NIBBLEMLA: DUAL-AXIS FP4 ATTENTION OVER A SINGLE LATENT CACHE

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-head latent attention (MLA) reduces the key–value cache to a single latent per token that serves as both key and value. What remains is numeric: how many bits store that latent, and in what precision the two attention GEMMs read it. FP4 latent caches already exist, but they dequantize to BF16 before attention. We show that this is forced by a structural mismatch rather than by accuracy: block-scaled FP4 tensor cores accept scale factors only along the contraction dimension, whereas the shared latent is contracted along channels by QK and along tokens by PV. We prove that a scale written once per token can be consumed by both GEMMs only if it is a per-token scalar times a static channel vector—precisely the layout that suits raw latents worst, because their channel magnitudes are highly uneven. NibbleMLA resolves this tension by changing the basis instead of the scale: a fixed 512-dimensional Hadamard co-rotation, which costs no storage, equalizes the channels so that one E4M3 scalar per token suffices. The resulting 257-byte-per-token cache ($0.44\times$ FP8) feeds both attention GEMMs on FP4 tensor cores without dequantization. On DeepSeek-V4-Flash, FP4 attention matches BF16 attention computed on the same cache (output cosine ≥ 0.9999 in all 43 layers); end-task accuracy is statistically indistinguishable from the FP8 cache on GSM8K, AIME, and LongBench-v2 and within 0.7 points on 13-task RULER at 128K; and the decode-attention kernel is 1.10–$1.27\times$ faster than the FP8 kernel and, on average, faster than the official FP4 kernel, which dequantizes.

1 INTRODUCTION

Decoding over long contexts is dominated by reading the key–value (KV) cache. Multi-head latent attention (MLA) (DeepSeek-AI et al., 2024; 2025) addresses this in the architecture: each token stores a single low-dimensional latent that every query head uses as both its key and its value. In DeepSeek-V4-Flash (DeepSeek-AI et al., 2026b), the model we study, this is one 512-dimensional vector per token and layer. Once the architecture has removed the redundancy across heads, the remaining levers are numeric: how many bits *store* each latent, and in what precision the two matrix multiplications that *read* it are computed.

Production deployments store the latent in FP8 ($584\,\mathrm{B}/\text{token per layer}$). The recent DeepSeek-V4.1 cache (DeepSeek-AI et al., 2026a) halves this with FP4 codes and one E4M3 scale per 16 channels ($288\,\mathrm{B}/\text{token}$), yet it dequantizes the cache to BF16 before attention: FP4 is used to reduce storage, not to accelerate the matrix multiplications. The latent is stored in four bits but still read in sixteen. This is puzzling at first sight: Blackwell tensor cores execute block-scaled FP4 GEMMs natively, and the FP4 codes are already in memory.

We show that the obstacle is structural rather than numerical. A block-scaled FP4 tensor core applies one scale to every 16 consecutive elements *along the contraction dimension*. The shared latent, however, is contracted along two different axes: the logits $\mathbf{q}^\top C_t$ reduce over channels, whereas the output $\sum_t p_t C_t$ reduces over tokens. A per-16-channel scale lies along the first contraction but cuts across the second, and honoring it in the second requires dequantizing and re-quantizing the value tile at several times the cost of the GEMM it feeds. Proposition 1 (§3) makes this exact: among scale layouts written once per token, the only ones that *both* GEMMs can consume directly are a per-token scalar times a static channel vector.

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.

This characterization exposes a tension. The layout that the hardware admits on both axes—one scale per token—is the one that suits the data least. MLA latents have highly uneven channel magnitudes, with most outliers in the 64 rotary channels, so a per-token scale is set by a few loud channels and wastes the four bits on the rest: a quarter of the non-rotary entries round to zero, and a single-scale FP4 cache falls 4.6 points below FP8 on GSM8K even when attention is computed in BF16 (§3).

NibbleMLA resolves the tension by changing the basis instead of the scale. A fixed 512-dimensional Hadamard co-rotation of the latent and the query leaves every logit unchanged, adds no bytes, and spreads energy evenly across channels, so that a single E4M3 scalar per token becomes sufficient. Each cache entry is then $256\,\mathrm{B}$ of E2M1 codes plus one scale byte—$257\,\mathrm{B/token}$, $0.44\times$ the FP8 cache and smaller than the per-16-channel FP4 cache. Because this layout satisfies Proposition 1, both GEMMs read it directly: the scalar acts as the key's block scale in QK and is folded into the softmax probabilities in PV. Attention thus runs in FP4 along *both* axes of the one latent cache.

Our contributions are as follows.

1. **The scale-axis constraint** (§3). We characterize which scale layouts block-scaled FP4 tensor cores can consume for a tensor that is contracted along two axes, and show that for a shared MLA latent the only per-token layouts readable by both attention GEMMs are single scalars up to a static channel weighting. This explains why existing FP4 latent caches dequantize, and it turns the design problem into an accuracy problem for one specific layout.
2. **NibbleMLA** (§4). A $257\,\mathrm{B/token}$ FP4 latent cache in which a zero-byte Hadamard co-rotation makes a single per-token scale accurate, together with a decode kernel that computes QK and PV on FP4 tensor cores directly from the paged cache.
3. **Evidence** (§5). On DeepSeek-V4-Flash, computing attention in FP4 is indistinguishable from BF16 attention on the same cache; end-task accuracy matches the FP8 cache within statistical noise at $0.44\times$ its memory; and the kernel is 1.10–$1.27\times$ faster than the FP8 kernel and faster on average than the official FP4 kernel that dequantizes. We claim accuracy parity with FP8, not accuracy superiority.

