RAQ-MLA: ROLE-AWARE QUANTIZATION FOR MULTI-HEAD LATENT ATTENTION CACHES

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-Head Latent Attention (MLA) has been adopted by recent frontier large language models. Instead of per-head keys and values, MLA caches a compressed KV latent that is up-projected into keys and values at decode time, together with a narrow positional key that carries position information into the attention score. Existing KV-cache quantization methods are designed around the keys and values of standard attention and do not account for what MLA actually caches, which costs accuracy at low bit widths. We analyze the MLA cache both in activation distribution and in how quantization error propagates to the attention output, and show that the two cached tensors play different roles: the latent mainly supplies the values, whereas the positional key dominates the attention score. We therefore propose **RAQ-MLA** (Role-Aware Quantization for Multi-Head Latent Attention Caches), which fits an invertible transform to each cached tensor against the error that tensor delivers to the attention output: attention-weighted value error for the latent and score perturbation for the positional key. Without training, RAQ-MLA quantizes the full MLA cache to about 2.5 bits per element, a $6.5\times$ increase in theoretical KV-cache capacity, while remaining nearly lossless on reasoning and coding tasks across four MLA models.

1 INTRODUCTION

Long-horizon agents have made ultra-long-context inference a mainstream serving workload (Yao et al., 2023; Yang et al., 2024; DeepSeek-AI, 2026). Serving it requires more than efficient long-sequence computation: the key-value (KV) cache must be held for the entire lifetime of a request and grows linearly with both context length and concurrency. At runtime, the cache competes with batch size for HBM capacity (Kwon et al., 2023); when persisted for prefix reuse, it consumes host memory, SSD capacity, and interconnect bandwidth (Qin et al., 2026; Zheng et al., 2024). Both pressures cap serving throughput and raise deployment cost, and both grow as agents run longer and at higher concurrency.

To reduce this footprint, Multi-Head Latent Attention (MLA) (DeepSeek-AI, 2024) compresses each token's key and value into a single low-dimensional latent $\mathbf{c}_t^{\mathrm{KV}}$, which is cached and up-projected into keys and values at decode time, alongside a narrow decoupled positional key $\mathbf{k}_t^{\mathrm{R}}$. MLA has been adopted by recent frontier models, increasingly together with sparse attention (DeepSeek-AI, 2025; GLM, 2026). Sparse attention, however, only selects which cached tokens each query attends to; the full cache is still retained. MLA thus shrinks the cache per token, but neither it nor sparse attention changes its linear growth with context length: at the one-million-token context length of recent models, a single sequence still occupies tens of GiB of cache. Compressing what MLA actually stores therefore remains necessary.

KV-cache quantization is the other line of work that reduces cache size, lowering the average bits per stored element without any training (Hooper et al., 2024; Liu et al., 2024; Zhou et al., 2026; Su et al., 2025). However, most prior methods (Liu et al., 2024; Hooper et al., 2024) design their granularity, precision, and transforms around the keys and values of standard attention. MLA caches neither: it stores the compressed latent $\mathbf{c}_t^{\mathrm{KV}}$ and the decoupled positional key $\mathbf{k}_t^{\mathrm{R}}$, and recovers the content key and value from the latent only at use time. Existing MLA cache quantizers (Zhang et al., 2026; Muller et al., 2026) keep the latent at four or more bits and leave $\mathbf{k}^{\mathrm{R}}$ in full precision. In this

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.

paper, we aim to quantize the full MLA cache, including the positional key, to extremely low bit widths (about 2.5 bits per cached element on average).

We analyze the two cached tensors both in distribution and in how their quantization error propagates through the MLA computation. Fixed-channel outliers appear in the decoupled positional key but not in the compressed latent (Figure 2), and the positional key carries 92.5% of the attention-score attribution, most of it in a few outlier channels. As a result, the two tensors reach the output through different routes. At 4 bits, the latent's 12.3% error in the cache falls to 1.7% at the attention output through the content key but remains 8.3% through the value, whereas the positional key's error rises from 9.1% to 14.7%. The latent therefore mainly plays the value role in MLA, while the tensor that sets the score is the one existing MLA methods exempt from quantization.

Motivated by this, we propose RAQ-MLA (Role-Aware Quantization for Multi-Head Latent Attention Caches). It quantizes the latent per token and the positional key per channel, following the distribution each exhibits, and stores each in an invertible basis fitted against the error that tensor delivers to the attention output. Both bases are fitted under a single objective, a quadratic form weighted per token by a *role metric* built from how attention consumes the tensor: for the latent, the value up-projection weighted by how strongly each token is attended; for the positional key, the directions of the queries that read it. We solve it by alternating minimization, requiring neither gradients through the model nor fine-tuning. Both transforms stay inside the compact spaces MLA already works in, 512×512 for the latent and 64×64 for the positional key, so RAQ-MLA improves low-bit accuracy at negligible extra computation.

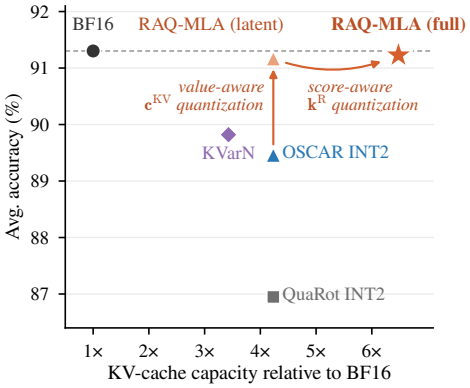


Figure 1: Cache capacity against accuracy on GLM-4.7-Flash.

Our contributions are:

- **Role analysis of the MLA cache.** We show that the two cached tensors play different roles in attention: error in the compressed latent reaches the output mainly through the value readout, whereas the decoupled positional key dominates the attention score and amplifies its own quantization error (Section 3).
- **Role-aware quantization.** We propose RAQ-MLA, a training-free quantization method designed for the MLA cache. It pairs per-token quantization with a value-aware transform for the latent, and per-channel quantization with a score-aware transform for the positional key. Both transforms are fitted under a single role-aware objective by alternating minimization (Section 4).
- **Full-cache quantization at about 2.5 bits.** On four MLA models from the GLM, DeepSeek, and Moonlight families, RAQ-MLA quantizes both cached tensors to about 2.5 bits per element, a $6.5\times$ increase in theoretical KV-cache capacity, while staying within 0.5 points of BF16 on reasoning and coding (Figure 1).

2 Background: Multi-Head Latent Attention

Multi-Head Latent Attention (MLA) (DeepSeek-AI, 2024) was introduced to shrink the KV cache of multi-head attention; we briefly recall its computation and the cache it stores. Per-token activations are row vectors, a layer has n_{h} heads of width d_{h} (used for both the content key and the value for simplicity), and $\mathbf{h}_t$ is the attention input at token t.

Low-rank key-value joint compression. MLA compresses keys and values into a shared latent and recovers content (non-positional) keys and values by per-head up-projections:

$$\mathbf{c}_t^{\mathrm{KV}} = \mathrm{RMSNorm}\big(\mathbf{h}_t(\mathbf{W}^{\mathrm{DKV}})^\top\big), \qquad \mathbf{k}_{t,h}^{\mathrm{C}} = \mathbf{c}_t^{\mathrm{KV}}(\mathbf{W}_h^{\mathrm{UK}})^\top, \qquad \mathbf{v}_{t,h}^{\mathrm{C}} = \mathbf{c}_t^{\mathrm{KV}}(\mathbf{W}_h^{\mathrm{UV}})^\top, \quad (1)$$

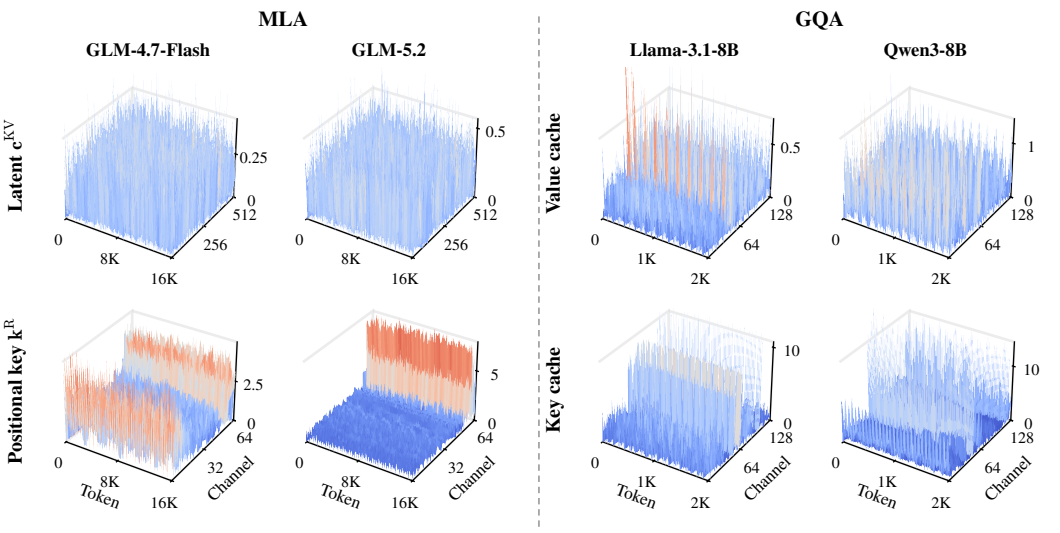


Figure 2: **Fixed-channel outliers appear in the decoupled positional key but not in the compressed KV latent.** The positional key behaves like the Key cache of standard attention, and the compressed latent like the Value cache.

where $\mathbf{c}_t^{\mathrm{KV}} \in \mathbb{R}^{d_c}$ is the compressed KV latent with $d_c \ll n_h d_h$, and $\mathbf{W}_h^{\mathrm{UK}}, \mathbf{W}_h^{\mathrm{UV}} \in \mathbb{R}^{d_h \times d_c}$ are the head-h blocks of the up-projections. $\mathbf{k}_{t,h}^{\mathrm{C}}$ and $\mathbf{v}_{t,h}^{\mathrm{C}}$ are not cached: at inference, $\mathbf{W}_h^{\mathrm{UK}}$ is absorbed into the query side and $\mathbf{W}_h^{\mathrm{UV}}$ into the output side, so neither is ever materialized.

