.560 (+0.017)
+ **SnapKV**, keep 0.125	32.0	0.967	0.531 (−0.012)
KIVI 4-bit G=128	3.7	0.971	0.515 (−0.028)
+ SnapKV, keep 0.125	29.5	0.949	0.464 (−0.079)
SnapKV, keep 0.25	4.0	0.935	0.501 (−0.042)
SnapKV, keep 0.125	8.0	0.916	0.478 (−0.064)
PyramidKV, keep 0.25	4.0	0.949	0.485 (−0.057)

the configurations, since every method solve single-needle (100%) while the multi-key scores remain similar: therefore we focus on QA. At a cache reduction factor of 4×, our selector reaches 0.614 on question answering, against 0.501 for SnapKV (keeping one position in four) and 0.485 for PyramidKV (at the same cache size). Table 12 in Appendix F shows similar results for a 7B model.

As we do not drop tokens, an evictor can run on the same cache and the cache reduction factors due to each approach multiply. SnapKV at one position in four combined with our 4× configuration gives a cache reduction factor of 16× at an accuracy of 0.560, and at one position in eight the factor reaches 32× at 0.531. Combining eviction with a *post-hoc* quantizer instead is worse: four-bit KIVI under the same evictor holds a larger cache, 29.5× against our 32×, and reaches a lower accuracy of 0.464.

3.5 One selector covers a set of budgets

Experiments so far train a selector to reach a single target compression factor $\rho_\star$. To let one selector serve more than one cache size, we make the target an input to the selector, through a zero-initialized FiLM layer (Perez et al., 2018) on the pooled feature, and training is so that each input sequence

Table 3: **Mean IFEval and GSM8K accuracy with budget-conditioned selectors on `Qwen2.5-7B`.** $\rho_\star$ is the requested cache reduction factor. Against $\{1, 2, 4\}$, the uncompressed control scores 0.679 at $1\times$, and single-target selectors 0.690 at $2\times$ (precision only) and 0.679 at $4\times$. With all three approaches the MLA control scores 0.574; requests are met within 2% except $1.94\times$, $1.78\times$ and $1.90\times$ at $2\times$, in row order, and $3.81\times$ for the 540k selector at $4\times$.

			Mean accuracy ↑ at requested $\rho_\star$			
Approaches	Training	Steps	$1\times$	$2\times$	$4\times$	$8\times$
Precision + depth	$\{1, 2, 4\}$	270k	**0.703**	**0.659**	**0.612**	–
All three	One per target	270k each	–	0.590	0.547	0.541
All three	$[2, 8]$	270k	–	0.526	0.487	0.475
All three	$[2, 8]$	540k	–	**0.553**	**0.524**	**0.502**

draws its own target. Trained on $\{1, 2, 4\}$ with precision and depth, one selector covers all three, ahead of the uncompressed control at $1\times$ and 0.067 behind a single-target selector at $4\times$ (Table 3).

A wider compression range requires more training. In Table 3 we compare the selectors trained over $[2, 8]$ with all three approaches, against one selector per target. Trained for twice as long, a single selector trails them by 0.038, 0.023 and 0.039 at $2\times$, $4\times$ and $8\times$, for only two thirds of the steps the three single-target selectors would take. Wider sets of targets do worse: they stop following the request and prevent the rate multiplier from settling (Appendix F.7).

3.6 Cache configuration diversity

We benchmark the per-layer choices of selectors trained at 3B, 7B, and 14B and at several target compression factors, on sample prompts from IFEval, GSM8K, and RULER of up to 16k tokens (Appendix F.9). Selectors adapt the configuration to the context, but some configurations are shared across prompts, so a selector returns fewer distinct configurations than the prompts it sees. Configurations also change from one dataset to another, and a cache configuration found on one dataset need not suit another. We observed that some selectors even return a global configuration that realizes $\rho_\star$ on every prompt, so they could be dropped at test time. Whether a selector can be replaced by a single configuration therefore depends on the selector and has to be checked, ideally on representative test data. The selector only costs about 3 ms (Figure 2), negligible compared to pre-fill.

We also note that rank, precision, and depth are not interchangeable. Precision and rank change a layer by degrees, a coarser quantizer holding the same values less exactly and a narrower latent keeping the leading coordinates of a wider one, so the layer goes on computing what it computed before. Depth has no setting in between: a layer that gives up its cache works from keys and values it did not produce. That is the larger change, and the selector uses it least as a consequence. Depth does however enable higher compression factors. Precision cannot pass $8\times$ (its two-bit floor), and rank knob is unavailable on non-MLA models, so every configuration we train past $8\times$ on a non-MLA backbone must give up at least half of its layers, and none could reach that factor otherwise.

4 Benchmarking cache compression approaches

We benchmark memory and speed on Apple M3 Ultra with 512GB of RAM. We run `Qwen2.5-14B` with 4-bit weights at batch size 1, with the same approach on every layer: a 4-bit cache ($3.56\times$), an MLA latent of rank 128 ($10.7\times$), and 12 groups of 4 layers sharing one cache ($4\times$): a total of $152\times$ for all three. The 4-bit cache uses the grouped quantizer of MLX (Hannun et al., 2023).

We study two synthetic scenarios: we sweep over context lengths to measure the impact on pre-filling (lower is better ↓) and we sweep over completions lengths measure decoding throughput (higher is better ↑). A BF16 cache adds 12.3 GB of peak memory and the composed one 0.17 GB (Figure 5, right). The BF16 baseline, which is the most expensive in memory, is also the fastest at pre-filling, a compute-intensive operation. The *depth* knob always help in throughput. *MLA* mostly pays off at long sequences lengths. The 4-bit KV-cache is more compute-intensive due to de-quantization cost, which increases the cost of pre-filling and MLA, but pays off at long horizons.

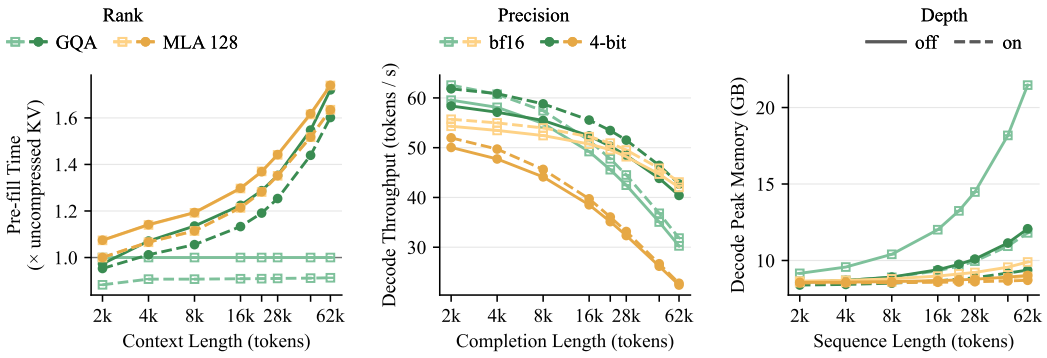

Figure 5: **Pre-fill time, decode throughput and peak decode memory on an Apple M3 Ultra.** `Qwen2.5-14B` with 4-bit weights at batch size 1. Hue marks GQA or an MLA latent of rank 128 (rank), shade and marker a BF16 or 4-bit cache (precision), and a dashed line 12 groups of 4 layers sharing a cache (depth). Pre-fill time is reported relative to the uncompressed cache in bf16 precision.

5 Related work

Most recent work focuses on dropping or merging token positions by attention score or recency (Xiao et al., 2024; Li et al., 2024; Zhang et al., 2023; Cai et al., 2024; Qin et al., 2025; Shen et al., 2025; Ge et al., 2023). An orthogonal set of approaches makes each position it keeps cheaper, and this is where the three cache reduction strategies we use come from: sharing KV states within a layer (Shazeer, 2019; Ainslie et al., 2023) or between layers (Brandon et al., 2024; Liu et al., 2024c; Yang et al., 2024; Wang et al., 2025a; Mu et al., 2024; Wu & Tu, 2024; Monteiro et al., 2024; Wu et al., 2025; Filippova et al., 2026), replacing them with a low-rank latent (Liu et al., 2024a;b; Meng et al., 2025; Zhou et al., 2026), or holding each entry at fewer bits (Hooper et al., 2024; Liu et al., 2024d). In most of these non-evicting approaches, the cache design is fixed at pre-training. Non-uniform allocation across layers is not new (Cai et al., 2024; Qin et al., 2025; Zaccone et al., 2026; Feng et al., 2026; Wang et al., 2025b; Chang et al., 2024), nor is a compression target in the objective (Nawrot et al., 2024), and closest to us Filippova et al. (2026) randomize the source layer during fine-tuning so one checkpoint tolerates sharing patterns chosen at test time. Prior work has looked into learned allocation of non-evicting cache compression. For instance, Bhatnagar et al. (2026) choose a rank per head and per block with a differentiable threshold and quantize the result, and Lin et al. (2025) train nested projections and then pick a rate per layer and head for a budget set after training. Both allocate within a narrower action space than ours, so a given cache reduction factor has to be reached by pushing fewer mechanisms further, where we claim a wider space lets the selector combine mild settings that cost less accuracy. A detailed account of relevant literature is reported in Appendix A.