2 PRELIMINARIES

One latent, two contractions. In MLA, token t writes one latent $C_t \in \mathbb{R}^d$ per layer. After the key and value up-projections are absorbed into the query and output projections, attention for a query $\mathbf{q}$ computes

$$s_t = \tau\, \mathbf{q}^\top C_t + a_t, \qquad \mathbf{p} = \mathrm{softmax}(\mathbf{s}), \qquad \mathbf{o} = \textstyle\sum_t p_t\, C_t, \tag{1}$$

with softmax scale τ and additive bias a_t. The same tensor $C = [C_t]$ is therefore contracted over *channels* to form the logits (QK) and over *tokens* to form the output (PV). In DeepSeek-V4-Flash (DeepSeek-AI et al., 2026b), $d = 512$ (448 content and 64 rotary coordinates, stored after RoPE), a single latent head (`num_key_value_heads` $= 1$) serves all query heads, and attention runs over paged entries—a top-k subset of compressed entries chosen by a lightning indexer, plus a sliding window—merged in one softmax with a sink logit. In earlier MLA models the 512-dimensional latent is likewise shared by key and value, alongside a separate 64-dimensional rotary key (DeepSeek-AI et al., 2024; 2025); our analysis concerns the shared part in either case.

Block-scaled FP4 GEMMs. E2M1 ("FP4") has a sign bit and eight magnitudes, 0, 0.5, 1, 1.5, 2, 3, 4, and 6, so every use requires a scale. Microscaling formats (Rouhani et al., 2023) attach one scale to each contiguous block of elements; NVFP4 on NVIDIA Blackwell uses blocks of 16 with unsigned E4M3 scales, and the tensor core applies these scales inside the multiply. For operands $A \in \mathrm{E2M1}^{M\times K}$ and $B \in \mathrm{E2M1}^{N\times K}$ with scale tensors S^A and S^B,

$$D_{mn} \;=\; \textstyle\sum_k \; S^A_{m,\lfloor k/16\rfloor} A_{mk}\; S^B_{n,\lfloor k/16\rfloor} B_{nk}, \tag{2}$$

accumulated in FP32. Scales thus live on each operand's (row, K-block) grid: they may vary freely along the free dimension but must be constant over every 16 consecutive elements of the contraction dimension. Everything in Section 3 follows from this rule. Applied per 16 channels of the latent, NVFP4 yields the DeepSeek-V4.1 FP4 cache: $256\,\mathrm{B}$ of codes plus 32 scale bytes, i.e. $288\,\mathrm{B/token}$.

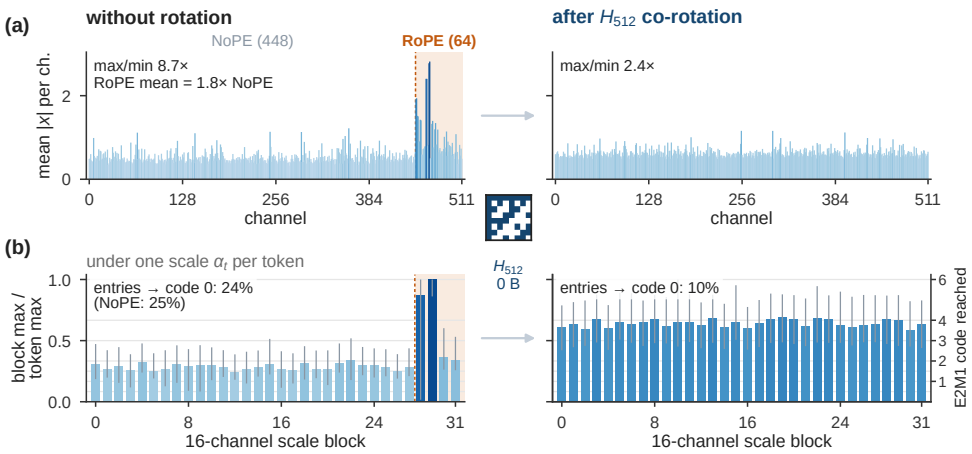

Figure 1: **The accuracy problem of the legal layout, and its fix.** Stored latent of layer 32 of DeepSeek-V4-Flash (2000 tokens), before and after the zero-byte H_{512} co-rotation (§4.1). *(a)* Mean $|x|$ per channel. *(b)* Under one scale per token, $\alpha_t = \max_c |x_{t,c}|/6$, the fraction of the E2M1 range reached by each 16-channel block (median over tokens, whiskers p10–p90).

3 The Scale-Axis Constraint

A quantized latent cache stores E2M1 codes $q_{t,c}$ together with a scale layout S, so that $\hat{C}_{t,c} = S_{t,c}\, q_{t,c}$. We ask which layouts allow both GEMMs of Eq. (1) to run as Eq. (2) with the stored codes as a tensor-core operand, without dequantization or re-quantization. We call a layout *directly consumable* by a GEMM if S can be realized through the two operands' block scales, the values of the other (transient) operand, and a static rescaling of the GEMM output. In QK the cache is operand B with $n = t$ and $k = c$; in PV it is operand A with $m = c$ and $k = t$, while the query and the probabilities are the transient operands.