Decoupled rotary position embedding. MLA utilizes a *decoupled RoPE* strategy to carry position information: per-head positional queries $\mathbf{q}_{t,h}^{\mathrm{R}} \in \mathbb{R}^{d^{\mathrm{R}}}$ and a single positional key $\mathbf{k}_t^{\mathrm{R}} \in \mathbb{R}^{d^{\mathrm{R}}}$ *shared by all* n_h *heads* are rotated by RoPE, while $\mathbf{k}_{t,h}^{\mathrm{C}}$ remains the content key. With a low-rank query latent $\mathbf{c}_t^{\mathrm{Q}}$,

$$\mathbf{c}_t^{\mathrm{Q}} = \mathrm{RMSNorm}\big(\mathbf{h}_t(\mathbf{W}^{\mathrm{DQ}})^\top\big), \qquad \mathbf{q}_{t,h}^{\mathrm{C}} = \mathbf{c}_t^{\mathrm{Q}}(\mathbf{W}_h^{\mathrm{UQ}})^\top, \tag{2}$$

$$\mathbf{q}_{t,h}^{\mathrm{R}} = \mathrm{RoPE}\big(\mathbf{c}_t^{\mathrm{Q}}(\mathbf{W}_h^{\mathrm{QR}})^\top\big), \qquad \mathbf{k}_t^{\mathrm{R}} = \mathrm{RoPE}\big(\mathbf{h}_t(\mathbf{W}^{\mathrm{KR}})^\top\big), \tag{3}$$

$$\mathbf{q}_{t,h} = \big[\mathbf{q}_{t,h}^{\mathrm{C}};\, \mathbf{q}_{t,h}^{\mathrm{R}}\big], \qquad \mathbf{k}_{t,h} = \big[\mathbf{k}_{t,h}^{\mathrm{C}};\, \mathbf{k}_t^{\mathrm{R}}\big], \tag{4}$$

where $[\cdot;\cdot]$ denotes concatenation.

Attention computation. With the concatenated queries and keys, MLA computes

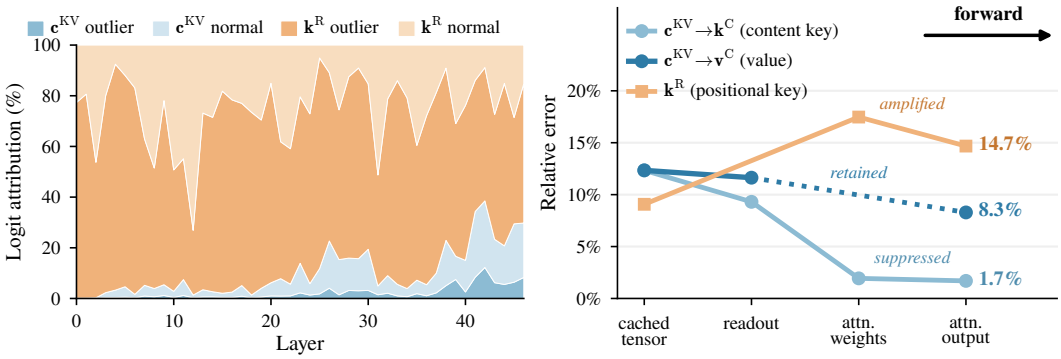
$$\mathbf{o}_{q,h} = \sum_{t=1}^{q} S_{h,q,t}\, \mathbf{v}_{t,h}^{\mathrm{C}}, \qquad S_{h,q,t} = \mathrm{Softmax}_t\left(\frac{\mathbf{q}_{q,h}^{\mathrm{C}}(\mathbf{k}_{t,h}^{\mathrm{C}})^\top + \mathbf{q}_{q,h}^{\mathrm{R}}(\mathbf{k}_t^{\mathrm{R}})^\top}{\sqrt{d_h + d^{\mathrm{R}}}}\right), \tag{5}$$

where $S_{h,q,t}$ is the weight head h assigns to cached token t when answering the query at position q, and the logit splits into a content term and a positional term. The layer output is assembled by the usual output projection over heads.

Inference cache. The persisted cache therefore holds two tensors per layer: the compressed latent $\mathbf{c}_t^{\mathrm{KV}}$ of width d_c, and the decoupled positional key $\mathbf{k}_t^{\mathrm{R}}$ of width d^{R}, shared across heads. Most MLA models set $d_c = 512$ and $d^{\mathrm{R}} = 64$ (DeepSeek-AI, 2024; DeepSeek-AI et al., 2024; DeepSeek-AI, 2025; Liu et al., 2025a; GLM, 2026), so the positional key accounts for $64/576 = 11.1\%$ of the cached elements.

3 Motivation

We analyze the MLA cache from a quantization perspective to decide, for each cached tensor, how it should be quantized and what its quantized bits must preserve. The analysis proceeds in three



(a) (b)

Figure 3: **The positional key dominates the attention score, while the latent mainly plays the value role.** (a) Logit attribution to the outlier and normal channels of the latent $\mathbf{c}^{KV}$ and the positional key $\mathbf{k}^{R}$, per layer. (b) Relative error at each stage under INT4, with the latent quantized per token and the positional key per channel.

steps: the distribution of each tensor, how its quantization error propagates to the attention output, and its contribution to the attention score, which explains that propagation. Unless stated otherwise, all analyses use GLM-4.7-Flash on calibration prompts.

How are the two cached tensors distributed? As shown in Figure 2, channel structure separates the two cached tensors. We compare $\mathbf{c}^{KV}$ and $\mathbf{k}^{R}$ in GLM-4.7-Flash and GLM-5.2 with the Key and Value caches of standard (non-MLA) attention in Llama-3.1-8B (Grattafiori et al., 2024) and Qwen3-8B (Yang et al., 2025), and each aligns with one of the two references. $\mathbf{k}^{R}$ carries the channel-aligned outliers that conventional Key caches exhibit. $\mathbf{c}^{KV}$ instead resembles a Value cache, spreading its magnitude broadly, with no fixed channel standing out. The first design decision in RAQ-MLA follows: the positional key is quantized per channel, which keeps its outlier channels out of a shared scale, and the latent per token, which confines the error to the token being quantized (Liu et al., 2024). This granularity reduces the quantization error of each cached tensor, but it does not target the errors that matter at the attention output.

How does quantization error reach the output? We quantize the latent per token and the positional key per channel, both at 4 bits, and measure the relative error stage by stage (Figure 3b). The latent's two readouts start out alike: right after up-projection, $\mathbf{k}^{C}$ and $\mathbf{v}^{C}$ carry comparable relative error. Only the content key enters the attention weights, and there its error has shrunk to 1.9%, an order of magnitude below the positional key's 17.5%. At the attention output, the latent's 12.3% error in the cache has fallen to 1.7% through the content key but remains 8.3% through the value, whereas the positional key's error has risen from 9.1% to 14.7%.

Why do the two routes differ? The channel structure of Figure 2 accounts for this split. The outlier channels of $\mathbf{k}^{R}$ dominate the attention score: pooled over all 47 layers, the positional term carries 92.5% of the logit attribution (Appendix D) against 7.5% for the content term, and the outlier channels of $\mathbf{k}^{R}$, a tenth of its 64, carry 63.8% on their own (Figure 3a). Because the content term contributes so little to the logit, perturbing it barely moves the attention weights; the positional term sets them, so its error is amplified into the attention weights and carried on to the output. Within the latent, then, the value readout carries most of its error through to the output, while the attention score is dominated by $\mathbf{k}^{R}$.

Why do we choose a learnable transform for the MLA cache? These roles call for a storage basis chosen by how attention reads each tensor rather than by its distribution: the latent's error should stay out of the directions the value up-projection reads, and the positional key's out of the directions its queries probe. We therefore propose RAQ-MLA, which learns a basis for each cached tensor against the error that tensor delivers to the attention output. The MLA architecture keeps such transforms cheap. First, both cached tensors are low-dimensional ($d_c = 512$ and $d^{R} = 64$), so

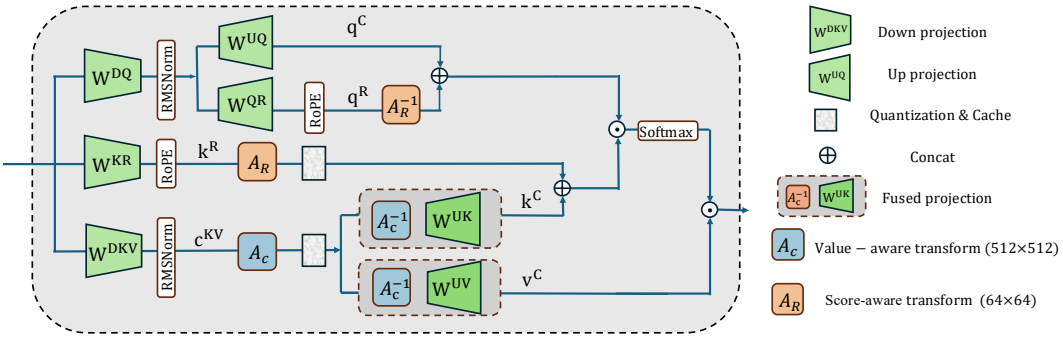


Figure 4: **RAQ-MLA at one MLA layer.** A_{c} is applied to the latent after RMSNorm and A_{R} to the positional key after RoPE, so both cached tensors are quantized in a learned basis. On the read path A_{c}^{-1} is fused into $\mathbf{W}^{\mathrm{UK}}$ and $\mathbf{W}^{\mathrm{UV}}$, while the positional key is compensated on the query side.

the two transforms together cost $d_{\mathrm{c}}^2 + (d^{\mathrm{R}})^2$ coefficients per layer (0.08% of the total parameters of GLM-4.7-Flash). Second, as detailed in Section 4.5, the cached tensors never have to be transformed back at decode time: for the latent, the inverse folds into $\mathbf{W}^{\mathrm{UK}}$ and $\mathbf{W}^{\mathrm{UV}}$; for the positional key, the compensation is applied to the query $\mathbf{q}^{\mathrm{R}}$, which is never cached, and cancels the transform in the attention score.