6 Conclusions, limitations, and future work

We introduced KV-Kaizen, which learns per-layer *depth*, *rank*, and *precision* choices under one cache budget during SFT. Combining these approaches enables significant KV cache size reductions at little to no accuracy degradation. Moreover, at 16k context length, combining 4× representation compression with token eviction gives a 32× smaller decode-time cache, with significant accuracy difference from the uncompressed control for a 14B parameters model. We observed larger models to generally tolerate compression better, favoring the training of larger models with compression prior to deployment. We also noted that a budget-conditioned selector can serve several compression targets, although at a larger training cost. Some selectors vary with the input and between datasets, while others return the same configuration across prompts.

Fully realizing the memory and speed benefits of learned configurations, which mix widths and bit-widths across layers, requires specialized kernels. Rank compression also needs MLA conversion, whose accuracy cost varies across the evaluated models. Budget conditioning depends on the training targets: widening their range can reduce accuracy even at the less compressed settings. Finer control of the budget could let one model adapt to each deployment's hardware and accuracy needs.

Reproducibility statement

All models, data and benchmarks we use are public: the backbones in Table 4, the Dolci instruction mixture, and IFEval, GSM8K and RULER through the lm-evaluation-harness. Section 2 defines the method, its cost model and the training objective. Appendix B gives the fine-tuning recipe, the conversion to MLA, the selector's architecture, and the evaluation and timing settings; Appendix C gives the action space and how each action is stored, and Appendix D the rate multiplier's update. Appendix F reports variation across training seeds, and Section 4 states the device and settings of the on-device measurements.

AI use statement

In this work, we used AI coding agents to help develop, review and debug parts of the research code. We iterated on the method with the help of AI agents, which we also used to check and refine our hypotheses, to give feedback on our proposals and experiments, and to help validate and analyze results. We did not use generative AI tools to generate, clean or reformat data sets, to formulate mathematical claims or to write proofs, and translation and qualitative data analysis are not applicable to this work. Additionally, we used AI tools to refine the writing of the manuscript and to prepare figures. We have reviewed all AI-assisted work, and every reported result comes from a logged training or evaluation run. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

Ethics statement

This work uses publicly released models and benchmarks under their licenses, with no new data collection or human subjects. Reducing inference cost may also lower the cost of harmful uses of language models. Compression affects tasks differently, and our evaluations do not establish its effects across all languages or domains. Deployment therefore requires evaluation under the intended tasks and compression settings. We have adhered to the ICLR Code of Ethics.

References

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.

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

Priyansh Bhatnagar, Ashkan Moradifirouzabadi, Se-Hyun Yang, SeungJae Lee, Jungwook Choi, and Mingu Kang. Star-kv: Low-rank kv cache compression via soft thresholding for adaptive rank control. *arXiv preprint arXiv:2606.08382*, 2026.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv:2405.12981*, 2024.

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

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S Abdelfattah, and Kai-Chiang Wu. Palu: Compressing kv-cache with low-rank projection. *arXiv preprint arXiv:2407.21118*, 2024.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, Xilai Dai, Huiqiang Jiang, Yucheng Li, Luis Ceze, Kai-Chiang Wu, et al. xkv: Cross-layer kv-cache compression via aligned singular vector extraction. *arXiv preprint arXiv:2503.18893*, 2025.

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

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Y Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations*, volume 2026, 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. *Advances in Neural Information Processing Systems*, 38:113152–113188, 2026.

Anastasiia Filippova, David Grangier, Marco Cuturi, and Joao Monteiro. Stochastic kv routing: Enabling adaptive depth-wise cache sharing. *arXiv preprint arXiv:2604.22782*, 2026.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, July 2024. URL https://zenodo.org/records/12608602.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Awni Hannun, Jagrit Digani, Angelos Katharopoulos, and Ronan Collobert. MLX: Efficient and flexible machine learning on Apple silicon, 2023. URL https://github.com/ml-explore/mlx.

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

Coleman Hooper, Sebastian Zhao, Luca Manolache, Sehoon Kim, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Multipole attention for efficient long context reasoning. *arXiv preprint arXiv:2506.13059*, 2025.

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

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

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. *arXiv preprint arXiv:2505.23416*, 2025.

Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

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

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

Bokai Lin, Zihao Zeng, Zipeng Xiao, Siqi Kou, Tianqi Hou, Xiaofeng Gao, Hao Zhang, and Zhijie Deng. Matryoshkakv: Adaptive kv compression via trainable orthogonal projection. In *International Conference on Learning Representations*, volume 2025, pp. 86669–86690, 2025.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

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

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024c.

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 Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR, 21–27 Jul 2024d. URL https://proceedings.mlr.press/v235/liu24bz.html.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Fanxu Meng, Pingzhi Tang, Xiaojuan Tang, Zengwei Yao, Xing Sun, and Muhan Zhang. Transmla: Multi-head latent attention is all you need. *arXiv preprint arXiv:2502.07864*, 2025.

Francesco Maria Molfese, Momchil Hardalov, Rexhina Blloshmi, Bill Byrne, and Adrià de Gispert. Exploring fine-tuning for in-context retrieval and efficient kv-caching in long-context language models. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 617–635, 2026.

Joao Monteiro, Étienne Marcotte, Pierre-Andre Noel, Valentina Zantedeschi, David Vazquez, Nicolas Chapados, Christopher Pal, and Perouz Taslakian. Xc-cache: Cross-attending to cached context for efficient llm inference. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15284–15302, 2024.

João Monteiro, Michal Klein, Pierre Ablin, and Marco Cuturi. Nectar: Neural estimation of cached-token attention via regression. *arXiv preprint arXiv:2605.09778*, 2026.

Luca Moschella, Laura Manduchi, and Ozan Sener. Learning to evict from key-value cache. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=0OevIlRMYN.

Junlin Mu, Hantao Huang, Jihang Zhang, Minghui Yu, Tao Wang, and Yidong Li. Sals: Sparse attention in latent space for kv cache compression. *Advances in Neural Information Processing Systems*, 38:384–403, 2026.

Yongyu Mu, Yuzhang Wu, Yuchun Fan, Chenglong Wang, Hengyu Li, Qiaozhi He, Murun Yang, Tong Xiao, and Jingbo Zhu. Cross-layer attention sharing for large language models. *arXiv preprint arXiv:2408.01890*, 2024.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M Ponti. Dynamic memory compression: Retrofitting llms for accelerated inference. *arXiv preprint arXiv:2403.09636*, 2024.

Olmo Team, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. Cake: Cascading and adaptive kv cache eviction with layer preferences. In *The Thirteenth International Conference on Learning Representations*, 2025.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Yiqun Shen, Song Yuan, Zhengze Zhang, Xiaoliang Wang, Daxin Jiang, and Cam-Tu Nguyen. Lava: Layer-wise kv cache eviction with dynamic budget allocation, 2025.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361, 2024.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Yixuan Wang, Haoyu Qiao, Lujun Li, Qingfu Zhu, and Wanxiang Che. Commonkv: Compressing kv cache with cross-layer parameter sharing. *arXiv preprint arXiv:2508.16134*, 2025a.

Zihao Wang, Bin Cui, and Shaoduo Gan. Squeezeattention: 2d management of kv-cache in llm inference via layer-wise optimal budget. In *International Conference on Learning Representations*, pp. 23593–23606, 2025b.

Xiuying Wei and Caglar Gulcehre. Rat+: Train dense, infer sparse–recurrence augmented attention for dilated inference. *arXiv preprint arXiv:2602.18196*, 2026.

Haoyi Wu and Kewei Tu. Layer-condensed kv cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11175–11188, 2024.