Proposition 1 (Scale-axis constraint)**.** *Suppose each token's scales $S_{t,\cdot}$ are fixed when the token is written, as a function of C_t alone. Then (i) every per-token, per-16-channel layout $S_{t,c} = u(t, \lfloor c/16 \rfloor)$ is directly consumable by QK; (ii) a layout is directly consumable by PV if and only if $S_{t,c} = \alpha_t\, b_c$, where α_t depends on the token and b is a static channel vector; hence (iii) the layouts directly consumable by both GEMMs are exactly $S_{t,c} = \alpha_t\, b_c$.*

Proof sketch. In QK, $u(t, \lfloor c/16 \rfloor)$ is a valid block scale of B. In PV, a block scale of the cache operand must be constant over 16 consecutive tokens, which a scale fixed by each token alone cannot be unless it is static. A token-dependent factor can instead move into the transient operand, whose entries $x_{h,t}$ carry no channel index, and a channel-dependent factor into a static rescaling of the output; nothing else can be absorbed. The full argument is in Appendix A. □

The proposition has three consequences, which organize the rest of the paper.

Why FP4 latent caches dequantize. The 288 B/token per-16-channel cache satisfies (i) but not (ii): QK could read it in FP4, PV cannot. Running PV in FP4 anyway requires either dequantizing the value tile and re-quantizing it along tokens—about 8× the arithmetic, which two independent implementations measured at 0.083–0.22× the throughput of a directly consumable layout—or re-quantizing P once for each of the 32 channel blocks. A 16-token × 16-channel tile scale would be legal for both GEMMs, but it couples the scales of 16 consecutive tokens and breaks write-once caching. Dequantizing to BF16 is the rational response, and it is what DeepSeek-V4.1 does.

What remains. With b absorbed into the query for QK and into the output for PV, the feasible layouts reduce to one scalar per token. In QK, α_t is broadcast as the cache operand's block scale across all 32 channel blocks; in PV, the cache operand carries unit scale and α_t is multiplied into the probability p_t before $\mathbf{p}$ is quantized, where a per-token factor lies along the contraction and is legal (§4.3).

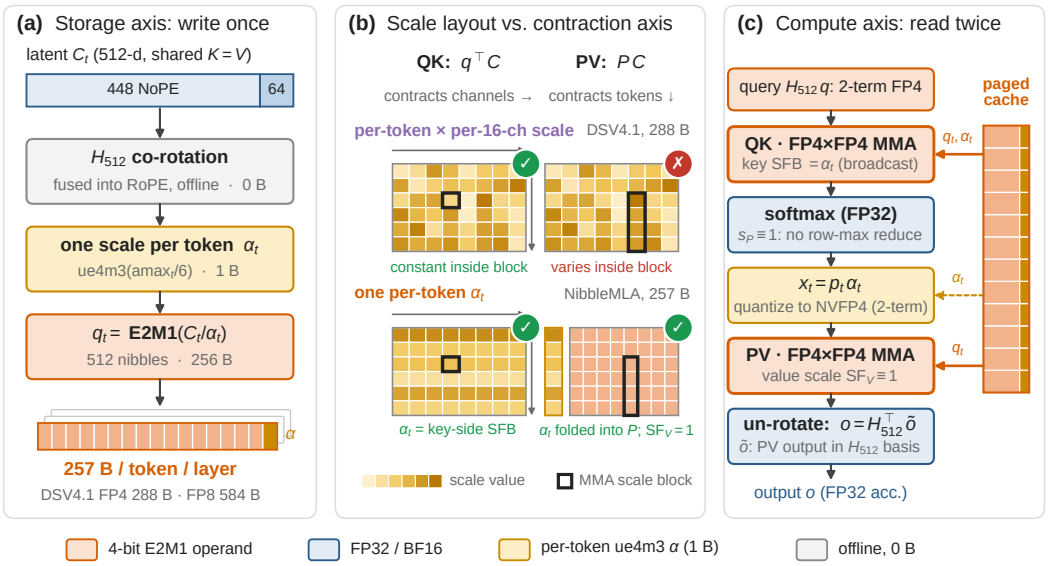

Figure 2: **NibbleMLA writes the shared latent once and reads it along two axes in FP4.** *(a)* Storage axis (write once): co-rotate the 512-d shared $K{=}V$ latent by H_{512} (0 bytes, fused into RoPE), choose one per-token scale α_t, and round to E2M1—257 B/token (256 B of codes + 1 B for α_t), against 288 B for the DeepSeek-V4.1 FP4 cache and 584 B for FP8. *(b)* Scale layout versus contraction axis: a per-16-channel scale is constant inside QK's scale blocks but varies inside PV's (Proposition 1); a single per-token α_t is the key-side block scale in QK and folds into P in PV, leaving the value scale at 1. *(c)* Compute axis (read twice): both FP4×FP4 MMAs read the same packed codes from the paged cache; the PV output, formed in the H_{512} basis, is mapped back by $H_{512}^\top$; softmax and accumulation are FP32.

The accuracy problem. A single scale must cover all 512 channels, and MLA latents are far from isotropic. In a representative layer (Figure 1a) the 64 rotary channels carry the outliers and mean magnitudes vary by 8.7× across channels. A per-token scale set by those channels leaves the median content block at 0.29 of the E2M1 range, and 25% of content entries round to zero (Figure 1b). The damage is visible on downstream tasks: an FP4 cache with this layout and no further treatment (`naive_fp4`, §5) is 4.6 points below FP8 on GSM8K ($p = 1.5{\times}10^{-6}$), although it computes attention in BF16. The layout that the hardware admits on both axes is the one that needs help most.

4 NibbleMLA

NibbleMLA keeps the layout that Proposition 1 permits and changes what it is applied to. Section 4.1 makes a single per-token scale accurate, Section 4.2 specifies the cache entry, and Section 4.3 computes both GEMMs from it. Figure 2 shows the write path, the legal scale layout, and the FP4 read path.