4 RAQ-MLA

4.1 Formulation

RAQ-MLA quantizes the latent and the positional key in learned invertible bases A_{c} and A_{R} (Figure 4), both fitted under one objective. We state it once for a generic cached tensor; a separate transform is fitted for each cached tensor at every layer. Write $X \in \mathbb{R}^{T\times d}$ for a cached tensor with rows $\mathbf{x}_t$, let $\mathcal{Q}$ be the quantize–dequantize operator, and let A denote its transform (A_{c} or A_{R}), so that the dequantized tensor is $\widehat{X} = \mathcal{Q}(XA)A^{-1}$. Without quantization $\widehat{X} = X$, so A changes only how the quantization error is distributed, not the model. We fit A by minimizing

$$F(A) := \mathcal{D}\big(\mathcal{Q}(XA)A^{-1}\big), \qquad \mathcal{D}(\widehat{X}) = \sum_t (\widehat{\mathbf{x}}_t - \mathbf{x}_t) M_t (\widehat{\mathbf{x}}_t - \mathbf{x}_t)^\top, \qquad M_t \succeq 0, \tag{6}$$

a token-weighted quadratic in which the *role metric* $M_t \in \mathbb{R}^{d\times d}$ encodes how attention reads the tensor at token t: the same error costs more in directions attention reads strongly. M_t plays the role of the Hessian in second-order post-training quantization objectives (Frantar & Alistarh, 2022; Frantar et al., 2023), with one difference: a weight matrix has a single Hessian per layer, whereas M_t comes from how attention consumes the cached tensor and therefore varies from token to token. Sections 4.2 and 4.3 derive M_t for the two cached tensors from calibration statistics, together with the approximations each involves.

4.2 Value-aware quantization of the compressed KV latent

Let $C^{\mathrm{KV}} \in \mathbb{R}^{T\times d_c}$ collect the cached latent vectors. The latent is quantized as $Z_{\mathrm{c}} = \mathcal{Q}(C^{\mathrm{KV}}A_{\mathrm{c}})$ and read as $\widehat{C}^{\mathrm{KV}} = Z_{\mathrm{c}}A_{\mathrm{c}}^{-1}$, with A_{c}^{-1} folded into the up-projections at inference (Section 4.5).

Value-aware objective. We model the latent's distortion along the value path and neglect the content-key path: as demonstrated in Section 3, the latent's error reaches the attention output as 1.7% through the content key against 8.3% through the value. Write $\mathbf{e}_t = \widehat{\mathbf{c}}_t^{\mathrm{KV}} - \mathbf{c}_t^{\mathrm{KV}}$ for the latent's quantization error at token t; it perturbs the value that head h reads from token t by $\Delta\mathbf{v}_{t,h}^{\mathrm{C}} = \mathbf{e}_t(\mathbf{W}_h^{\mathrm{UV}})^\top$. Let $S_h \in \mathbb{R}^{T\times T}$ collect the causal attention weights $S_{h,q,t}$ of head h from Equation (5), held at the values the unquantized model produces; holding S_h fixed is exactly what sets the content-key path aside. The attention output of query q at head h is then perturbed by $\Delta\mathbf{o}_{q,h} = \sum_t S_{h,q,t}\Delta\mathbf{v}_{t,h}^{\mathrm{C}}$, and

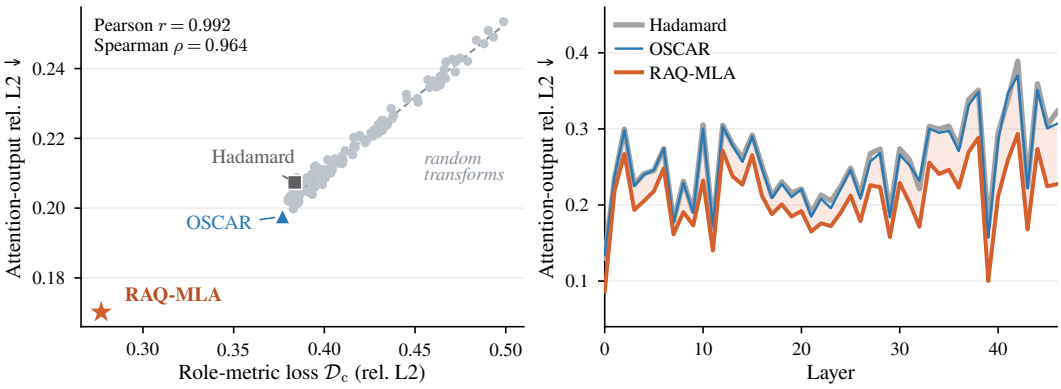


Figure 5: **Evaluation of the latent role metric.** (a) At layer 23, the role-metric loss and the exact attention-output error it is meant to predict agree across 128 random transforms (Pearson $r = 0.992$, Spearman $\rho = 0.964$), and RAQ-MLA minimizes both. (b) Layer-wise attention-output error of each method on held-out validation prompts; shading marks the reduction from the better baseline to RAQ-MLA.

summing over queries and heads gives

$$\sum_{h,q} \|\Delta \mathbf{o}_{q,h}\|_2^2 = \sum_{h,t,t'} [S_h^\top S_h]_{t,t'} \langle \Delta \mathbf{v}_{t,h}^{\mathrm{C}}, \Delta \mathbf{v}_{t',h}^{\mathrm{C}} \rangle. \tag{7}$$

Because the latent is quantized per token, errors at different tokens are nearly uncorrelated (Appendix Figure 9), so each off-diagonal term is small. We therefore keep only the diagonal $\gamma_{t,h} = [S_h^\top S_h]_{t,t} = \sum_q S_{h,q,t}^2$, which we call the *attention exposure* of token t at head h: how strongly head h reads token t over all queries. This puts the value-path distortion in the form of Equation (6),

$$\mathcal{D}_{\mathrm{c}}(\widehat{C}^{\mathrm{KV}}) := \sum_{h,t} \gamma_{t,h} \left\| \Delta \mathbf{v}_{t,h}^{\mathrm{C}} \right\|_2^2 = \sum_t \mathbf{e}_t M_t^{\mathrm{c}} \mathbf{e}_t^\top, \tag{8}$$

$$M_t^{\mathrm{c}} = \sum_h \gamma_{t,h} \, (\mathbf{W}_h^{\mathrm{UV}})^\top \mathbf{W}_h^{\mathrm{UV}} \in \mathbb{R}^{d_{\mathrm{c}} \times d_{\mathrm{c}}}. \tag{9}$$

The value up-projection sets which directions of the latent are read at all, and $\gamma_{t,h}$ scales that geometry by how strongly head h reads token t. To test whether this metric tracks the error that actually reaches the output, we score 128 random transforms at layer 23 by both $\mathcal{D}_{\mathrm{c}}$ and the exact attention-output error (Figure 5a). The two agree almost linearly (Pearson $r = 0.992$, Spearman $\rho = 0.964$), so minimizing $\mathcal{D}_{\mathrm{c}}$ is a faithful proxy for minimizing the attention-output error. Fitting A_{c} this way lowers the attention-output error below Hadamard and OSCAR at every one of the 47 layers, and the reduction carries over to held-out validation prompts (Figure 5b).

4.3 Score-aware quantization of the decoupled positional key

Section 3 showed that the positional key carries 92.5% of the attention-score attribution and that its error is amplified on the way to the output, from 9.1% in the cache to 14.7% at the attention output. Let $K^{\mathrm{R}} \in \mathbb{R}^{T \times d^{\mathrm{R}}}$ collect the positional keys after RoPE. The positional key is quantized as $Z_{\mathrm{R}} = \mathcal{Q}(K^{\mathrm{R}} A_{\mathrm{R}})$ and read as $\widehat{K}^{\mathrm{R}} = Z_{\mathrm{R}} A_{\mathrm{R}}^{-1}$; since the positional query is never cached, A_{R}^{-1} is compensated on the query side at inference (Section 4.5).

Score-aware objective. We take the perturbation of the pre-softmax positional logit as the distortion. Writing $\mathbf{e}_t^{\mathrm{R}} = \widehat{\mathbf{k}}_t^{\mathrm{R}} - \mathbf{k}_t^{\mathrm{R}}$, this again gives the form of Equation (6),

$$\mathcal{D}_{\mathrm{R}}(\widehat{K}^{\mathrm{R}}) := \sum_{h,q} \sum_{t \le q} \left(\mathbf{q}_{q,h}^{\mathrm{R}} (\mathbf{e}_t^{\mathrm{R}})^\top\right)^2 = \sum_t \mathbf{e}_t^{\mathrm{R}} M_t^{\mathrm{R}} (\mathbf{e}_t^{\mathrm{R}})^\top, \tag{10}$$

$$M_t^{\mathrm{R}} = \sum_h \sum_{q \ge t} (\mathbf{q}_{q,h}^{\mathrm{R}})^\top \mathbf{q}_{q,h}^{\mathrm{R}} \ \in \mathbb{R}^{d^{\mathrm{R}} \times d^{\mathrm{R}}}. \tag{11}$$

M_t^{R} is the second moment of the positional queries that read token t, so it penalizes error along the directions those queries probe most. Across 128 random transforms at layer 23, $\mathcal{D}_{\mathrm{R}}$ also tracks the exact attention-output error (Pearson $r = 0.852$, Spearman $\rho = 0.847$; Appendix Figure 10).

4.4 Optimization

Both objectives are instances of Equation (6) and are fitted by the same procedure. Because $\mathcal{Q}$ rounds to a grid, F is discontinuous in A and the gradient through the rounding vanishes almost everywhere, so rather than relying on straight-through estimates we alternate between the quantized tensor and the transform. Holding $Z^{(r)} = \mathcal{Q}(XA^{(r)})$ fixed, the distortion becomes a convex quadratic in the inverse transform $B = A^{-1}$, which we update with a proximal term and then invert,

$$A^{(r+1)} = \left(\arg\min_B \ \mathcal{D}\Big(Z^{(r)} B\Big) + \lambda \left\| B - (A^{(r)})^{-1} \right\|_F^2 \right)^{-1}, \tag{12}$$

after which X is requantized in the new basis. The proximal term keeps the update close to the current inverse transform and makes the inner problem strongly convex for $\lambda > 0$. M_t is instantiated by Equation (9) for the latent and by Equation (11) for the positional key, and no gradient is taken through the model. Algorithm 1 in Appendix B gives the full procedure.