You Wu, Haoyi Wu, and Kewei Tu. A systematic study of cross-layer kv sharing for efficient llm inference. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 396–403, 2025.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

Yuhui Xu, Zhanming Jie, Hanze Dong, Lei Wang, Xudong Lu, Aojun Zhou, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Think: Thinner key cache by query-driven pruning. In *International Conference on Learning Representations*, volume 2025, pp. 56691–56709, 2025.

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*, 2024.

Riccardo Zaccone, Stefanos Laskaridis, Marco Ciccone, and Samuel Horváth. Flexrank: Nested low-rank knowledge decomposition for adaptive model deployment. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=DK0kvnNelx.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Zhongzhu Zhou, Fengxiang Bie, Ziyan Chen, Zhenyu Zhang, Yibo Yang, Junxiong Wang, Ben Athiwaratkun, Xiaoxia Shirley Wu, and Shuaiwen Song. Care: Covariance-aware and rank-enhanced decomposition for enabling multi-head latent attention. In *International Conference on Learning Representations*, pp. 43245–43278, 2026.

Yuanbing Zhu, Zhenheng Tang, Xiang Liu, Ang Li, Bo Li, Xiaowen Chu, and Bo Han. OracleKV: Oracle guidance for question-independent KV cache compression. In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025. URL https://openreview.net/forum?id=KHM2YOGgX9.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via attention matching. In *International Conference on Machine Learning (ICML)*, 2026.

APPENDIX CONTENTS

A Extended related work **16**

B Experimental details **17**

C Action space and cache storage **18**

C.1 Compression ranges 18

C.2 MLA cache representation 19

C.3 Quantization and storage overhead 19

D Setting the rate multiplier β **20**

D.1 Update rule 20

D.2 Budget accuracy and task performance 20

E Comparing cache configurations at a fixed budget **21**

E.1 Alternatives to the learned configuration 21

E.2 Offline search 21

F Additional experimental results **22**

F.1 Quantization and smaller-model baselines 22

F.2 Accuracy by compression axis 23

F.3 Long-context evaluation 23

F.4 Effect of fine-tuning 24

F.5 Learned and static cache configurations 25

F.6 Training seeds and model size 26

F.7 Discrete and continuous budget conditioning 26

F.8 Use of cross-layer sharing 27

F.9 Cache configuration diversity 27

F.10 Selector architecture 28

G On Device benchmarks **28**

G.1 The Role of Precision 29

A Extended related work

Temporal eviction and selection. Most recent work drops or merges tokens along the time axis. StreamingLLM (Xiao et al., 2024) keeps a sliding window plus attention sinks; SnapKV (Li et al., 2024) uses attention scores to retain the chunks a prompt attends to; PyramidKV (Cai et al., 2024), CAKE (Qin et al., 2025) and LAVa (Shen et al., 2025) vary the token budget across layers and heads; H_2O (Zhang et al., 2023) and FastGen (Ge et al., 2023) manage cache updates during generation. The recurring difficulty is that token importance is query-dependent (Zhu et al., 2025), so a position discarded for one query may be needed by the next. Recent methods mitigate this by predicting future relevance (Zhu et al., 2025), re-scoring cheaply (Hooper et al., 2025) or reconstructing on demand (Kim et al., 2025). SALS (Mu et al., 2026) composes the two directions rather than choosing between them, projecting the cache into a low-rank latent space and then selecting tokens inside that space with RoPE-free query-key products, so only the selected tokens are reconstructed. Attention may also be a weaker relevance signal than it looks: Molfese et al. (2026) find that ranking documents by cumulative attention score does not track task accuracy across their fine-tuned checkpoints. Our method leaves the time axis untouched, allowing it to be combined with token eviction. Moschella et al. (2026) make the same point from the eviction side, naming representation compression as orthogonal and complementary to its learned eviction policy.

Architectural cache reduction and quantization. Multi-query (Shazeer, 2019) and GQA (Ainslie et al., 2023) share KV states across query heads within a layer, and cross-layer attention (Brandon et al., 2024) shares one set of KV states between adjacent layers. Multi-head latent attention (MLA), introduced in DeepSeek-V2 and retained in DeepSeek-V3 (Liu et al., 2024a;b), caches one low-rank latent per token per layer with a decoupled RoPE key. Pre-training an MLA model is out of reach for most groups, so TransMLA (Meng et al., 2025) and CARE (Zhou et al., 2026) convert a pretrained GQA checkpoint instead. Orthogonally, KVQuant (Hooper et al., 2024) identifies the per-channel and outlier structure that makes low-bit KV caching viable. Each of these fixes a single structural choice, or one bit-width, at conversion or pre-training time, and fixes it for all layers. Closest to our claim that this kind of compression has to be trained for, Molfese et al. (2026) fine-tune long-context models for in-context retrieval and then ask whether that transfers to robustness under KV-cache compression, reporting moderate gains that vary by task. They train for the task and measure compression robustness as a downstream property. We put the budget in the objective and train under the compression itself, across three composed axes.

Cross-layer sharing. Layers are redundant enough that a full per-layer cache is not obviously necessary (Monteiro et al., 2024; Wu & Tu, 2024). XC-Cache (Monteiro et al., 2024) shares a cache across layers but needs a separate encoder, so every update costs an encoder pass. Layer-Condensed KV Cache (Wu & Tu, 2024) shares a single layer's cache at a cost in time to first token, since approximating the top layer requires repeated forward passes. *Post-hoc* methods avoid retraining at the cost of how far they can go, merging pairs of late layers (Liu et al., 2024c), selecting shareable layers by a dissimilarity criterion (Yang et al., 2024), sharing projection weights by SVD (Wang et al., 2025a), or training alignment networks for cross-layer differences (Mu et al., 2024); Wu et al. (2025) evaluate sharing configurations systematically but pretrain each one. xKV (Chang et al., 2025) factorizes the caches of a group of layers into one shared low-rank subspace after training, on the observation that their dominant singular vectors align across layers, which composes our depth and rank axes in a single *post-hoc* step with the grouping and the rank both fixed before any training under them. Closest to our depth axis, Filippova et al. (2026) randomize the source layer of the keys and values during fine-tuning so one checkpoint tolerates sharing patterns chosen at test time.

Non-uniform and adaptive allocation. Several methods accept that a uniform per-layer setting is wasteful. PyramidKV (Cai et al., 2024) and CAKE (Qin et al., 2025) allocate token budgets non-uniformly across layers, and FlexRank (Zaccone et al., 2026) decomposes weights into nested low-rank factors so a rank can be chosen per deployment, under an additive cost model solved by a dynamic program. Ada-KV (Feng et al., 2026) and SqueezeAttention (Wang et al., 2025b) allocate one eviction budget non-uniformly across heads and across layers. KVTuner (Li et al., 2025) searches offline for a pair of key and value bit-widths per layer under a multi-objective criterion and then serves the configuration it found, which is our precision axis allocated per layer by search rather than learned, and applied to a checkpoint that never trained under it. MatryoshkaKV (Lin et al., 2025)

trains orthogonal projections of the cache features with a distillation objective and a nested schedule, then searches a compression rate per layer and per head for whatever budget it is given, which is our rank axis with a budget that can be set after training. STAR-KV (Bhatnagar et al., 2026) selects the rank per head and per block through a differentiable threshold and quantizes the result. Palu (Chang et al., 2024) projects the keys and values into a low-rank latent and searches a non-uniform rank per layer against a total compression target, then composes that with quantization. DMC (Nawrot et al., 2024) retrofits a pretrained model by continued training with a target compression ratio in the loss and a per-token gate trained through a straight-through estimator. Our setting is distinguished by three properties together, none of which is enough on its own. The cache configuration is over a *composed* action space in which depth, rank and precision are chosen jointly. It is made against *one* budget, so configurations are comparable. And the budget can be an input at runtime, so one selector answers a range of requests (Section 3.5). To our knowledge, none has placed all these three axes in one action space under one cost model and let a learned selector spend a single budget across them per layer. ThinK (Xu et al., 2025) prunes key channels per query and composes that with a quantizer, so what it chooses is a function of the prompt where what ours chooses is not. RAT+ (Wei & Gulcehre, 2026) pretrains one dense model that can be switched at inference to any of several dilated attention patterns, each reducing the cache by its dilation factor.