4.1 A zero-byte co-rotation makes one scale enough

Let $H_{512} \in \mathbb{R}^{512\times512}$ be the normalized Walsh–Hadamard matrix, $H_{512}^\top H_{512} = I$. NibbleMLA stores the rotated latent $\tilde{C}_t = H_{512}C_t$ and rotates each query, $\tilde{\mathbf{q}} = H_{512}\mathbf{q}$. Orthogonality preserves every logit, $\tilde{\mathbf{q}}^\top \tilde{C}_t = \mathbf{q}^\top C_t$, and the output formed from rotated values, $\sum_t p_t \tilde{C}_t = H_{512}\mathbf{o}$, is mapped back by $H_{512}^\top$. The transform is fixed, costs $O(d \log d)$ via the fast Walsh–Hadamard transform, is applied in the same pass as RoPE when an entry is written, and adds no bytes to the cache.

Rotation matters here for a specific reason. Every coordinate of $H_{512}C_t$ is a $\pm 1/\sqrt{512}$ combination of all 512 original coordinates, so energy that is concentrated in a few channels is spread over all of them—exactly what a single scale shared by the whole vector requires. In the layer of Figure 1, the channel spread falls from $8.7\times$ to $2.4\times$, the median 16-channel block reaches 0.64 instead of 0.29 of the E2M1 range, the fraction of entries that round to code 0 falls from 24% to 10%, and the median per-token reconstruction error $1 - \cos(C_t, \hat{C}_t)$ falls from 1.2×10^{-2} to 6.4×10^{-3} (Appendix C). This is the opposite regime from recent FP4 studies, which find that global rotations help little or even hurt when every 16 or 32 elements have their own scale (Egiazarian et al., 2026; Shao et al., 2025): there each block scale already adapts to local outliers, whereas here one scale spans all 512 coordinates. The rotation does not change which layouts are legal; it changes how accurate the legal one is.

4.2 The 257-byte cache entry

Each entry stores the 512 rotated coordinates as E2M1 codes and one unsigned E4M3 scale:

$$b_{\text{NibbleMLA}} = \underbrace{512 \times \tfrac{4}{8}}_{\text{E2M1 codes}} + \underbrace{1}_{\alpha_t} = 257 \text{ B per token and layer,} \tag{3}$$

which is $0.44\times$ the FP8 cache (584 B), $0.89\times$ the per-16-channel FP4 cache (288 B), and $0.25\times$ BF16 (1024 B). No per-channel metadata is stored ($b \equiv 1$ in Proposition 1). Because the rotary coordinates are part of the latent in DeepSeek-V4-Flash, they are rotated and quantized with the rest, and no separate 16-bit rotary cache is kept. The scale α_t is chosen when the entry is written, from three candidate clipping levels (1.0, 0.85, 0.7 of the per-token maximum), using a reconstruction criterion for compressed entries and an attention-aware criterion for sliding-window entries; with only these three candidates and run entirely at write time (never in the decode path), the search is lightweight. The only offline step is a label-free calibration of one constant per layer.

4.3 Computing QK and PV in FP4

Both GEMMs read the same packed codes from the paged cache; neither materializes a dequantized copy (pseudocode in Appendix D).

QK. The cache tile enters Eq. (2) as operand B with block scale α_t broadcast over its 32 channel blocks. The query is quantized to E2M1 with per-16-channel E4M3 scales as a two-term residual, $\mathbf{q} \approx \mathbf{q}^{(1)} + \mathbf{q}^{(2)}$ with $\mathbf{q}^{(2)} = \mathrm{fp4}(\mathbf{q} - \mathbf{q}^{(1)})$; the two terms are stacked along the MMA's M dimension and share one pass over the cache operand. Logits accumulate in FP32, and one FP32 softmax covers all branches and the sink.

PV. Here the contraction runs over tokens, where α_t cannot be a block scale of the cache. As Proposition 1 prescribes, the cache operand carries unit scale and α_t moves to the transient operand: $x_t = p_t\,\alpha_t$ is quantized online to a two-term NVFP4 residual whose per-16-token scales lie along the contraction. The output accumulates in FP32 and is rotated back by $H_{512}^{\top}$.

A unit row scale. NVFP4 quantization of P ordinarily also keeps a per-row FP32 scale derived from the running softmax maximum, which costs a cross-warp reduction at every step. Since probabilities are taken relative to the running maximum, $p_t \le 1$ and hence $x_t \le \alpha_t$, which is itself an E4M3 value; the block scale $\max_{16}(x)/6$ therefore never saturates, and NibbleMLA fixes the row scale to 1. Removing the reduction is the largest single kernel optimization (-23.1% latency at batch 128, top-k 2048).

Precision accounting. Every tensor-core operand is E2M1 with E4M3 block scales. The transient query and probabilities use a two-term expansion (one extra FP4 pass each), whereas the cache—the operand whose bytes dominate memory traffic—is read once, as single-term FP4. Softmax and accumulation are FP32. All accuracy numbers in Section 5 are produced by this kernel; no simulated quantization is used.

5 EXPERIMENTS

The argument so far makes three testable predictions: computing attention in FP4 on the legal layout should cost no accuracy relative to BF16 on the same cache (§5.2); with the co-rotation, the 257 B/token cache should match FP8 on downstream tasks, whereas the same layout without it should not (§5.3, §5.4); and reading fewer bytes without dequantizing should make the attention kernel faster (§5.5).