Theorem 4.1. *Let $\lambda > 0$, write $B^{(r)} = (A^{(r)})^{-1}$ and $\Delta_r = \|B^{(r+1)} - B^{(r)}\|_F^2$, so that $F(A^{(r)}) = \mathcal{D}(Z^{(r)} B^{(r)})$ is the objective value at round r. If each $B^{(r+1)}$ is invertible and each requantization does not increase the distortion at the current decoder, $\mathcal{D}(Z^{(r+1)} B^{(r+1)}) \le \mathcal{D}(Z^{(r)} B^{(r+1)})$, then $F(A^{(r+1)}) \le F(A^{(r)}) - \lambda \Delta_r$; hence $F(A^{(r)})$ converges, $\Delta_r \to 0$, and $F(A^{(R)}) \le F(A^{(0)})$ for every R.*

4.5 Inference

At inference, each cached tensor is written directly in its learned basis, and its inverse transform is moved to the read side. On the write path,

$$\widetilde{\mathbf{c}}_t^{\mathrm{KV}} = \mathrm{RMSNorm}\big(\mathbf{h}_t (\mathbf{W}^{\mathrm{DKV}})^\top\big)\, A_{\mathrm{c}}, \qquad \widetilde{\mathbf{k}}_t^{\mathrm{R}} = \mathrm{RoPE}\big(\mathbf{h}_t (\mathbf{W}^{\mathrm{KR}})^\top\big)\, A_{\mathrm{R}}, \tag{13}$$

and the cache stores $\mathcal{Q}(\widetilde{\mathbf{c}}_t^{\mathrm{KV}})$ and $\mathcal{Q}(\widetilde{\mathbf{k}}_t^{\mathrm{R}})$, i.e., the rows of Z_{c} and Z_{R}. On the read path, A_{c}^{-1} is folded into the up-projections and $A_{\mathrm{R}}^{-\top}$ is applied to the positional query after RoPE,

$$(\widetilde{\mathbf{W}}_h^{\mathrm{UK}})^\top = A_{\mathrm{c}}^{-1} (\mathbf{W}_h^{\mathrm{UK}})^\top, \qquad (\widetilde{\mathbf{W}}_h^{\mathrm{UV}})^\top = A_{\mathrm{c}}^{-1} (\mathbf{W}_h^{\mathrm{UV}})^\top, \qquad \widetilde{\mathbf{q}}_{q,h}^{\mathrm{R}} = \mathbf{q}_{q,h}^{\mathrm{R}} A_{\mathrm{R}}^{-\top}, \tag{14}$$

so that $Z_{\mathrm{c}} (\widetilde{\mathbf{W}}_h^{\mathrm{UK}})^\top = \widehat{C}^{\mathrm{KV}} (\mathbf{W}_h^{\mathrm{UK}})^\top$, likewise for $\mathbf{W}_h^{\mathrm{UV}}$, and $\widetilde{\mathbf{q}}_{q,h}^{\mathrm{R}} Z_{\mathrm{R}}^\top = \mathbf{q}_{q,h}^{\mathrm{R}} (\widehat{K}^{\mathrm{R}})^\top$: attention reads exactly the dequantized cache of Sections 4.2 and 4.3 without ever reconstructing it. Three small products remain at runtime, all acting on the current token and hence independent of context length: one $d_{\mathrm{c}} \times d_{\mathrm{c}}$ on the latent write path, one $d^{\mathrm{R}} \times d^{\mathrm{R}}$ on the positional-key write path, and one $d^{\mathrm{R}} \times d^{\mathrm{R}}$ per head on the query path.

5 Experiments

5.1 Setup

Models and benchmarks. We evaluate four MLA models: GLM-4.7-Flash (Zeng et al., 2025), GLM-5.2-FP8 (GLM, 2026), Moonlight-16B-A3B-Instruct (Liu et al., 2025a), and DeepSeek-Coder-V2-Lite-Instruct (DeepSeek-AI et al., 2024), all with $d_{\mathrm{c}} = 512$ and $d^{\mathrm{R}} = 64$. Benchmarks

Table 1: **Reasoning and coding tasks. BPE** is effective bits per cached element, and Δ is the mean change from BF16 over the five tasks.

Model	Method	BPE	AIME25	GPQA	MATH-500	GSM8K	HumanE	Δ
GLM-4.7	BF16	16.00	93.33	77.27	97.40	95.22	93.29	—
	KVarN INT3	4.67	90.00	73.23	**98.20**	95.60	92.07	-1.48
	QuaRot INT2	3.78	86.67	69.19	**98.20**	94.09	86.59	-4.35
	OSCAR INT2	3.78	87.78	72.06	97.40	95.75	94.31	-1.84
	RAQ-MLA	**2.47**	**92.22**	**75.25**	97.67	**96.08**	**94.92**	$\mathbf{-0.08}$
GLM-5.2	BF16	16.00	90.00	86.87	97.80	97.50	99.39	—
	QuaRot INT2	3.78	86.67	78.28	98.20	97.80	96.34	-2.85
	OSCAR INT2	3.78	**93.33**	**85.86**	**98.60**	97.62	98.58	$\mathbf{+0.49}$
	RAQ-MLA	**2.50**	92.22	84.85	97.80	**97.90**	**98.78**	0.00
			ARC-C	PIQA	MATH-500	GSM8K	HumanE	Δ
Moonlight	BF16	16.00	52.22	52.56	63.20	85.14	64.02	—
	QuaRot INT2	3.78	45.14	33.13	32.20	59.29	44.51	-20.57
	OSCAR INT2	3.78	50.77	48.04	57.13	80.42	**69.21**	-2.31
	RAQ-MLA	**2.47**	**51.79**	**52.67**	**61.80**	**83.85**	67.27	$\mathbf{+0.05}$
DS-V2-Lite	BF16	16.00	69.97	75.30	58.00	84.91	78.05	—
	QuaRot INT2	3.78	60.67	68.39	51.80	75.28	70.73	-7.87
	OSCAR INT2	3.78	**68.69**	**74.54**	**61.13**	82.74	**79.68**	$\mathbf{+0.11}$
	RAQ-MLA	**2.47**	68.60	**74.54**	60.60	**83.52**	78.25	-0.14

Table 2: **RULER-NIAH on GLM-4.7-Flash.** Each cell averages eight subtasks of 100 samples.

Method	BPE	4K	8K	16K	32K	64K	128K	Avg.
BF16	16.00	100.00	99.75	99.31	98.44	96.97	87.22	96.95
OSCAR	3.78	**99.56**	98.64	**98.34**	**96.01**	90.39	78.65	93.60
RAQ-MLA	**2.66**	99.38	**99.06**	97.96	95.74	**90.69**	**79.13**	**93.66**

cover reasoning and coding on GPQA-Diamond (Rein et al., 2023), AIME25 (Dekoninck et al., 2026), MATH-500 (Lightman et al., 2024), GSM8K (Cobbe et al., 2021), and HumanEval (Chen et al., 2021), with ARC-Challenge (Clark et al., 2018) and PIQA (Bisk et al., 2020) in place of AIME25 and GPQA-Diamond on Moonlight and DS-V2-Lite, and long-context retrieval on the eight RULER NIAH subtasks (Hsieh et al., 2024) from 4K to 128K. More details are given in Appendix E.

Baselines. Among existing KV-cache quantization methods, KVarN (Muller et al., 2026) reports MLA experiments, compressing the latent with a Hadamard rotation and a dual-scaling variance normalization while leaving the decoupled positional key in full precision. We also implement two standard KV-cache transform methods, QuaRot (Ashkboos et al., 2024) and OSCAR (Zhou et al., 2026), in the same setting: transform the latent cache, quantize it to low bits, and leave the decoupled positional key in full precision.

Implementation. The transforms are fitted on 128 RedPajama (Together Computer, 2023) sequences of 64K tokens. The latent uses asymmetric per-token INT2 with group size 128 and a clipping ratio of 0.94, and $\mathbf{k}^{\mathrm{R}}$ uses asymmetric per-channel INT4 within each cache page. Following KVarN, a cache page is quantized only once it is full, and the open page stays in BF16 for all quantized methods. We report effective bits per cached element (BPE), which include the FP16 scale and minimum; the full cache costs 2.47 BPE, or 2.50 on GLM-5.2, whose 64-token pages double the metadata of $\mathbf{k}^{\mathrm{R}}$.

5.2 Main results

Reasoning and coding. Table 1 shows that RAQ-MLA keeps BF16 accuracy at the lowest bit width of all methods. At about 2.5 bits, it stays within 0.5 points of BF16 on all four models (-0.08,

Table 3: **Component ablation on RAQ-MLA**, GLM-4.7-Flash.

Step	Latent	$\mathbf{k}^{\mathrm{R}}$	BPE	AIME25	GPQA
A	naive	BF16	3.78	83.33	71.72
B	clip + tail	BF16	3.78	86.67	76.26
C	Hadamard	BF16	3.78	86.67	69.19
D	OSCAR	BF16	3.78	87.78	72.06
E	value-aware	BF16	3.78	92.22	76.09
F	value-aware	direct	2.47	87.78	74.24
G	value-aware	score-aware	2.47	92.22	75.25

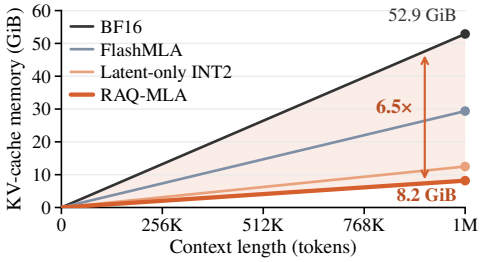


Figure 6: **Theoretical KV-cache memory** of one GLM-4.7-Flash sequence.

0.00, $+0.05$, -0.14), whereas QuaRot and OSCAR store 3.78 BPE and KVarN 4.67 (Figure 1). On GLM-4.7, every baseline loses more despite its larger cache: QuaRot 4.35 points, OSCAR 1.84 and KVarN 1.48; QuaRot also falls by 20.57 points on Moonlight and 7.87 on DS-V2-Lite. OSCAR, the strongest baseline, is slightly ahead on GLM-5.2 and DS-V2-Lite ($+0.49$ and $+0.11$) but loses 2.31 points on Moonlight, and it needs about $1.5\times$ the cache of RAQ-MLA.