B Experimental details

Backbones and MLA conversion. Table 4 lists the backbones and their KV dimensions. For OLMo-3 we use its reasoning variant, `Olmo-3-7B-Think`. OLMo-3 mixes sliding-window and full attention layers, and a sliding-window layer keeps only its most recent positions, so no other layer can read its cache. We therefore run every OLMo-3 layer with full attention, in every OLMo-3 run including the controls. Configurations using rank start from a non-MLA model converted to MLA with CARE (Zhou et al., 2026) and balanced key-value scaling from TransMLA (Meng et al., 2025). We set $r = n_{\text{kv}} d_h$, qk_nope_head_dim $=$ v_head_dim $= d_h$, and the RoPE key width $d_R = 64$. We retain the `Qwen2.5` projection biases.

Table 4: **Backbone dimensions used to compute cache size.** L is the number of layers, n_{kv} the number of KV heads per layer, and $2n_{\text{kv}}d_h$ the number of cached scalars per token per layer. All models have head dimension $d_h = 128$. The converted MLA latent has width $r = n_{\text{kv}}d_h$, excluding the 64-dimensional positional key.

Backbone	Layers L	KV heads n_{kv}	Scalars/token/layer $2n_{\text{kv}}d_h$	MLA width r
`Qwen2.5-1.5B-Instruct`	28	2	512	256
`Qwen2.5-3B-Instruct`	36	2	512	256
`Qwen2.5-7B-Instruct`	28	4	1024	512
`Qwen2.5-14B-Instruct`	48	8	2048	1024
`Qwen2.5-32B-Instruct`	64	8	2048	1024
`Mistral-7B-Instruct-v0.3`	32	8	2048	1024
`Olmo-3-7B-Think`	32	32	8192	4096

Quantization. On non-MLA backbones, we quantize K_l and V_l. On MLA backbones, we quantize the retained latent c_l^{KV} and the positional key k_l^R at the same bit-width. We use symmetric uniform quantization with one scale per stored vector, determined by its maximum absolute value. Training quantizes and dequantizes the cache in the forward pass and uses straight-through gradients. Appendix C gives the quantization rule and storage accounting.

Data and optimization. We fine-tune on Dolci-Instruct-SFT[1] (Olmo Team et al., 2025), applying the chat template, supervising assistant turns only, and truncating sequences to 4096 tokens. Each run uses 270,000 steps, corresponding to approximately one epoch, with one sequence per data-parallel rank on eight GPUs: eight sequences and at most 32,768 tokens per step. A held-out instruction-following subset provides validation loss and token accuracy.

[1] https://huggingface.co/datasets/allenai/Dolci-Instruct-SFT

We optimize the backbone and selector jointly with AdamW (Loshchilov & Hutter, 2017), using a shared learning rate of 10^{-5}, linear decay, 2% warmup, weight decay 10^{-4}, and gradient clipping at 1.0. Training uses BF16 and gradient checkpointing. Models up to 7B use A100 or H100 nodes; 14B models use Blackwell nodes, with eight accelerators per job.

Selector architecture. The selector projects the backbone's token embeddings from width H to 256 and applies two bidirectional transformer blocks. Each block has four attention heads of width 64, rotary embeddings on queries and keys, and a SwiGLU feed-forward layer of inner width 1024. RMSNorm precedes each sublayer, with residual connections around both; a final RMSNorm follows the second block. We average the output over tokens to obtain $z \in \mathbb{R}^{256}$. Each of the L layers has an independent linear head mapping z to $|\mathcal{A}|$ action logits.

With 17 actions, the selector has 2.61, 2.78, 3.14, 3.62, and 3.69M parameters for `Qwen2.5` at 1.5, 3, 7, 14, and 32B, respectively, and 3.29M for both `Mistral-7B` and `OLMo-3-7B`, which share a width and a depth. These are 0.17%, 0.09%, 0.041%, 0.025%, and 0.011% of the `Qwen2.5` backbones, and 0.045% of each of the other two. The per-layer heads account for 122k–280k parameters. Appendix F.10 compares alternative architectures.

Budget conditioning. A FiLM MLP with 33k parameters maps the normalized target $\tilde{\rho}_\star$ to a scale and shift:

$$(\gamma, \beta_{\mathrm{F}}) = \mathrm{MLP}(\tilde{\rho}_\star), \qquad z \mapsto z \odot (1 + \gamma) + \beta_{\mathrm{F}}.$$

Each training sequence samples a target from the training set of compression factors (Table 3), normalized to approximately $[-1, 1]$. We exclude runs whose range of realized compression factors across requested targets is less than 5% of their mean.

Initialization and schedules. We initialize trunk and head weights from $\mathcal{N}(0, 10^{-3})$. Head biases are zero except for a bias of 5.0 on the least-compressed action. The Gumbel temperature decreases geometrically from 2.0 to 0.1 over training, with one sample per sequence at each optimizer step. Appendix D specifies the adaptive rate multiplier, whose default cap is 10^4. For constant-multiplier runs ($\eta = 0$), we increase β linearly to the selected value during warmup.

Evaluation. We rely on the lm-evaluation-harness (Gao et al., 2024). GSM8K scores use the flexible-extract metric. Decoding is greedy on fixed evaluation sets. Table 16 reports variation across training seeds.

Timing. We measure pre-fill and selector latency at batch size one on an H100, using precision and depth at a compression factor of 4×. We fix the scaled-dot-product attention kernel, synchronize the device before and after each timed region, discard four warmups, and report the median of 12 repetitions. The reference uses the same weights, kernel, and batch size without the selector. Appendix F.10 reports latency and training cost.

C Action Space and Cache Storage

C.1 Compression Ranges

Table 5 lists the available actions and compression ranges for two backbone geometries. Disabling depth masks the inherit action. The minimum compression factor uses the highest-cost action at every layer. The maximum uses the lowest-cost action at every layer when depth is disabled, or at a single anchor when depth is enabled:

$$\rho_{\min} = \frac{C_0}{L \max_a c(a)}, \qquad \rho_{\max} = \begin{cases} C_0 / \min_a c(a) & \text{depth enabled,} \\ C_0 / (L \min_a c(a)) & \text{depth disabled.} \end{cases} \tag{5}$$

The extrema are over actions that produce a cache. We check fixed training targets against this range. At the precision-only maximum of 8×, every layer must use two bits, leaving only one feasible configuration.

The positional key limits rank compression because truncating the latent does not reduce d_R. MLA conversion itself reduces cache size, so its compression factors start above 1×. All factors in Table 5 include this initial reduction.

Table 5: **Action counts and feasible cache reduction factors.** Approaches lists the enabled compression axes; backbone identifies the attention architecture. Actions counts choices per layer, Plans counts assignments across L layers before budget and first-layer constraints, and $[\rho_{\min}, \rho_{\max}]$ gives minimum and maximum cache reduction relative to the original BF16 GQA cache. The two models have $(L, 2n_{\text{kv}}d_h, r) = (48, 2048, 1024)$ and $(36, 512, 256)$. Rank uses MLA with positional-key width $d_R = 64$. Bit-widths are $\{2, 4, 8, 16\}$ and latent widths $\{r/8, r/4, r/2, r\}$.

			`Qwen2.5-14B`		`Qwen2.5-3B`	
Approaches	Backbone	Actions	Plans	$[\rho_{\min}, \rho_{\max}]$	Plans	$[\rho_{\min}, \rho_{\max}]$
Precision	GQA	4	10^{29}	1.00–8.0×	10^{22}	1.00–8.0×
Depth	GQA	2	10^{14}	1.00–48.0×	10^{11}	1.00–36.0×
Depth + precision	GQA	5	10^{34}	1.00–384×	10^{25}	1.00–288×
Rank	MLA	4	10^{29}	1.88–10.7×	10^{22}	1.60–5.3×
Rank + precision	MLA	16	10^{58}	1.88–85.3×	10^{43}	1.60–42.7×
Depth + rank	MLA	5	10^{34}	1.88–512×	10^{25}	1.60–192×
Depth + rank + precision	MLA	17	10^{59}	1.88–4096×	10^{44}	1.60–1536×

C.2 MLA CACHE REPRESENTATION

An MLA layer stores a latent c^{KV} and a positional key k^R for each token. The content key is reconstructed from the latent using the layer's projection weights. Since it does not undergo RoPE, this projection can be absorbed into the query projection. The positional key is stored after RoPE and shared across heads. The cached width is therefore $d + d_R$, with full widths $512 + 64$ at 7B and $1024 + 64$ at 14B.