5.1 SETUP

Model and methods. All experiments use DeepSeek-V4-Flash with its 512-dimensional shared latent (§2). We compare six cache configurations in one serving harness. `fp8` is the production FP8 cache (584 B/token). Four FP4 caches share the E2M1 codes and differ in scale layout: `naive_fp4` uses one scale per token, i.e. NibbleMLA's layout without the co-rotation; `group16_fp4` uses one E4M3 scale per 16 channels, the layout of the 288 B DeepSeek-V4.1 cache (in our implementation); `scalesearch_fp4` adds ScaleSearch's block-scale search (Gupta et al., 2026), which chooses each 16-channel E4M3 scale by a search that minimizes quantization error rather than by the block maximum; and `sage3` adds SageAttention3-style channel centering (Zhang et al., 2026a). All five baselines dequantize the cache and compute attention in BF16, giving them the more precise attention arithmetic; NibbleMLA alone computes both GEMMs in FP4. Per-layer byte counts are in Appendix E.

Benchmarks and statistics. We use GSM8K (Cobbe et al., 2021) (500 problems), AIME (60 problems), the full 13-task RULER (Hsieh et al., 2024) at 128K (100 samples per task), and all 503 questions of LongBench-v2 (Bai et al., 2025) (contexts middle-truncated to 128K). Prompts use the model's chat template; decoding is greedy with one request in flight, so every run is deterministic. We compare each method with FP8 on the same items using an exact McNemar test and, for NibbleMLA, report the paired accuracy difference with a 95% confidence interval (Appendix B.2 gives the same statistics for the baselines). Parity means that this interval contains zero and is narrow; we never claim to beat FP8.

5.2 FP4 ATTENTION MATCHES BF16 ATTENTION ON THE SAME CACHE

The cleanest test of the compute path holds the cache fixed. On the same 257 B/token cache we compute attention either with NibbleMLA's FP4 kernel or by dequantizing to BF16. On a paired set of 200 LongBench-v2 questions (the 200 shortest by context length, 512 generated tokens), FP4 attention scores .510 and BF16 attention .505 (net $+1$ question, McNemar $p = 1.000$). The attention outputs of the FP4 GEMMs agree with an FP32 GEMM on the same cache to a cosine similarity of at least 0.99990 in all 43 layers (worst: layer 27), and the kernel's per-row output cosine is at least 0.9998 across 20 benchmark shapes. Computing attention in FP4 on the legal layout therefore costs no measurable accuracy; whatever separates NibbleMLA from FP8 lies in the cache contents, not in the arithmetic.

5.3 ACCURACY AT 0.44× THE CACHE MEMORY

Table 1 gives the main result. On GSM8K, NibbleMLA and FP8 disagree on only 7 of 500 problems (4 wins, 3 losses), and the paired interval bounds any difference to $[-0.8, +1.2]$ points. On LongBench-v2 the difference is $+1.0$ points with interval $[-2.6, +4.6]$ (45 wins, 40 losses). AIME, with 60 problems, is too small to resolve differences of a few points; we report it for completeness (interval $[-11.5, +8.1]$). None of these differences is significant (paired McNemar $p = 1.00$, 1.00, and 0.66 for GSM8K, AIME, and LongBench-v2); on GSM8K and LongBench-v2, where the intervals are narrow, we read them as parity at 0.44× the cache memory.

The baselines show that this parity is not automatic. Three of the four FP4 caches are significantly below FP8 on GSM8K (-2.8 to -4.6 points) although they compute attention in BF16, and all four score below FP8 on AIME and LongBench-v2, though not significantly. In this comparison the cache format, not the attention precision, determines accuracy: despite the less precise attention arithmetic, NibbleMLA is the only 4-bit cache that stays within two points of FP8 on GSM8K, AIME, and LongBench-v2.

Table 1: **Accuracy on DeepSeek-V4-Flash.** All baselines dequantize the cache and compute attention in BF16; only NibbleMLA computes both attention GEMMs in FP4. The B/token column is the cache size in bytes per token and *Attn.* the attention-compute precision. RULER: 13 tasks at 128K, mean recall. LongBench-v2: 503 questions, 128K. †Significantly below FP8 (paired McNemar, $p < 10^{-3}$).

Method	Cache scale	B/token	Attn.	GSM8K	AIME	RULER	LongBench-v2
`fp8`	FP8	584	BF16	0.958	0.483	0.920	0.475
`sage3`	per-16 ch., centered	288	BF16	0.944	0.417	0.910	0.453
`group16_fp4`	per-16 ch.	288	BF16	0.930†	0.450	0.914	0.467
`scalesearch_fp4`	per-16 ch., searched	288	BF16	0.918†	0.383	0.918	0.467
`naive_fp4`	per-token	257	BF16	0.912†	0.417	0.914	0.451
NibbleMLA	per-token + H_{512}	**257**	**FP4**	0.960	0.467	0.913	0.485

Table 2: **Decode-attention latency** (µs) per (batch, top-k) shape on a Blackwell GPU (s_{kv}=8192, d=512, 64 query heads; kernel-only timing, minimum over three interleaved rounds of 100 runs). *Mean* is the arithmetic mean over the nine shapes; on the speedup rows it is the ratio of the mean latencies. Both baselines dequantize to BF16 for attention. *$(8, 512)$ is the official kernel's own documented slow shape.