Long-context retrieval. Table 2 reports RULER NIAH on GLM-4.7-Flash. Here RAQ-MLA keeps a wider quantizer at the first and last two layers (an INT4 latent and an INT8 positional key), for 2.66 BPE on average. It matches OSCAR on average (93.66 against 93.60) with a 29.6% smaller cache (2.66 against 3.78 BPE), and the two stay within 0.5 points of each other at every length. Both trail a BF16 control that itself falls from 96.97 at 64K to 87.22 at 128K.

5.3 Ablation Study

Component ablation. Table 3 adds the components of RAQ-MLA one at a time. Following OSCAR and KVarN, B adds clipping and keeps the open cache page in BF16. On top of B, C–E compare three latent transforms, and our value-aware transform is the most accurate on both tasks: 92.22 on AIME25 and 76.09 on GPQA, against 87.78 and 72.06 for OSCAR and 86.67 and 69.19 for Hadamard. When $\mathbf{k}^{\mathrm{R}}$ is quantized as well, our score-aware transform (G) improves over direct INT4 (F) from 87.78 to 92.22 on AIME25 and from 74.24 to 75.25 on GPQA at the same 2.47 BPE.

5.4 Efficiency

RAQ-MLA reduces the KV cache by $6.5\times$ compared with BF16 and by $3.6\times$ compared with the FP8 cache of FlashMLA (Li & Liu, 2025) (Figure 6). On two H20 GPUs with TP2 and batch size 1, the decode kernel at 512 context drops from 17.4 to 12.6 μs per layer against FlashMLA ($1.38\times$), and the time per output token at 512 input and 512 output tokens from 6.56 to 5.97 ms ($1.10\times$), because each decode step reads less cache (Table 4). TPOT stays 6.8–9.0% lower for outputs of 1K to 8K tokens, and applying A_{c} inside the cache-write kernel costs 1.7–2.4 μs per layer, under 2% of TPOT at batch size 1 (Appendix F).

Table 4: **Kernel latency and TPOT on H20 GPUs.**

Method	Kernel (μs/layer)	TPOT (ms)
FlashMLA	17.4	6.56
RAQ-MLA	**12.6**	**5.97**
Speedup	**1.38**×	**1.10**×

6 Conclusion

We showed that the two tensors of the MLA cache play different roles in attention: the decoupled positional key dominates the attention score, whereas error in the latent reaches the output mainly through the value readout. RAQ-MLA therefore stores each tensor in an invertible basis fitted, without training, against the error it delivers to the attention output. It compresses the full MLA cache, positional key included, to about 2.5 bits per element, a $6.5\times$ capacity gain over BF16, within 0.5 points of BF16 on reasoning and coding across the GLM, DeepSeek, and Moonlight families.

AI USE STATEMENT

We used a generative AI assistant (Claude by Anthropic, accessed through the Cursor editor) only for auxiliary tasks: polishing the wording and checking the grammar of text written by the authors; formatting the LaTeX source, tables, and references, and checking them against the submission requirements; and restyling figures (fonts, tick labels, and layout) through the plotting scripts, without changing the plotted data. The core content of this work is the authors' own: the research questions, the analysis of the MLA cache, the RAQ-MLA method and its derivations, the design, implementation, and execution of all experiments, and the interpretation of the results. The authors reviewed every AI-assisted edit and verified that no figure's data were altered. We take full responsibility for the entire content of this paper, including any text, figures, or code produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 4 specifies RAQ-MLA: the objective in Equation (6), the role metrics of the latent and the positional key (Sections 4.2 and 4.3), the alternating fit (Section 4.4), and how the fitted transforms are folded into inference (Section 4.5). Appendix B gives pseudocode for fitting and decoding (Algorithms 1 and 2) and the proof of Theorem 4.1. Section 5.1 lists the evaluated models, benchmarks, baselines, and calibration data. Appendix E.1 gives the quantizer settings and the accounting of effective bits per element (Table 5), and Appendix E.2 gives the serving stack, evaluation toolkits, and sampling configuration (Table 6). The hardware and software settings of the latency measurements are stated with each result in Section 5.4 and Appendix F.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/b5b939436789f76f08b9d0da5e81af7c-Abstract-Conference.html.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code, 2021.

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

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

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *CoRR*, abs/2405.04434, 2024. doi: 10.48550/ARXIV.2405.04434. URL https://doi.org/10.48550/arXiv.2405.04434.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models. *CoRR*, abs/2512.02556, 2025. doi: 10.48550/ARXIV.2512.02556. URL https://doi.org/10.48550/arXiv.2512.02556.

DeepSeek-AI. Deepseek harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026.

DeepSeek-AI, Qihao Zhu, Daya Guo, et al. DeepSeek-Coder-V2: Breaking the barrier of closed-source models in code intelligence, 2024. URL https://arxiv.org/abs/2406.11931.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: MathArena as an evaluation platform for mathematics with LLMs, 2026. URL https://arxiv.org/abs/2605.00674.

Haojie Duanmu, Zhihang Yuan, Xiuhong Li, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. SKVQ: sliding-window key and value cache quantization for large language models. *CoRR*, abs/2405.06219, 2024. doi: 10.48550/ARXIV.2405.06219. URL https://doi.org/10.48550/arXiv.2405.06219.

Elias Frantar and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/1caf09c9f4e6b0150b06a07e77f2710c-Abstract-Conference.html.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

GLM. GLM-5: from vibe coding to agentic engineering. *CoRR*, abs/2602.15763, 2026. doi: 10.48550/ARXIV.2602.15763. URL https://doi.org/10.48550/arXiv.2602.15763.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length LLM inference with KV cache quantization. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/028fcbcf85435d39a40c4d61b42c99a4-Abstract-Conference.html.

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? *CoRR*, abs/2404.06654, 2024. doi: 10.48550/ARXIV.2404.06654. URL https://doi.org/10.48550/arXiv.2404.06654.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Jiashi Li and Shengyu Liu. FlashMLA: Efficient multi-head latent attention kernels. https://github.com/deepseek-ai/FlashMLA, 2025.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *MLSys*, 2024. URL https://arxiv.org/abs/2306.00978.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, et al. Muon is scalable for llm training, 2025a. URL https://arxiv.org/abs/2502.16982.

Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. SpinQuant: LLM quantization with learned rotations. In *International Conference on Learning Representations*, 2025b.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) 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 A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32332–32344. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/liu24bz.html.

Yuexiao Ma, Huixia Li, Xiawu Zheng, Feng Ling, Xuefeng Xiao, Rui Wang, Shilei Wen, Fei Chao, and Rongrong Ji. AffineQuant: Affine transformation quantization for large language models. In *International Conference on Learning Representations*, 2024.

Lorenz K. Muller, Philippe Bich, Chiara Boretti, Hyun-Min Chang, Jiawei Zhuang, and Lukas Cavigelli. KVarN: Variance-normalized KV-cache quantization mitigates error accumulation in reasoning tasks, 2026. URL https://arxiv.org/abs/2606.03458.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: A kvcache-centric disaggregated architecture for LLM serving. *ACM Trans. Storage*, 22(4):37:1–37:38, 2026. doi: 10.1145/3773772. URL https://doi.org/10.1145/3773772.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. *CoRR*, abs/2311.12022, 2023. doi: 10.48550/ARXIV.2311.12022. URL https://doi.org/10.48550/arXiv.2311.12022.

Zunhai Su, Hanyu Wei, Zhe Chen, Wang Shen, Linge Li, Huangqi Yu, and Kehong Yuan. Rotatekv: Accurate and robust 2-bit KV cache quantization for llms via outlier-aware adaptive rotations. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025*, pp. 6200–6208. ijcai.org, 2025. doi: 10.24963/IJCAI.2025/690. URL https://doi.org/10.24963/ijcai.2025/690.

Zunhai Su, Rui Yang, Chao Zhang, Yaxiu Liu, Yifan Zhang, Wei Wu, Jing Xiong, Dayou Du, Xialie Zhuang, Yulei Qian, Yuchen Xie, Yik-Chung Wu, Hongxia Yang, and Ngai Wong. OScaR: The occam's razor for extreme KV cache quantization in LLMs and beyond, 2026. URL https://arxiv.org/abs/2605.19660.

Yuxuan Sun, Ruikang Liu, Haoli Bai, Han Bao, Kang Zhao, Yuening Li, Jiaxin Hu, Xianzhi Yu, Lu Hou, Chun Yuan, Xin Jiang, Wulong Liu, and Jun Yao. FlatQuant: Flatness matters for LLM quantization. In *International Conference on Machine Learning*, pp. 57587–57613, 2025.

Together Computer. RedPajama: An open source recipe to reproduce LLaMA training dataset. https://github.com/togethercomputer/RedPajama-Data, April 2023.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *ICML*, 2023. URL https://arxiv.org/abs/2211.10438.

An Yang, Anfeng Li, Baosong Yang, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Aohan Zeng, Xin Lv, Qinkai Zheng, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Yifan Zhang, Zunhai Su, Shuhao Hu, Rui Yang, Wei Wu, Yulei Qian, Yuchen Xie, and Xunliang Cai. Snapmla: Efficient long-context MLA decoding via hardware-aware FP8 quantized pipelining. *CoRR*, abs/2602.10718, 2026. doi: 10.48550/ARXIV.2602.10718. URL https://doi.org/10.48550/arXiv.2602.10718.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark W. Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Zhongzhu Zhou, Donglin Zhuang, Jisen Li, Ziyan Chen, Shuaiwen Leon Song, Ben Athiwaratkun, and Xiaoxia Wu. OSCAR: offline spectral covariance-aware rotation for 2-bit KV cache quantization. *CoRR*, abs/2605.17757, 2026. doi: 10.48550/ARXIV.2605.17757. URL https://doi.org/10.48550/arXiv.2605.17757.