During training, rank truncation uses a prefix mask: coordinates after d are set to zero while the tensor retains its full shape, which allows different widths within a batch. A deployed kernel would instead write only the first d coordinates. We use no explicit nested-latent objective; the representation is initialized by conversion and updated during fine-tuning. Training computes all candidate actions for the straight-through estimator. With gradients disabled, it computes only the selected action; unselected actions have zero weight.

C.3 QUANTIZATION AND STORAGE OVERHEAD

Our symmetric quantizer uses one scale per stored vector and no zero-point. For bit-width b, let $q = 2^{b-1} - 1$ and $s = \max_i |x_i|/q$. The integer code is $\text{clip}(\text{round}(x/s), -q, q)$, and dequantization multiplies it by s. At two bits, this gives three levels. A grouped asymmetric quantizer instead stores a scale and zero-point per group of g channels and uses all 2^b levels.

Table 6: **Storage cost and cache reduction with quantization metadata.** b is the quantization bit-width; bits/element includes metadata and any full-precision residual window. ρ is 16 divided by this storage cost, relative to BF16. Our quantizer stores one BF16 scale per 128-element token-head row. KIVI stores a scale and zero-point per group of g elements and a 128-token BF16 window in a 2048-token context. Bold highlights the factors used in the text.

Scheme	Bit-width b	Bits/element ↓	ρ ↑
Our quantizer	8	8.125	1.97×
Our quantizer	4	4.125	**3.88**×
Our quantizer	2	2.125	7.53×
KIVI, g=32	4	5.688	2.81×
KIVI, g=64	4	5.219	3.07×
KIVI, g=128	4	4.984	3.21×
KIVI, g=32	2	3.812	4.20×
KIVI, g=64	2	3.344	4.79×
KIVI, g=128	2	3.109	**5.15**×

Table 6 includes quantization metadata and the unquantized residual window. At two bits and $g = 128$, KIVI reaches $5.15\times$ compression for a 2048-token context, or $7.11\times$ without the window. At 16k, the fixed 128-token window accounts for a smaller fraction of the cache: four-bit KIVI with $g = 128$ uses 4.34 bits per element, giving the $3.7\times$ factor in Table 2.

D SETTING THE RATE MULTIPLIER β

D.1 UPDATE RULE

We adapt the rate multiplier in Equation (4) to the difference between requested and realized compression. At each training step, we compute $v = \rho_\star/\rho - 1$, which is positive when the cache exceeds its target size, and update

$$\overline{|v|} \leftarrow \lambda\overline{|v|} + (1-\lambda)|v|, \qquad \beta \leftarrow \mathrm{clamp}\Big(\beta + \eta(\overline{|v|} - \delta), 0, \beta_{\max}\Big)\,.$$

We use $\lambda = 0.9$, $\eta = 0.05$, $\delta = 0.02$, $\beta_{\max} = 10^4$, and initialize $\beta = 0$. The multiplier is all-reduced across data-parallel ranks before the update.

D.2 BUDGET ACCURACY AND TASK PERFORMANCE

The adaptive rule reaches the target in all 51 evaluated fixed-target configurations. Constant multipliers are less reliable near the maximum feasible compression: with only precision on `Qwen2.5-7B`, tested constant settings reach $6.32\times$ rather than the requested $8\times$, for instance.

Table 7 compares adaptive and swept constant multipliers at matched realized compression. Six pairs differ by at most 0.053 in accuracy, the largest observed range across training seeds. The swept constant performs better in the other four, including all three precision-and-depth pairs. Reaching the target budget therefore does not guarantee the best task accuracy. One limitation is slow decay: the update decreases β by at most $\eta\delta = 0.001$ per step, so a large multiplier can persist after the target has been reached.

We thus cap the multiplier at $\beta_{\max} = 50$ to test this, at realized factors within 2% of the target. On `Qwen2.5-7B` with precision and depth, the cap raises accuracy from 0.255 to 0.496 at $8\times$ and from 0.162 to 0.327 at $16\times$, and on `Qwen2.5-14B` at $16\times$ from 0.322 to 0.535. With precision and rank, which do not use depth, the same cap raises accuracy by 0.049 at $8\times$ and 0.055 at $16\times$. With all three axes, the capped selector inherits on no layer at $8\times$ and on one of 28 at $16\times$. At 14B and $16\times$, the cap changes accuracy by $+0.005$ with all three approaches, and almost no layer inherits.

Table 7: **Mean IFEval and GSM8K accuracy with adaptive and swept constant rate multipliers.** Rows identify the compression axes and `Qwen2.5` model size; $\rho_\star$ is the target cache reduction factor. Adaptive and Swept give accuracies; the last column subtracts adaptive from swept accuracy. Pairs differ by at most 2% in realized compression. The upper six accuracy differences are within 0.053, the largest observed seed range.

		Mean accuracy ↑		
Approaches	Target $\rho_\star$	Adaptive	Swept	Accuracy difference ↑
All three, 7B	$8\times$	0.541	0.534	-0.007
Precision, 7B	$4\times$	0.670	0.668	-0.002
All three, 14B	$16\times$	0.610	0.571	-0.039
All three, 7B	$16\times$	0.429	0.448	$+0.019$
All three, 14B	$8\times$	0.622	0.644	$+0.022$
All three, 7B	$4\times$	0.547	0.579	$+0.032$
All three, 3B	$4\times$	0.327	0.484	$+0.157$
Precision + depth, 7B	$16\times$	0.162	0.388	$+0.226$
Precision + depth, 14B	$16\times$	0.322	0.563	$+0.241$
Precision + depth, 7B	$8\times$	0.255	0.562	$+0.307$

Figure 6 shows representative runs with all three axes. Compression stabilizes near the target by about 30% of training. The multiplier initially increases and then decreases as the target is reached.

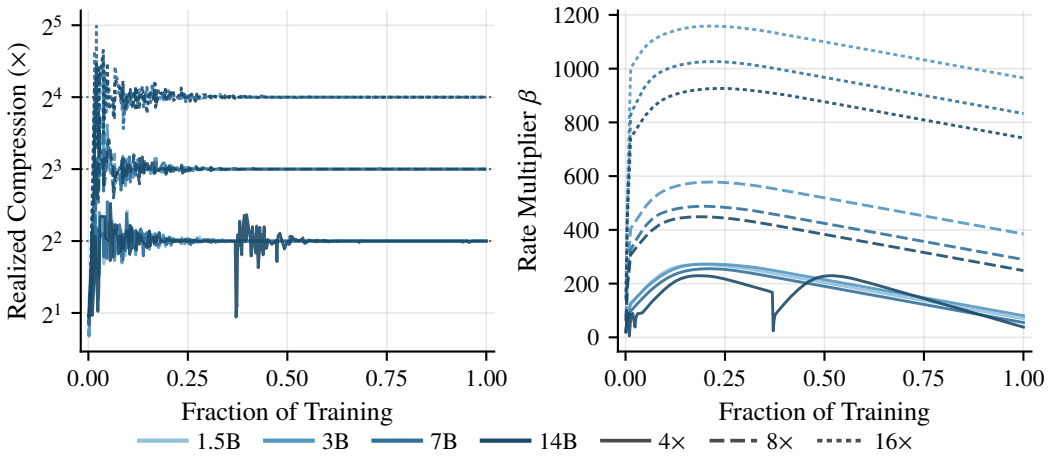

Figure 6: **Compression factor (left) and rate multiplier (right) during training.** Ten runs use all three axes on `Qwen2.5` at 1.5, 3, 7, and 14B, with targets $4\times$, $8\times$, and $16\times$; 1.5B is evaluated only at $4\times$. Color denotes model size and line style denotes target. Training progress is normalized by 270,000 steps.

E Comparing cache configurations at a fixed budget

E.1 Alternatives to the learned configuration

We check whether the selector could have picked a better configuration for the model it trained with. For each trained 7B model we hold the weights fixed and score 24 other configurations of about the same cache cost on 128 held-out prompts. Table 8 answers three questions. *Loss range* asks whether the choice matters at all: it is how far the alternatives' losses spread on a prompt, averaged over prompts. *Per-group loss gain* asks whether choosing a different configuration for each group of prompts would help: the groups' configurations are chosen on half the prompts and scored on the other half. *Mean regret* asks whether any alternative beats the selector: it is the selector's loss minus the best alternative's, so a negative value means none does.

Table 8: **Loss differences between learned and alternative cache configurations.** We score 24 alternatives on 128 prompts with trained 7B weights fixed and cache costs approximately matched. Negative regret favors the selector. All differences are in nats.