Batch	8			64			128			
Top-k	128	512	2048	128	512	2048	128	512	2048	Mean
FP8 cache, 584 B	11.23	11.90	14.35	12.20	21.33	41.30	13.37	24.75	68.09	24.28
DeepSeek-V4.1 FP4, 288 B	9.80	18.40*	13.60	10.81	19.45	34.29	11.99	20.70	55.79	21.65
NibbleMLA, 257 B	9.92	10.74	12.58	10.86	19.38	32.54	12.07	20.45	54.99	**20.39**
Speedup vs. FP8	1.13	1.11	1.14	1.12	1.10	1.27	1.11	1.21	1.24	**1.19**
Speedup vs. FP4 288 B	0.99	1.71*	1.08	1.00	1.00	1.05	0.99	1.01	1.01	**1.06**

On RULER at 128K, NibbleMLA reaches a mean recall of .913 against .920 for FP8. Retrieval is unaffected—the eight needle-in-a-haystack tasks score .995 for both, and question answering is 3.0 points higher—while the deficit is concentrated in variable tracking and common/frequent-word extraction (-3.8 to -6.4 points), tasks that aggregate information over many positions. All six configurations lie within one point of each other overall, so RULER at this length does not separate the formats (per-task results in Appendix B).

5.4 What makes the single scale work

The co-rotation. `naive_fp4` and NibbleMLA use the same 257 B/token layout; NibbleMLA adds the co-rotation and the write-time scale search, and it computes attention in FP4 rather than BF16. Adding these raises GSM8K from .912 to .960, AIME from .417 to .467, and LongBench-v2 from .451 to .485 (Table 1). Because the change in attention precision works against NibbleMLA, and §5.2 shows it to be numerically neutral, the gain comes from making the single scale accurate—through the co-rotation and the scale search, which we do not separate here. Figure 1, which uses the plain scale $\max_c |x_{t,c}|/6$ without search, isolates the co-rotation at the level of the stored codes.

Ignoring the constraint. Proposition 1 predicts that a per-16-channel cache cannot be fed to a 4-bit PV GEMM as is. A configuration that does so anyway—SageAttention3-style per-16-channel scales with 4-bit attention—produced degenerate output at every context length we evaluated, so no score is reported. The per-16-channel caches in Table 1 are therefore all evaluated with BF16 attention.

5.5 Kernel latency

Table 2 compares the fused decode-attention kernel with the FP8 kernel and the official DeepSeek-V4.1 FP4 kernel, all timed directly on the same GPU. Averaged over the nine shapes, NibbleMLA

decodes in $20.4\,\mu s$ against 24.3 for FP8 and 21.7 for the official FP4 kernel—$1.19\times$ faster than FP8 and $1.06\times$ faster than the official kernel. The gain over FP8 is uniform: NibbleMLA is faster on all nine shapes, by 1.10–$1.27\times$, with the largest margins at top-k 2048, where the most cache bytes are read per query. Against the official FP4 kernel—which dequantizes the cache to BF16—NibbleMLA is faster on six of the nine shapes and behind on the other three by less than 1.3%, and it stays faster on average even when that kernel's documented slow shape at $(8, 512)$ is set aside (21.6 vs $22.1\,\mu s$). It reaches this while running FP4 on both attention GEMMs and storing 11% fewer cache bytes (257 versus 288). These are operator-level measurements of decode attention; the protocol is in Appendix D.

6 Related Work

Latent attention and its caches. MLA (DeepSeek-AI et al., 2024; 2025) compresses the KV cache architecturally by storing one latent per token that serves as key and value; DeepSeek-V4 (DeepSeek-AI et al., 2026b) extends it with compressed and sparse attention. Such latents are typically cached in FP8. Concurrently, DeepSeek-V4.1 (DeepSeek-AI et al., 2026a) caches the main latent in FP4 with per-16-channel scales and dequantizes it to BF16 for attention; its $890\,\mathrm{B/token}$ whole-model figure additionally reflects cross-layer KV reuse, which is orthogonal to and composable with our per-layer format (Appendix E). Section 3 explains why a per-16-channel latent cannot be fed to FP4 attention directly.

KV-cache quantization. KIVI (Zirui Liu et al., 2023), KVQuant (Hooper et al., 2025), QServe (Lin et al., 2025), Atom (Zhao et al., 2024), and ZipCache (He et al., 2024) store keys and values in 2–8 bits and dequantize before, or inside, a higher-precision attention computation. Several quantize keys per channel and values per token (Zirui Liu et al., 2023; Hooper et al., 2025)—a split that presumes separate key and value tensors. For a shared latent, this split turns into the scale-axis constraint: the same tensor must satisfy both GEMMs, and only a per-token scalar does. ZipCache's token-by-channel separable scales are the closest precedent to the factorized form $\alpha_t b_c$ of Proposition 1.

Low-bit attention. FlashAttention-3 (Shah et al., 2024) computes attention in FP8, and SageAttention (Zhang et al., 2025b;a; 2026a) quantizes the transient attention operands, progressing from INT8 to INT4/FP8 and to microscaled FP4 on Blackwell; Attn-QAT (Zhang et al., 2026b) trains models to tolerate 4-bit attention. These methods quantize operands on the fly and leave the stored cache unchanged. NibbleMLA instead makes the persistent, paged latent itself the FP4 operand of both GEMMs, which is where the scale-axis constraint arises.

Rotations for quantization. Hadamard and learned rotations suppress outliers before low-bit quantization (Tseng et al., 2024; Ashkboos et al., 2024; Liu et al., 2025). Recent FP4 analyses find that such global rotations bring little or even hurt when scales are shared by only 16 or 32 elements (Egiazarian et al., 2026; Shao et al., 2025). NibbleMLA uses a fixed Hadamard rotation in the opposite regime—one scale for all 512 coordinates—where equalizing energy across the whole vector is precisely what a single scale needs, and where the scale-axis constraint rules out the small-block alternative for PV.