A Related Work

KV-cache quantization. Post-training quantization of weights and activations is well established (Frantar et al., 2023; Lin et al., 2024; Xiao et al., 2023). As context grows, the KV cache rather than the weights dominates decoding memory, and the same tools have been carried over to the cache. Existing KV-cache quantizers choose their granularity from the statistics of the stored tensor: keys have persistent channel-wise outliers and values do not, so KIVI (Liu et al., 2024) quantizes keys per channel and values per token, keeping a short window of recent tokens in full precision. KVQuant (Hooper et al., 2024) quantizes keys per channel before RoPE, fits non-uniform datatypes, and stores a small fraction of outlier values in a sparse full-precision matrix. At two bits, SKVQ (Duanmu et al., 2024) reorders channels and clips dynamically, with a full-precision sliding window of recent tokens. Finer groups are not free, since each group stores its own scale and zero point, so we report effective bits per cached element throughout. All of these methods choose a tensor's quantization policy from that tensor's own statistics, not from how attention reads it.

Transforms for quantization. Rotating the cache before rounding is the most common remedy for outliers at low bit widths. QuaRot (Ashkboos et al., 2024) applies Hadamard rotations that leave the network output unchanged and quantizes weights, activations, and the KV cache to four bits; SpinQuant (Liu et al., 2025b) learns these rotations instead. RotateKV (Su et al., 2025) rotates keys before RoPE with a grouped-head Hadamard transform and outlier-aware channel reordering. OSCAR (Zhou et al., 2026) derives fixed rotations and clipping thresholds offline from attention-aware covariance statistics, combining their eigenbases with a Hadamard transform. OScaR (Su et al., 2026), a different method despite the similar name, targets token-norm imbalance with canalized rotation and omni-token scaling. KVarN (Muller et al., 2026) applies a Hadamard rotation followed by variance normalization along both axes of the key and value matrices, without calibration. Beyond orthogonal transforms, AffineQuant (Ma et al., 2024) and FlatQuant (Sun et al., 2025) learn general invertible affine transforms for low-bit LLM quantization by gradient descent on layer or block outputs.

MLA cache quantization. Few methods target the MLA cache directly. SnapMLA (Zhang et al., 2026) quantizes the latent to FP8 per token and keeps the positional key in BF16, after observing that FP8 raises the error of the RoPE part by an order of magnitude. The FP8 cache format of FlashMLA (Li & Liu, 2025) likewise stores the latent in FP8 and the positional key in BF16. The official vLLM

implementation of KVarN (Muller et al., 2026) supports MLA models by quantizing only the latent, to INT4. All of them leave $\mathbf{k}^{\mathrm{R}}$ in full precision.

Distinction from existing methods. RAQ-MLA is a low-bit post-training quantization method designed for the MLA cache. The general KV-cache quantizers and transforms above are designed around the keys and values of standard attention, and the MLA cache quantizers keep the latent in FP8 or INT4 and leave $\mathbf{k}^{\mathrm{R}}$ in BF16; even with a 2-bit latent, a BF16 positional key keeps the cache at 3.78 effective bits per element or more. RAQ-MLA quantizes the full MLA cache, including the positional key, to about 2.5 bits per cached element. It treats the two cached tensors according to their roles in attention: the latent is quantized per token with a value-aware transform, and the positional key per channel with a score-aware transform. Hadamard is fixed, and KVarN and OSCAR derive their transforms from data statistics before any rounding takes place; RAQ-MLA instead fits general invertible transforms to the error that rounding leaves at the attention output. AffineQuant and FlatQuant also learn invertible transforms, but for weights and activations and by backpropagating through the network, whereas RAQ-MLA is fitted by alternating minimization without any gradient through the model.

B Overall Algorithm

This appendix states the two halves of RAQ-MLA in full: the offline fit of Section 4.4, which produces one transform per layer and per cached tensor, and the decode-time computation of Section 4.5, which consumes it.

Fitting. Algorithm 1 runs once per layer and per cached tensor and does not train the model. The unquantized model is run once on the calibration and validation splits to collect the cached tensor together with the statistics its role metric needs: the exposure weights $\gamma_{t,h}$ for the latent and the positional queries $\mathbf{q}^{\mathrm{R}}$ for the positional key. M_t is never formed explicitly; the objective and the inner solve use these factors directly. Each round freezes the quantized tensor, solves the proximal problem of Equation (12) by conjugate gradient, and line-searches toward the solution. Every trial step $B = A^{-1} + \alpha(\tilde{B} - A^{-1})$ is requantized and accepted only if the objective decreases, $F(B^{-1}) < F(A)$, and $\mathrm{cond}(B) \le \kappa_{\max}$; if no step is accepted, λ is increased tenfold, at most twice, before the fit stops. We run the fit for each λ in a small grid and return, among the initialization and all accepted iterates, the transform with the lowest attention-output error on the validation split.

Algorithm 1 Fitting one RAQ-MLA transform.

Require: calibration and validation splits; quantizer $\mathcal{Q}$; initial transform $A^{(0)}$; proximal weights Λ; round budget R; condition bound $\kappa_{\max}$
Ensure: transform A

Run the unquantized model on both splits; collect X and the factors of M_t ($\gamma_{t,h}$ and $\mathbf{W}^{\mathrm{UV}}$ for the latent, $\mathbf{q}^{\mathrm{R}}$ for the positional key)
$\mathcal{C} \leftarrow \{A^{(0)}\}$
for $\lambda \in \Lambda$ **do**
  $A \leftarrow A^{(0)}$
  for $r = 1, \dots, R$ **do**
    $Z \leftarrow \mathcal{Q}(XA)$
    $\tilde{B} \leftarrow \arg\min_B \ \mathcal{D}(ZB) + \lambda \|B - A^{-1}\|_F^2$ ▷ strongly convex; solved by CG
    Find the largest accepted $\alpha \in \{1, \frac{1}{2}, \dots\}$ ▷ requantized line search
    if no α is accepted **then break**
    $A \leftarrow \left(A^{-1} + \alpha(\tilde{B} - A^{-1})\right)^{-1}$; $\mathcal{C} \leftarrow \mathcal{C} \cup \{A\}$
  end for
end for
return the transform in $\mathcal{C}$ with the lowest attention-output error on the validation split

Inference. The fitted transforms enter decoding in two ways. The inverse of the latent transform is folded once, before serving, into the up-projections by Equation (14), so attention reads the cached

Algorithm 2 One RAQ-MLA decode step at a layer.

Require: hidden state $\mathbf{h}$; transforms A_{c}, A_{R}; folded up-projections $(\widetilde{\mathbf{W}}_h^{\mathrm{UK}})^\top = A_{\mathrm{c}}^{-1}(\mathbf{W}_h^{\mathrm{UK}})^\top$ and $(\widetilde{\mathbf{W}}_h^{\mathrm{UV}})^\top = A_{\mathrm{c}}^{-1}(\mathbf{W}_h^{\mathrm{UV}})^\top$; caches Z_{c}, Z_{R} (quantized pages followed by an open BF16 page)

Write path

1: $\widetilde{\mathbf{c}}^{\mathrm{KV}} \leftarrow \mathrm{RMSNorm}\big(\mathbf{h}(\mathbf{W}^{\mathrm{DKV}})^\top\big)\, A_{\mathrm{c}}$ ▷ $d_{\mathrm{c}} \times d_{\mathrm{c}}$ product

2: $\widetilde{\mathbf{k}}^{\mathrm{R}} \leftarrow \mathrm{RoPE}\big(\mathbf{h}(\mathbf{W}^{\mathrm{KR}})^\top\big)\, A_{\mathrm{R}}$ ▷ $d^{\mathrm{R}} \times d^{\mathrm{R}}$ product

3: Append $\widetilde{\mathbf{c}}^{\mathrm{KV}}$ and $\widetilde{\mathbf{k}}^{\mathrm{R}}$ to the open page; once the page is full, replace it by its quantized form

Read path, per head h

4: $\mathbf{q}_h^{\mathrm{C}},\ \mathbf{q}_h^{\mathrm{R}} \leftarrow$ query projections of $\mathbf{h}$, with RoPE applied to $\mathbf{q}_h^{\mathrm{R}}$

5: $\widetilde{\mathbf{q}}_h^{\mathrm{R}} \leftarrow \mathbf{q}_h^{\mathrm{R}} A_{\mathrm{R}}^{-\top}$ ▷ $d^{\mathrm{R}} \times d^{\mathrm{R}}$ product; cancels A_{R}

6: $L_h \leftarrow \Big(\mathbf{q}_h^{\mathrm{C}} \widetilde{\mathbf{W}}_h^{\mathrm{UK}} Z_{\mathrm{c}}^\top + \widetilde{\mathbf{q}}_h^{\mathrm{R}} Z_{\mathrm{R}}^\top\Big) / \sqrt{d_{\mathrm{h}} + d^{\mathrm{R}}}$

7: $\mathbf{o}_h \leftarrow \mathrm{softmax}(L_h)\, Z_{\mathrm{c}} (\widetilde{\mathbf{W}}_h^{\mathrm{UV}})^\top$

8: **return** the head outputs combined through $\mathbf{W}^{\mathrm{O}}$

latent directly in its learned basis. The forward transforms cannot be folded, because RMSNorm precedes A_{c} and RoPE precedes A_{R}; each is applied to the current token on the write path. Likewise, $A_{\mathrm{R}}^{-\top}$ must follow RoPE on the query side, which is admissible because $\mathbf{q}^{\mathrm{R}}$ is never cached. Algorithm 2 gives the resulting decode step, written with row vectors to match Section 4.

Relative to an unmodified MLA implementation, RAQ-MLA adds three small matrix products per decode step, all acting on the current token and hence independent of context length: one $d_{\mathrm{c}} \times d_{\mathrm{c}}$ product on the latent write path (line 1), one $d^{\mathrm{R}} \times d^{\mathrm{R}}$ product on the positional-key write path (line 2), and one $d^{\mathrm{R}} \times d^{\mathrm{R}}$ product per head on the query path (line 5). Attention itself runs on the quantized cache with the same structure as unmodified MLA.