Approaches	Loss range	Per-group loss gain ↑	Mean regret ↓
Depth only	1.4826	+0.0000	−0.4988
All three	0.3014	+0.0048	−0.0143

No alternative beats the selector in either action space, and choosing per prompt group gains almost nothing. The choice matters most with only depth available, where the alternatives' losses spread by 1.48 and the selector is 0.50 below the best of them. With all three approaches the spread is 0.30 and the margin 0.014. Because the weights are those trained with the selector, this does not show which configuration would be best if each were trained on its own. Table 15 compares that on downstream tasks.

E.2 Offline search

We check how much the choice of configuration matters when the cache size is fixed. To do so, we search for a configuration on the fine-tuned, uncompressed MLA control. We compress one layer at a time with each action to estimate how much each costs, and a dynamic program combines these estimates into the configuration with the lowest estimated loss that fits the budget. We compare it with random configurations and with a uniform one, which applies the same action to every layer.

Table 9: **Loss increase from compressing the uncompressed MLA control of `Qwen2.5-7B` without training.** Measured is the rise in teacher-forced loss on calibration data; predicted is the sum of per-layer rises that the search minimizes. Random is the best of 64 random configurations at the same budget. No uniform configuration meets the $4\times$ budget exactly, and dashes mark values not measured. All values are in nats.

	Measured ↓		Predicted
Configuration	$4\times$	$8\times$	$8\times$
Searched	0.005	0.030	0.022
Random	0.125	1.253	–
Uniform	–	4.00	0.264

At the same cache size, the searched configuration barely changes the loss, random ones raise it clearly, and the uniform one raises it far more (Table 9). The per-layer estimates predict the searched configuration well but miss most of the uniform one's loss, because when every layer is compressed too much, the damage no longer adds up layer by layer. Which configuration is used therefore matters far more than the cache size alone, which is why we choose it per layer.

F Additional experimental results

This appendix collects the results that support Section 3. Each subsection states what it checks and what it finds, and gives the full tables behind numbers the main text quotes.

F.1 Quantization and smaller-model baselines

We check whether the same cache reduction can be had without training, either by quantizing a model after training or by serving a smaller model. Table 10 compares the three against the uncompressed 7B control, with every cache size relative to that model. Applying our plain quantizer after training destroys the model, while training under the same quantizer keeps it level with the control, so the quantizer is not the problem; the model has to train under it. A stronger post-hoc quantizer, KIVI, holds up at four bits but starts to lose accuracy at two bits. Serving a smaller uncompressed model is the weakest option: `Qwen2.5-1.5B` saves only $2\times$ and falls far below a 7B model compressed $4\times$.

Table 10: **Cache reduction and task accuracy with quantization, trained compression, and smaller models.** ρ is the cache reduction factor relative to the full `Qwen2.5-7B` BF16 cache. IFEval and GSM8K report task accuracies on $[0, 1]$; Mean is their arithmetic mean. Parentheses give mean-accuracy differences from the SFT control; bold marks means within 0.053 of that control or higher. G is the quantization group size. Post-hoc factors include metadata; trained precision gives $4.00\times$ before metadata and $3.88\times$ after it.

		Accuracy ↑		
Configuration	ρ ↑	IFEval	GSM8K	Mean
`Qwen2.5-7B-Instruct`, base	1.00×	0.706	0.701	0.704
Our SFT, uncompressed (control)	1.00×	0.719	0.639	0.679
+ post-hoc 4-bit, plain	3.88×	0.113	0.001	0.057 (−0.622)
+ *trained under* KIVI, 4-bit G=32	2.81×	0.701	0.680	**0.690** (+0.011)
+ post-hoc KIVI, 4-bit G=32	2.81×	0.721	0.635	**0.678** (−0.001)
+ post-hoc KIVI, 4-bit G=128	3.21×	0.723	0.638	**0.680** (+0.001)
+ post-hoc KIVI, 2-bit G=128	5.15×	0.713	0.590	**0.652** (−0.027)
+ trained precision	4.00×	0.723	0.617	**0.670** (−0.009)
+ trained precision + depth	4.00×	0.721	0.637	**0.679** (−0.000)
`Qwen2.5-3B`, uncompressed	1.56×	0.706	0.641	0.673
`Qwen2.5-1.5B`, uncompressed	2.00×	0.645	0.509	0.577

F.2 Accuracy by compression axis

Table 11 breaks the 7B column of Table 1 down per task, to show where each approach loses accuracy. Converting to MLA costs accuracy, so configurations using rank are read against the converted control, which separates the choice from the conversion. Against that reference, every configuration that uses rank or precision stays within the seed range. Depth on its own is the exception, and it loses far more on reasoning than on instruction following.

Table 11: **Task accuracy by compression axis on `Qwen2.5-7B` at $4\times$ cache reduction.** IFEval and GSM8K report task accuracies on $[0, 1]$; Mean is their arithmetic mean. Ref. identifies the control used for parenthesized mean-accuracy differences: MLA for configurations using rank, GQA otherwise. The conversion-only row is compared with GQA. Bold marks means within 0.053 of the corresponding control or higher

		Accuracy ↑		
Approaches	Ref.	IFEval	GSM8K	Mean
Uncompressed (GQA)	–	0.719	0.639	0.679
MLA conversion only	GQA	0.636	0.513	0.574 (−0.105)
Precision	GQA	0.723	0.617	**0.670** (−0.009)
Depth	GQA	0.547	0.245	0.396 (−0.283)
Precision + depth	GQA	0.721	0.637	**0.679** (−0.000)
Rank	MLA	0.601	0.492	**0.546** (−0.028)
Precision + rank	MLA	0.619	0.522	**0.571** (−0.003)
Depth + rank	MLA	0.593	0.473	**0.533** (−0.041)
All three	MLA	0.632	0.461	**0.547** (−0.028)

F.3 Long-context evaluation

Eviction implementations. Our SnapKV (Li et al., 2024), PyramidKV (Cai et al., 2024), and H_2O(Zhang et al., 2023) implementations sum attention over a 32-query observation window. Published H_2O accumulates attention over all past queries; our pre-fill implementation is therefore a restricted variant. StreamingLLM (Xiao et al., 2024) retains attention sinks and a recent-token window. Recency and random selection do not use attention scores.

Results at 7B. We repeat the long-context comparison at 7B, with four further eviction policies (Table 12). Our $4\times$ configuration is at the same level as query-informed SnapKV and PyramidKV, and combined with SnapKV at $16\times$ it is at the same level as four-bit KIVI under the same evictor. The policies that choose which tokens to keep before the query arrives fall far behind, as does using depth or rank on its own at this context length.

Splitting a larger reduction at 14B. We ask how to spend a large decode-time cache reduction at 16k: on our compression, on eviction, or on both. Splitting it works best (Table 13). At $32\times$, a $4\times$ configuration with eviction keeping one position in eight stays level with the uncompressed control, an $8\times$ configuration with eviction keeping one in four falls behind it, and reaching $16\times$ with our compression alone collapses question answering. Under the same evictor, our configuration also beats four-bit KIVI at every retained fraction, each time with a smaller cache. This panel uses the non-MLA backbone because converting to MLA alone costs most of the question-answering accuracy at this context length.

Table 12: **RULER accuracy at 16k on `Qwen2.5-7B`.** ρ is the decode-time cache reduction factor relative to the uncompressed GQA model. NIAH single and multi-key report retrieval accuracy; QA reports SQuAD question-answering accuracy. Parentheses give QA differences from the control. Keep is the retained token fraction, and G the quantization group size. Rank uses MLA; other rows use GQA. Bold identifies KVK configurations and their combinations with eviction. The last four policies select tokens before the query arrives.