7 Conclusion

A shared MLA latent is read along two axes, and block-scaled FP4 tensor cores accept scales along only one axis per GEMM. We showed that this leaves a single per-token scale as the only layout both attention GEMMs can consume, which explains why FP4 latent caches are dequantized before attention, and that a zero-byte Hadamard co-rotation makes this layout accurate. The resulting NibbleMLA cache stores $257\,\mathrm{B/token}$ ($0.44\times$ FP8), computes QK and PV in FP4 directly from it, matches BF16 attention on the same cache, matches the FP8 cache within statistical noise on downstream tasks, and runs decode attention 1.10–$1.27\times$ faster than FP8 and, on average, faster than the official FP4 kernel that dequantizes. The per-layer format composes with cross-layer cache reuse, and the constraint itself applies to any tensor that is contracted along two axes under block-scaled arithmetic.

Limitations. All results are for one model, DeepSeek-V4-Flash; the analysis covers any shared-latent MLA, but accuracy parity must be re-established per model. The constraint is derived from the block-scaled MMA semantics of NVIDIA Blackwell, and hardware that scales along other axes would change the feasible set.

USE OF LARGE LANGUAGE MODELS

Large language models were used as general-purpose assistants for source-code editing, refactoring, and prose polishing during the preparation of this manuscript. They were not used to generate experimental results, and no reported number originates from an LLM. All LLM-assisted code was read, tested, and verified by the authors against the deployed kernel, and all claims were checked against the underlying result files. The authors take full responsibility for the correctness of the method, the code, the experiments, and the text; no LLM meets the criteria for authorship.

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, 2024. URL https://arxiv.org/abs/2404.00456.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL https://arxiv.org/abs/2412.15204.

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, 2021. URL https://arxiv.org/abs/2110.14168.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Yang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jin Chen, Jingyang Yuan, Junjie Qiu, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruizhe Pan, Runxin Xu, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Size Zheng, T. Wang, Tian Pei, Tian Yuan, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Liu, Xin Xie, Xingkai Yu, Xinnan Song, Xinyi Zhou, Xinyu Yang, Xuan Lu, Xuecheng Su, Y. Wu, Y. K. Li, Y. X. Wei, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Zheng, Yichao Zhang, Yiliang Xiong, Yilong Zhao, Ying He, Ying Tang, Yishi Piao, Yixin Dong, Yixuan Tan, Yiyuan Liu, Yongji Wang, Yongqiang Guo, Yuchen Zhu, Yuduan Wang, Yuheng Zou, Yukun Zha, Yunxian Ma, Yuting Yan, Yuxiang You, Yuxuan Liu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhewen Hao, Zhihong Shao, Zhiniu Wen, Zhipeng Xu, Zhongyu Zhang, Zhuoshu Li, Zihan Wang, Zihui Gu, Zilin Li, and Ziwei Xie. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024. URL https://arxiv.org/abs/2405.04434.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li,

Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI, :, Anyi Xu, B. Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, C. C. Yu, Chao Jin, Chaofan Lin, Chen Dong, Chenbing Wang, Chenfan Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyuan Zhang, Chenhao Xu, Chenqi Zhao, Chenze Shao, Chuhao Wang, Chuqi Zhang, Damai Dai, Dejian Yang, Deli Chen, Di Huang, Di Wu, Donghao Li, Erhang Li, Eric Fu, F. Zhou, Fangwei Zhou, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanglin Li, Guanting Chen, Guoai Cao, Guofan Fan, Guolai Meng, Guowei Li, Haichuan Zhang, Haiyang Ma, Haiyang Shen, Han Li, Han Yu, Han Zhang, Hangyuan Deng, Hanwei Xu, Hanxiang Xu, Hanxun Zhong, Hao Guo, Hao Jiang, Hao Li, Hao Qin, Haodong Wen, Haofen Liang, Haofeng Huang, Haohua Liu, Haoling Zhang, Haoming Luo, Haoran Yang, Haotian Xu, Haotian Yuan, Haoting Huang, Haowen Luo, Haoyang Cai, Haoyu Chen, Haozhe Ji, Hengran Zhang, Hengrui Wang, Hengxu Wu, Honghui Ding, Hongxuan Tang, Huadong Wang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J. Yang, J. H. Jin, J. H. Zhang, J. X. Zou, Jia Yu, Jiahui Zhou, Jiajun Chen, Jialiang Huang, Jialin Zhao, Jiamin Tang, Jian Zhou, Jianan Tong, Jianwen Li, Jiaqi Zhu, Jiarui Wang, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiaying Ding, Jibai Lu, Jiewen Hu, Jin Yan, Jincheng Zhai, Jingchang Chen, Jingcheng Hu, Jingli Zhou, Jingsheng Xu, Jingting Xiang, Jingyan Yun, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jinpeng Wang, Jinyi Chen, Jinyi Hu, Jiping Yu, Jueliang Guo, Junbo Pei, Junbo Sun, Junguang Jiang, Junjie Qiu, Junkang Zhou, Junqi Liu, Junren Li, Junxian Li, Junxiao Song, Junyi Guo, Kai Dong, Kaifeng Chen, Kaige Gao, Kang Guan, Kangdong Yuan, Ke Hong, Ke Xu, Kefan Zhao, Kexin Ji, Kexin Zhang, Kexing Zhou, Kuai Yu, Lan Zhang, Lean Wang, Lecong Zhang, Lei Wang, Letian Gao, Liang Zhao, Liansheng Xu, Lihua Guo, Lingxiao Luo, Lingyue Fu, Litao Deng, Litong Wang, Liyue Zhang, Longhao Chen, Lu Chen, Luotian Huang, Luyao Ma, Luyao Wang, M. S. Di, Max Mei, Menghao Ye, Miao Cui, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingjing Zhang, Mingqi Wei, Mingshu Chen, Mingxing Liu, Mingxu Zhou, Mingyu Xu, Mingyu Yang, Mingze Wang, Muyang Chen, Ni Shentu, Ning Wang, Niufang Ning, Panpan Huang, Peixin Cong, Peiyi Wang, Peiyuan Xin, Pengfei Ren, Pengfei Yan, Pengle Zhang, Qi Kang, Qi Tang, Qiancheng Wang, Qiang Li, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qizhou Guo, Rongxian Xu, Rui Ding, Rui Hu, Rui Tian, Rui Yu, Ruidong Zhu, Ruifan Xu, Ruihan Yang, Ruihang Xia, Ruijie Lu, Ruilin Geng, Ruipeng Hong, Ruiqi Ge, Ruisong Zhang, Ruize Sun, Ruizhe Pan, Runji Wang, Runqian Chen, Runxin Xu, Ruohong Tian, Ruomeng Shen, Ruoyu Zhang, Ryan X., S. H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoyuan Chen, Shengding Hu, Shengkai Lin, Shengwen Ran, Shengyu Liu, Shengyuan Jia, Shi Bai, Shi Feng, Shicheng Xu, Shichun Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shiyuan Feng, Shufan Gong, Shuhan Lin, Shuiping Yu, Shunfeng Zhou, Shuo Yang, Shuomeng Wang, Shuting Guo, Shuting Pan, Shuying Yu, Sinuo Cao, Siyi Lin, Sizhe