Proof of Theorem 4.1. Let $G_r(B) = \mathcal{D}(Z^{(r)}B) + \lambda\|B - B^{(r)}\|_F^2$, whose minimizer $\tilde{B}$ is the update of Equation (12). Because $G_r(B^{(r)}) = F(A^{(r)})$ and $G_r(\tilde{B}) \le G_r(B^{(r)})$, the choice $B^{(r+1)} = \tilde{B}$ gives $\mathcal{D}(Z^{(r)}B^{(r+1)}) \le F(A^{(r)}) - \lambda\Delta_r$, and the hypothesis of the theorem replaces the left-hand side by $F(A^{(r+1)})$. The same bound holds for the damped step $B^{(r+1)} = B^{(r)} + \alpha(\tilde{B} - B^{(r)})$ accepted by Algorithm 1, with λ the value used in that round, since G_r is convex and does not exceed $F(A^{(r)})$ at either end of the segment. The sequence $F(A^{(r)})$ is therefore nonincreasing and bounded below by 0, so it converges, and summing the descent inequality over r gives $\sum_r \lambda\Delta_r \le F(A^{(0)})$, hence $\Delta_r \to 0$. Fitting therefore cannot degrade its initialization on the calibration objective under the stated condition; Algorithm 1 also enforces descent directly, since it accepts a step only if the requantized objective decreases. Because the rounding in $\mathcal{Q}$ makes F discontinuous in A, we claim no stationarity or optimality; how the fitted transforms transfer to unseen data is evaluated in Section 5.

C Analysis of the RAQ-MLA Fit

Figure 7 examines the latent fit on GLM-4.7-Flash from three angles. First, the fit converges within a few rounds (a, b). Averaged over the 47 INT2 layers, one update lowers the calibration objective to about two thirds of its initial value and the Val attention-output error from 25.3% to 21.7%. The Val error levels off at about 21.1% from the second round. Hadamard and OSCAR initializations follow nearly the same curves. Second, the choice of role metric matters: minimizing the latent's own quantization error, i.e., fitting with $M_t = I$, does not minimize the error that reaches the attention output. Under the same initialization, optimizer, and budget, the $M_t = I$ fit lowers the Val attention-output error from 25.3% to 23.3%, whereas the role metric lowers it to 21.1% and is better on all 47 layers. Third, the fitted transform reduces the error on both latent paths (c). Under the INT4 quantizer of Figure 3b, A_{c} lowers the error that reaches the attention output through the value from 8.3% to 5.9%, and through the content key from 1.7% to 1.6%, even though the value-aware objective does not model the content-key path.

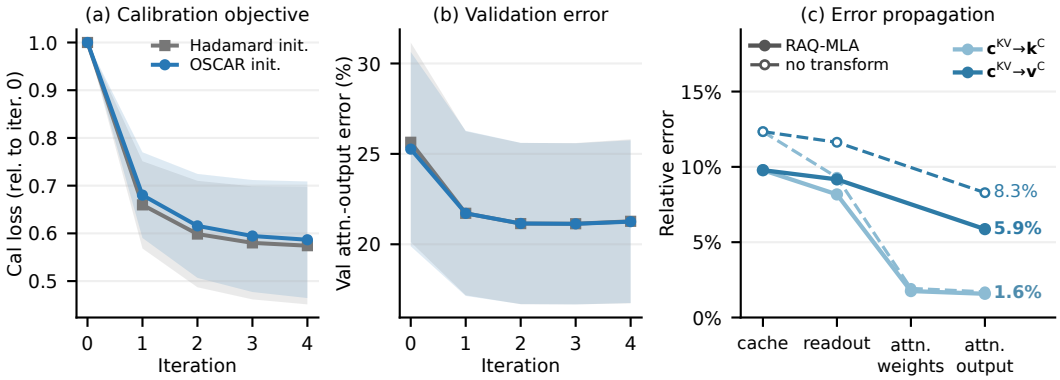


Figure 7: **Analysis of the RAQ-MLA latent fit on GLM-4.7-Flash.** (a, b) Calibration objective relative to iteration 0 and Val attention-output error per round, mean±std over the 47 INT2 layers along each layer's Val-selected damping trajectory, from Hadamard and OSCAR initializations. (c) Relative error of the latent at each stage under the INT4 quantizer of Figure 3b, with the fitted A_c (solid) and without a transform (dashed).

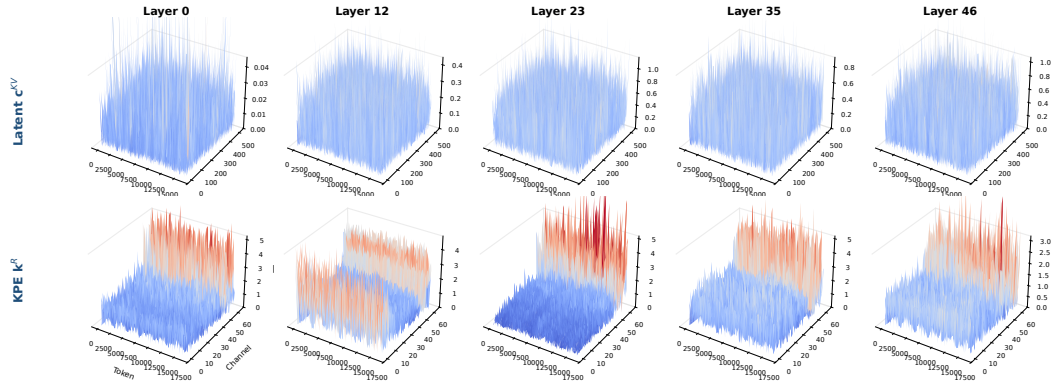


Figure 8: **The latent and positional key retain different channel structures throughout depth.** Token–channel magnitude surfaces for the normalized compressed latent $\mathbf{c}^{\mathrm{KV}}$ (top) and post-RoPE positional key $\mathbf{k}^{\mathrm{R}}$ (bottom) at five approximately uniformly spaced layers of GLM-4.7-Flash. Each panel uses its own 99th-percentile vertical scale, so the comparison concerns structure rather than absolute magnitude. The latent remains diffuse across channels, whereas the positional key exhibits persistent fixed-channel bands from early to late layers.

D Additional Diagnostic Figures

Logit attribution. The logit attribution in Section 3 is the share of the squared logit norm carried by each group of channels. For head h, query position q, and cached token t, the content contribution of a channel set $\mathcal{S}$ is $(\mathbf{q}^{\mathrm{C}}_{q,h}\mathbf{W}^{\mathrm{UK}}_{h})_{\mathcal{S}}(\mathbf{c}^{\mathrm{KV}}_{t})^{\top}_{\mathcal{S}}$, and the positional contribution is $(\mathbf{q}^{\mathrm{R}}_{q,h})_{\mathcal{S}}(\mathbf{k}^{\mathrm{R}}_{t})^{\top}_{\mathcal{S}}$. Outlier channels are the 10% of largest magnitude. Stacking each group's contributions over heads, queries, and tokens $t \le q$ gives L_g, with $L = \sum_g L_g$. The share is $a_g = \langle L_g, L\rangle_F / \|L\|_F^2$, pooled by summing the numerator and the denominator over all 47 layers. The reported 92.5% is the positional key's share.

Outlier structure across layer depth (extension of Figure 2). Figure 2 contrasts the token–channel magnitude patterns of the compressed latent and positional key at one representative layer. The depth extension in Figure 8 shows the same two cached tensors at five approximately uniformly spaced layers.

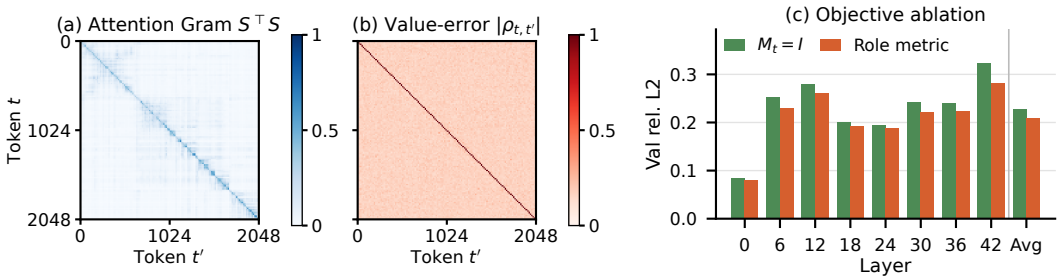


Figure 9: **Value-readout errors are nearly uncorrelated across tokens, although attention reads nearby tokens together.** (a–b) Layer 23 of GLM-4.7-Flash, first 2048 tokens of a calibration prompt, deployed INT2 quantizer with the fitted RAQ-MLA transform, visualized at 16-token resolution. (a) Attention Gram $S^\top S$, normalized by its diagonal and averaged over heads. (b) Magnitude of the correlation $\rho_{t,t'}$ between the value-readout errors at tokens t and t', pooled over heads. (c) Val attention-output error of the latent transform fitted with $M_t = I$ or with the role metric under the same fitting pipeline.

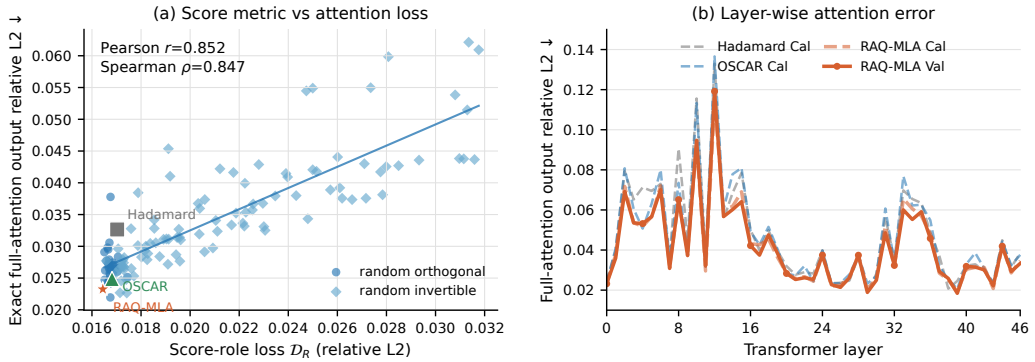


Figure 10: **Evaluation of the positional-key role metric.** (a) Score-role loss $\mathcal{D}_{\mathrm{R}}$ against the exact attention-output error at layer 23 across 128 random transforms; Hadamard, OSCAR, and RAQ-MLA are excluded from the correlations. (b) Layer-wise attention-output error; baselines are shown on calibration prompts only.