		Accuracy ↑		
Configuration	ρ ↑	NIAH single	NIAH multi-key	QA (SQuAD)
No compression (control)	1.0	1.000	0.985	0.540
KVK, 4× (precision + depth)	4.0	0.996	0.956	0.510 (−0.030)
+ SnapKV, keep 0.25	16.0	0.985	0.887	0.472 (−0.068)
KVK, 3.9× (precision only)	3.9	0.978	0.913	0.508 (−0.032)
KVK, 4× (depth only)	4.0	0.964	0.324	0.107 (−0.434)
KVK, 4× (rank only, converted)	4.0	0.978	0.247	0.165 (−0.376)
KIVI 4-bit G=32	3.1	1.000	0.989	0.545 (+0.005)
KIVI 4-bit G=128	3.7	1.000	0.978	0.534 (−0.006)
KIVI 4-bit G=128 + SnapKV 0.25	14.8	1.000	0.865	0.474 (−0.067)
SnapKV, keep 0.25	4.0	1.000	0.891	0.474 (−0.066)
PyramidKV, keep 0.25	4.0	1.000	0.807	0.469 (−0.072)
StreamingLLM, keep 0.25	4.0	0.229	0.305	0.272 (−0.268)
Recency, keep 0.25	4.0	0.229	0.305	0.242 (−0.299)
H_2O, keep 0.25	4.0	0.331	0.000	0.396 (−0.144)
Random, keep 0.25	4.0	0.000	0.000	0.179 (−0.362)

Table 13: **Question answering at 16k on `Qwen2.5-14B` when a larger cache reduction is split between our compression and eviction.** ρ is the decode-time cache reduction relative to the uncompressed GQA model, and keep the fraction of positions SnapKV retains. KVK configurations use precision and depth. Parentheses give differences from the uncompressed control; the 32× configuration with keep 0.125 is 0.012 below it. In the lower block, KVK and four-bit KIVI with $G = 128$ share the evictor, and KVK is ahead by 0.098, 0.074 and 0.067.

Configuration	ρ ↑	QA (SQuAD) ↑
Uncompressed control	1.0	0.543
32× in total		
KVK 4× + SnapKV, keep 0.125	32.0	0.531 (−0.012)
KVK 8× + SnapKV, keep 0.25	32.0	0.491 (−0.052)
Without eviction		
KVK 16×	16.0	0.258 (−0.285)
MLA conversion only	1.9	0.210 (−0.333)
KVK 4× / four-bit KIVI, same evictor		
Keep 0.5	8.0 / 7.4	0.595 / 0.498
Keep 0.25	16.0 / 14.8	0.560 / 0.486
Keep 0.125	32.0 / 29.5	0.531 / 0.464

F.4 Effect of fine-tuning

SFT improves the smaller Qwen models, while changes at 7B and above range from −0.052 to +0.014. Mistral loses 0.152. These differences motivate reporting compression relative to a control trained with the same recipe.

Table 14: **Mean IFEval and GSM8K accuracy before and after SFT without cache compression.** Untuned and After our SFT report arithmetic means of the two task accuracies. The Accuracy change is the score after SFT minus the untuned score, computed before rounding.

	Mean accuracy ↑		
Backbone	Untuned	After our SFT	Accuracy change
`Qwen2.5-0.5B`	0.297	0.436	+0.139
`Qwen2.5-1.5B`	0.488	0.577	+0.090
`Qwen2.5-3B`	0.581	0.673	+0.093
`Qwen2.5-7B`	0.704	0.679	−0.025
`Qwen2.5-14B`	0.774	0.723	−0.052
`Qwen2.5-32B`	0.779	0.793	+0.014
`Mistral-7B`	0.501	0.349	−0.152

F.5 Learned and static cache configurations

Here, we test two parts of the per-layer choice separately (Table 15). The first is where to place the layers that keep their own cache once their number is fixed: with only depth available at $4\times$, three layers in four must inherit, and we compare the learned placement with evenly spaced and randomly drawn anchors. The second is whether layers should share at all: with all three approaches available, we compare the learned configuration with one that forces every second layer to inherit and one that allows no sharing.

The three placements are close to each other at 3B, and evenly spaced anchors do better than learned ones at 7B. What costs seems to be how many layers inherit, not which. Simple sharing patterns can therefore place anchors when sharing is required, but whether to share is worth learning, since precision and rank can often meet the budget without it, and a learned selector can rely on depth only if needed.

Table 15: **Task accuracy with learned and fixed cross-layer sharing.** Inheriting fraction is the fraction of layers reading another layer's cache; $4\times$ and $8\times$ columns give arithmetic mean IFEval and GSM8K accuracy at those cache reduction targets. Control rows give uncompressed accuracy. Parentheses subtract the block's control; dashes indicate unavailable comparisons. Bold marks scores within 0.053 of the best compressed configuration in each block and column, excluding the starred entry. In the third block, learned, fixed-sharing, and no-sharing runs realize 4.00, 3.94, and $4.00\times$, respectively, and $8.00\times$ at the higher target. *Constant rate multiplier; other compressed rows use the adaptive rule.

		Mean accuracy ↑ at target $\rho_\star$	
Sharing configuration	Inheriting fraction	$4\times$	$8\times$
Which layers hold the anchors: only depth available, GQA backbone, `Qwen2.5-3B`			
Uncompressed control	0.000	0.673	–
Learned	0.750	**0.320** (−0.354)	–
Spaced evenly	0.750	**0.339** (−0.334)	–
Drawn at random	0.750	**0.360** (−0.313)	–
Which layers hold the anchors: only depth available, GQA backbone, `Qwen2.5-7B`			
Uncompressed control	0.000	0.679	–
Learned	0.750	0.396 (−0.283)	–
Spaced evenly	0.750	**0.452** (−0.227)	–
Drawn at random	0.750	**0.437** (−0.242)	–
How many layers inherit: all three axes, MLA backbone, `Qwen2.5-7B`			
Conversion only (control)	0.000	0.574	0.574
Learned	0.000	**0.547**	**0.541**
Fixed sharing, every second layer	0.500	0.460	0.466
No sharing	0.000	**0.571**	0.558*

F.6 Training seeds and model size

Table 16 reports repeated training runs under different random seeds, to show how much a result moves when only the seed changes. The largest accuracy range, 0.053, belongs to the uncompressed 7B control and serves as a descriptive reference.

Table 16: **Mean IFEval and GSM8K accuracy across training seeds.** Rows identify the `Qwen2.5` model size and compression axes. $\rho_\star$ is the target cache reduction factor; Seeds counts training runs. Mean accuracy lists the arithmetic mean accuracy for each seed. Bold marks the largest accuracy range, 0.053. Other tables use the first listed seed.

Configuration	Target $\rho_\star$	Seeds	Mean accuracy ↑
Uncompressed, 7B	$1\times$	3	**0.679, 0.635, 0.626**
Uncompressed, 14B	$1\times$	2	0.723, 0.681
Precision + depth, 7B	$4\times$	3	0.679, 0.679, 0.652
Precision + depth, 7B	$8\times$	3	0.255, 0.262, 0.288
Precision + depth, 14B	$16\times$	2	0.322, 0.295
All three, 3B	$4\times$	3	0.327, 0.334, 0.347

Table 17 extends the compression comparison to 32B. The accuracy cost of MLA conversion is not monotonic with model size: accuracy decreases by 0.178, 0.105, 0.091, and 0.131 at 3, 7, 14, and 32B. At 32B, language modeling loss nevertheless improves from 0.448 to 0.353, showing that lower loss need not imply higher task accuracy.

Table 17: **Mean IFEval and GSM8K accuracy differences across `Qwen2.5` model sizes.** Columns give parameter counts; blocks give realized cache reduction factors. The first two rows report absolute arithmetic mean accuracies after and before SFT. Other entries subtract the size-matched MLA control for configurations using rank, or GQA control otherwise. Bold marks differences ≥ -0.053; dashes indicate unavailable runs.

	Mean accuracy / difference ↑				
Approaches	1.5B	3B	7B	14B	32B
Uncompressed control	0.577	0.673	0.679	0.723	0.793
Untuned model	0.488	0.581	0.704	0.774	0.779
At a matched realized $4\times$					
Precision	-0.034	-0.041	$\mathbf{-0.009}$	$\mathbf{+0.040}$	$\mathbf{+0.016}$
Precision + depth	–	-0.063	$\mathbf{-0.000}$	$\mathbf{+0.031}$	$\mathbf{+0.030}$
All three	-0.174	-0.169	$\mathbf{-0.028}$	$\mathbf{+0.021}$	–
At a matched realized $8\times$					
All three	–	-0.161	$\mathbf{-0.033}$	$\mathbf{-0.010}$	–

F.7 Discrete and continuous budget conditioning

We first check whether it matters if the targets are given as a set of factors or as a continuous range. With all three approaches, a selector trained on the set $\{2, 4, 8\}$ and one trained over the range $[2, 8]$ are within close performance of each other wherever both meet the request (Tables 3 and 18), so the form of the targets matters little.

We then check whether more training closes the gap to single-target selectors. Doubling the steps helps the range selector at every requested factor, and it reaches near the single-target selectors it replaces, for two thirds of their total training cost in terms of steps (Table 3).