Token independence of the value-path error. The value-aware objective of Section 4.2 keeps only the diagonal of $S_h^\top S_h$, which assumes that quantization errors at different tokens are nearly uncorrelated. Figure 9 tests this assumption at layer 23 over the first 2048 tokens of a calibration prompt, under the deployed INT2 quantizer and the fitted RAQ-MLA transform. The attention Gram is far from diagonal: nearby tokens are read together by the same queries. The value-readout errors, in contrast, are correlated only with themselves; the mean off-diagonal $|\rho_{t,t'}|$ is 0.044. Summed over all token pairs, the residual off-diagonal terms still account for 14.2% of the value-path distortion at this layer (13.1% for Hadamard and 16.3% without a transform). Because this share is similar across transforms, dropping them mainly rescales $\mathcal{D}_{\mathrm{c}}$ rather than reordering candidate transforms (Figure 5a).

Positional-key role metric (extension of Figure 5). Figure 5 validates the role metric of the latent; Figure 10 repeats both tests for the score-aware metric M_t^{R} of Section 4.3. Only the positional key is quantized, to INT4 per channel over each cache page, while the latent stays in BF16. At layer 23, we score 32 random orthogonal and 96 random invertible 64×64 transforms by both $\mathcal{D}_{\mathrm{R}}$ and the exact attention-output error. The two remain strongly correlated (Pearson $r = 0.852$, Spearman $\rho = 0.847$), though less tightly than for the latent ($r = 0.992$). Across layers, fitting A_{R} lowers the mean attention-output error on validation prompts from 0.050 for Hadamard and 0.047 for OSCAR to 0.042, and is below OSCAR at 43 of the 47 layers and below Hadamard at 40. Its calibration and validation curves nearly coincide, differing by 3.7% on average.

E Additional Implementation and Evaluation Details

E.1 Quantizer and Cache Accounting

All cached tensors use asymmetric uniform quantization with FP16 metadata. A group of values x with minimum m and maximum M is quantized to b bits as

$$s = \frac{M - m}{2^b - 1}, \qquad q = \mathrm{clamp}\left(\mathrm{round}\left(\frac{x - m}{s}\right), 0, 2^b - 1\right), \qquad \hat{x} = s\,q + m, \tag{15}$$

where s and m are stored in FP16 and rounding is to nearest, with ties to even. The two cached tensors differ in bit width, grouping, and clipping (Table 5). The offline fit uses the same quantizers as the runtime, matched bit for bit.

Table 5: Quantizer configuration of RAQ-MLA. P is the cache-page size in tokens.

	Latent $\mathbf{c}^{\mathrm{KV}}$	Positional key $\mathbf{k}^{\mathrm{R}}$
Quantized tensor	$\mathbf{c}_t^{\mathrm{KV}} A_{\mathrm{c}}$	$\mathbf{k}_t^{\mathrm{R}} A_{\mathrm{R}}$ (after RoPE)
Bits (levels)	2 (4)	4 (16)
Grouping	per token, groups of 128 channels	per channel, within a cache page
Clipping	per token, 0.94 quantile of $\lvert x\rvert$	none
Metadata	FP16 scale and minimum per group	FP16 scale and minimum per channel and page
Effective bits	$2 + 32/128 = 2.25$	$4 + 32/P$ (4.25 at $P = 128$, 4.50 at $P = 64$)

Latent. The latent is quantized per token in its learned basis. Before grouping, each token's $d_{\mathrm{c}} = 512$ channels are clipped: with τ_t the 94th percentile of the token's channel magnitudes, every channel is clamped to $[-\tau_t, \tau_t]$, which affects the 30 largest of the 512 channels. At two bits a group has only four levels, so a few large channels would otherwise stretch its range and coarsen the step for all others; clipping trades their error for a finer step on the rest. The clipped row is then split into four groups of 128 contiguous channels, each quantized with its own scale and minimum.

Positional key. The positional key is quantized after RoPE and A_{R}, per channel within each cache page: the P values of one channel in one page share a scale and a minimum. This grouping follows the fixed-channel outliers of $\mathbf{k}^{\mathrm{R}}$ (Section 3): a large channel receives its own range rather than stretching the range of the other channels in the same token. No clipping is applied.

Cache pages and the BF16 tail. Following KVarN (Muller et al., 2026), a page is quantized when it is sealed. New tokens are written to the open page in BF16, and the whole page is quantized once its last slot is written; per-channel quantization of $\mathbf{k}^{\mathrm{R}}$ requires this anyway, because its scales are computed over the tokens of a page. The page size follows the attention backend: $P = 128$ tokens for GLM-4.7-Flash, Moonlight, and DS-V2-Lite, and $P = 64$ for GLM-5.2, whose sparse-attention backend uses 64-token pages. The most recent tokens therefore stay in BF16, but on a sawtooth schedule rather than as a sliding window: the BF16 region grows to a full page and then collapses. All quantized methods, including the baselines, use the same page schedule.

Bits per element. We report effective bits per cached element (BPE), counting code bits and FP16 metadata and averaging over both cached tensors,

$$\mathrm{BPE} = \frac{d_{\mathrm{c}}\, b_{\mathrm{c}} + d^{\mathrm{R}}\, b_{\mathrm{R}}}{d_{\mathrm{c}} + d^{\mathrm{R}}}, \tag{16}$$

where b_{c} and b_{R} are the effective bits of the latent and the positional key. RAQ-MLA costs $(512 \cdot 2.25 + 64 \cdot 4.25)/576 = 2.47$ BPE with 128-token pages and 2.50 with 64-token pages, a $6.47\times$ and $6.40\times$ capacity increase over BF16. Methods that leave the positional key in BF16 cost at least $(512 \cdot 2.25 + 64 \cdot 16)/576 = 3.78$ BPE ($4.23\times$) even with a two-bit latent. The RULER results in Table 2 raise the first two and last two layers to an INT4 latent and an INT8 positional key, which costs 2.66 BPE over the 47 layers of GLM-4.7-Flash. BPE excludes the BF16 open page, which holds at most one page per sequence and layer; at 64K tokens it adds at most 0.2% of the tokens, or 0.03 bits per element.

E.2 Evaluation Protocol

All models are served with vLLM behind an OpenAI-compatible API, with the quantizers of Appendix E.1 applied when a cache page is written. Table 6 lists the serving and sampling configuration of the reasoning and coding tasks; within a model, all methods share it.

Table 6: Serving and sampling configuration for the reasoning and coding tasks. ARC-Challenge, PIQA, and RULER NIAH use greedy decoding instead.

	GLM-4.7-Flash	GLM-5.2-FP8	Moonlight	DS-V2-Lite
`max_model_len`	202,752	202,752	8,192	32,768
`max_tokens`	131,072	65,536	7,168	7,168
Cache page (tokens)	128	64	128	128
Temperature	1.0	1.0	1.0	1.0
Top-p	0.95	0.95	0.95	0.95

AIME25, GPQA-Diamond, MATH-500, GSM8K, and HumanEval are evaluated with EvalScope 1.9.1, using eight concurrent requests, the sampling settings of Table 6, and a generation budget of `max_tokens`. We report accuracy, and pass@1 for HumanEval. Because sampling makes individual runs noisy, some of the reported results are averaged over three seeds. ARC-Challenge and PIQA, used for Moonlight and DS-V2-Lite, are scored as multiple-choice questions with greedy decoding at temperature 0. RULER NIAH is evaluated with OpenCompass over all eight needle subtasks at 4K, 8K, 16K, 32K, 64K, and 128K, with 100 samples per subtask and length, greedy decoding, an output limit of 128 tokens, and thinking disabled.

F Extended Results

Long-context understanding. Table 7 reports the LongBench-v1 macro average over all 21 tasks on GLM-4.7-Flash, with thinking enabled. RAQ-MLA stores a 30% smaller cache than OSCAR and scores on par with it.

Table 7: **LongBench-v1 macro average (%) over 21 tasks on GLM-4.7-Flash**, thinking on.

Method	BPE	Single-QA	Multi-QA	Summ.	Few-shot	Synth.	Code	Avg
OSCAR	3.78	**49.16**	**57.24**	**22.88**	60.46	**77.20**	**54.75**	**52.38**
RAQ-MLA	**2.66**	49.07	56.83	22.80	**61.84**	76.30	52.44	52.19

Decode latency across output lengths. Table 8 extends the single operating point of Table 4 to output lengths from 1K to 8K tokens, with a 512-token prompt and batch size 1. The packed cache lowers TPOT by 6.8–9.0% at every length as the cache grows to 8.7K tokens. The gain is smaller than the 1.38× of the decode kernel in Table 4 because attention is only part of each decode step at this scale.

Table 8: **Decode TPOT across output lengths on GLM-4.7-Flash**, with a 512-token prompt and batch size 1. Speedup is the FlashMLA TPOT divided by the RAQ-MLA TPOT.

Output tokens	FlashMLA (ms)	RAQ-MLA (ms)	Speedup
1024	6.61	**6.04**	1.09×
2048	6.69	**6.11**	1.09×
4096	6.87	**6.25**	1.10×
8192	7.02	**6.54**	1.07×

Cost of the latent transform. Applying A_c is the only per-token matrix product on the latent write path. We measure it on one H20 GPU with a single MLA layer that has the cache shape of

GLM-4.7-Flash ($d_c = 512$, $d^R = 64$), 64-token pages, and 2K cached tokens, under CUDA Graphs (Table 9). The layer has 64 heads; the Hopper kernel processes heads in tiles of 64, so this costs the same as the 10 heads per GPU of GLM-4.7-Flash. Fused into the cache-write kernel, the transform costs 1.7–2.4 μs per layer for decode batches from 1 to 64, nearly independent of the batch size. Over 47 layers this is about 0.1 ms per decode step, under 2% of the 6 ms TPOT at batch size 1. Its share of the per-layer write-plus-attention time falls from 9.9% at batch size 1 to 1.0% at batch size 64, since attention grows with the batch while the transform does not. Each layer stores one 512 KiB BF16 matrix, 23.5 MiB per GPU over 47 layers, and the fused and unfused paths write byte-identical cache entries.

Table 9: **Per-layer cost of the latent transform** on a single GLM-shaped MLA layer with 2K cached tokens. Standalone applies A_c as a separate GEMM before the cache write; fused applies it inside the write kernel. Share is the fused cost relative to the per-layer cache write plus attention.

Decode batch	Standalone (μs)	Fused (μs)	Share
1	2.66	**2.36**	9.9%
2	2.68	**1.72**	6.6%
4	2.64	**1.77**	5.5%
8	2.70	**1.81**	4.0%
64	2.83	**2.24**	1.0%