We finally check why wider sets fail. With depth available, every target in Table 18 can be reached, yet the wider sets stop following the request, mostly by compressing more than asked, and their rate multiplier does not settle, ending training up to ten times higher than for $\{1, 2, 4\}$.

Table 18: **Mean IFEval and GSM8K accuracy with selectors trained on wider sets of targets, on `Qwen2.5-7B`.** $\rho_\star$ is the requested cache reduction factor, and blocks follow Table 3. Parentheses give the realized factor where a selector misses the request by more than 2%. Final β is the rate multiplier at the end of training. Dashes mark targets outside the training set or values not recorded.

	Mean accuracy ↑ at requested $\rho_\star$						
Training targets	1×	2×	4×	8×	16×	32×	Final β
Precision and depth							
$\{1, 2, 4\}$	0.703	0.659	0.612	–	–	–	355
$\{1, 2, 4, 8\}$	0.672	0.395 (2.31×)	0.355	0.353	–	–	886
$\{1, 2, 4, 8, 16\}$	0.655	0.471 (2.33×)	0.269 (5.47×)	0.206 (9.00×)	0.213	–	2251
$\{1, 2, 4, 8, 16, 32\}$	0.585	0.263	0.100 (6.78×)	0.107 (20.4×)	0.106 (32.0×)	0.106	3722
All three approaches							
$\{2, 4, 8\}$	–	0.481	0.463	0.460	–	–	–
$\{4, 8, 16\}$	–	–	0.340 (3.80×)	0.319	0.306	–	–

F.8 Use of cross-layer sharing

Table 19: **Fraction of layers that inherit another layer's cache in the selector's configuration.** Ranges span the evaluated model sizes, and the all-three row covers eleven fixed-target configurations. Past 8× on a non-MLA backbone, seven of 73 trained configurations reach the factor, all with 50% to 94% of layers inheriting. One budget-conditioned configuration with all three approaches inherits at 3.66×, for 21.5% of its memory saving. Dashes mark values not recorded.

		Inheriting fraction at target $\rho_\star$		
Approaches	Backbone	4×	8×	16×
Precision + depth	Non-MLA	0.000	0.021–0.107	0.500–0.528
Depth + rank	MLA	–	–	0.333–0.667
All three	MLA	0.000	0.000	0.000

This appendix assesses when the selector uses depth, the approach that lets a layer read another layer's cache (Table 19). The answer depends on what else is available. On a non-MLA backbone, precision cannot pass 8×, so depth is the only way further, and every configuration we train past that factor lets at least half of its layers inherit. Below it, the selector uses depth little or not at all, and more of it as the budget tightens. With precision and rank both available, it does not use depth at any budget we train with a fixed target, because the two meet those budgets together.

The selector therefore treats depth as the approach of last resort, used when the others cannot meet the budget. This matches Table 15, where imposing sharing that is not needed costs accuracy.

F.9 Cache configuration diversity

Table 20 counts the unique per-layer configurations each selector returns on 64 IFEval prompts, 64 GSM8K prompts, and 96 RULER prompts, 32 each at 4k, 8k, and 16k tokens. Across model sizes, six selectors return several configurations and six return a single one, re-used across inputs.

Table 20: **Number of distinct per-layer cache configurations returned by selectors on `Qwen2.5.`** Rows give the model size and the target cache reduction factor $\rho_\star$. Each entry is the number of distinct configurations a selector returns over the prompts sampled from a dataset, out of the number of prompts: 64 from IFEval, 64 from GSM8K, and 96 from RULER, 32 at each of 4k, 8k, and 16k tokens. The last column pools all 224 prompts. Selectors in the upper block use all three approaches. †Rate multiplier capped at 50. ‡Constant rate multiplier; this selector realizes 7.73×.

		Distinct configurations / prompts			
Model size	Target $\rho_\star$	IFEval	GSM8K	RULER	All three
One configuration					
3B	4×	1/64	1/64	1/96	1/224
3B	16×	1/64	1/64	1/96	1/224
7B	4×	1/64	1/64	1/96	1/224
7B	8×	1/64	1/64	1/96	1/224
14B	4×	1/64	1/64	1/96	1/224
14B	16×	1/64	1/64	1/96	1/224
Several configurations					
3B	2×	4/64	1/64	6/96	6/224
3B†	4×	2/64	3/64	1/96	3/224
7B†	8×	5/64	2/64	1/96	5/224
7B†	16×	3/64	1/64	1/96	3/224
14B‡	8×	9/64	1/64	13/96	20/224
14B†	16×	4/64	2/64	1/96	4/224

F.10 Selector architecture

Architecture comparison. To check whether the selector's design matters, we compare our transformer over the token embeddings with a two-layer MLP over mean-pooled embeddings and with linear heads over the backbone's last-layer hidden states (Table 21). With precision only, the three reach similar accuracies. With all three approaches, the hidden-state variant scores higher, at a slightly larger cache, and the MLP falls behind. We keep the transformer because it reads only the prompt's embeddings and so runs before pre-fill, whereas the hidden-state variant needs a full backbone pass to choose the configuration and then a second pre-fill under it.

Table 21: **Mean IFEval and GSM8K accuracy with three selector architectures on `Qwen2.5-7B` at 4×.** Every selector realizes 4.00× except the hidden-state variant with all three approaches, which realizes 3.86×.

		Mean accuracy ↑	
Selector	Input	Precision	All three
Transformer (ours)	Token embeddings	0.670	0.547
Two-layer MLP	Mean-pooled embeddings	0.660	0.510
Linear heads	Last-layer hidden states	0.674	0.601

Training time. Across 3, 7, and 14B models, training takes 1.0–1.4× as long as ordinary fine-tuning with one cache-producing action per layer, 1.3–1.8× with four, and 1.9–3.1× with sixteen. Each run uses 270,000 steps on eight NVIDIA H100 GPUs. For precision-only `Qwen2.5-7B`, three separate target-specific runs total 136 hours, compared with 34.6 hours for one selector trained on four targets.

G On Device benchmarks

All benchmarks are realized on a M3 Ultra with 512GB of RAM. GPU and CPU temperatures and frequencies are monitored to ensure no thermal throttling take place during measurements.

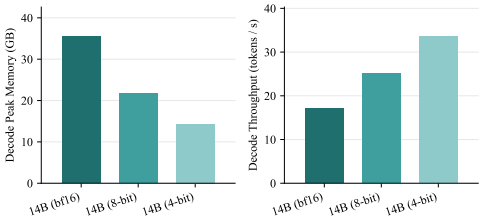

Figure 7: Qwen2.5-14B at three weight precisions, at 27,611-token prompt followed by 2,048 generated tokens. The KV cache is uncompressed bf16 in all three.

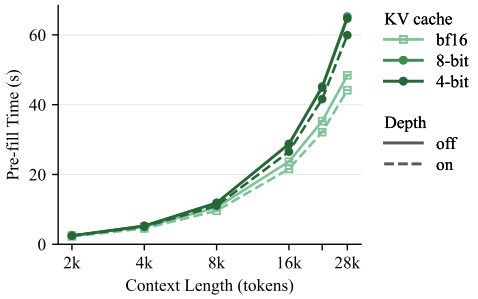

Figure 8: Qwen2.5-14B (4 bits weights) at three KV-cache precision.

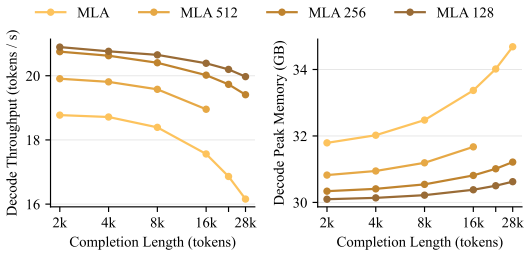

Figure 9: Qwen2.5-14B (4 bits weights) with bf16 KV-cache.

G.1 The Role of Precision

KV-cache Precision. In MLX (Hannun et al., 2023) (August 2026) the Self-Dot-Product-Attention (SDPA) relies on a Flash kernel for bf16 weights, but for quantized matrices SDPA is decomposed into a set of quantized matmul that do not benefit from Flash: this increases the cost of Self Attention in low-precision for long sequences as illustrated in Figure 8.

Model Weight Precision. As shown in Figure 7, for fixed KV-cache precision and a fixed sequence length (around 30K), the weight precision conserve a massive impact on the total throughput and total memory footprint. This axis is orthogonal to KV-cache precision.

MLA rank. As shown in Figure 9 the Decode Throughput increases when MLA rank is reduced, as one would expect.