Chen, Songyang Chen, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tongrui Xiong, Wangding Zeng, Wei Liu, Wei Zhang, Weibin Xu, Weihao Zeng, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Shao, Wenkai Yang, Wenli Zhang, Wenlu Wang, Wenlve Huang, Wenqian Yan, Wentao Zhang, Xi Gao, Xiang He, Xiang Li, Xiangli Li, Xiangwen Wang, Xiangying Zhang, Xiankui Wei, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojian Qu, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaoyao Zou, Xiaoyuan Li, Xicheng Guo, Xieting Chu, Xin Cheng, Xin Liu, Xin Xie, Xinbo Xu, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xintong Yao, Xinyang Chen, Xinyong Jiang, Xinyu Yang, Xinyu Yang, Xu Chen, Xuanyu Wang, Xubei Zhong, Xuecheng Su, Xuejie Liu, Xuheng Lin, Xujie Fan, Xuncheng Zhao, Xuwei Fu, Y. C. Yan, Y. H. Jiang, Y. T. Wu, Y. W. M., Y. Z. Wang, Yafei Gao, Yang Yang, Yang Zhang, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Meng, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yaoyang Ye, Yehang Yin, Yexinrui Wu, Yi Qian, Yi Tao, Yi Yu, Yichao Zhang, Yichen Jiang, Yicheng Wang, Yifan Ding, Yifan Shi, Yifeng Peng, Yifeng Zhai, Yijia Wu, Yiliang Xiong, Yilun Wang, Ying He, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yiping Wang, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyao Yang, Yiyuan Liu, Yizai Cai, Yizhen Wei, Yizhi Wang, Yonglun Yang, Yongqi Zhuo, Yongqiang Guo, Yongtong Wu, Yu Wu, Yu Zhang, Yuan Bian, Yuan Cheng, Yuan Ou, Yuan Sun, Yuanfan Xu, Yuanhang Sun, Yuanhao Li, Yuchen Liu, Yuchen Yao, Yudong Han, Yuduan Wang, Yuhan Wu, Yuhao Meng, Yuheng Zou, YuKun Li, Yunchuan Wang, Yunfan Xiao, Yunfan Xiong, Yupeng Chen, Yuqian Cao, Yuqian Wang, Yuqing Chen, Yushun Zhang, Yutong Lin, Yuwei Xiao, Yuxian Gu, Yuxiang Chen, Yuxiang Huang, Yuxiang Luo, Yuxiang You, Yuxin Chen, Yuxin Xiang, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhe Guo, Yuzhen Huang, Yuzhuo Bai, Z. Y. Z., Zanlin Ni, Zehao Wang, Zehua Zhao, Zehui Ren, Zejun Zhao, Zhangli Sha, Zhanying Wang, Zhaochen Zhang, Zhaoshuai Du, Zhe Fu, Zhean Xu, Zhenda Xie, Zheng Liu, Zhengyan Zhang, Zhenhua Dong, Zhewen Hao, Zhibang Wang, Zhibin Gou, Zhicheng Ma, Zhihao Li, Zhihong Shao, Zhihuan Huang, Zhijie Li, Zhirui Lu, Zhixian Huang, Zhixuan Chen, Zhixuan Chen, Zhixuan Pan, Zhiyu Wu, Zhizhou Ren, Zhu He, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zili Zhang, Zilin Li, Zilong Hou, Zilong Lyu, Ziqiao Wang, Ziwei Xie, Ziya Zhang, Ziyi Gao, Zizheng Pan, Zonglin Li, Zongqing Yao, Zui Chen, Zuofan Wu, Chenchen Ling, Chengyu Hou, Chong Chen, D. Li, Di Qi, Dongjie Ji, Fang Wei, Fanyi Xia, Fei Xie, Feiyi Tan, Hailong Guo, Haiyan Zhai, Hui Zhou, Huihui Tan, Huijie Li, Jia Luo, Jia Song, Jialu Cai, Jian Liang, Jiangting Zhou, Jiaqi Gao, Jiayi Shao, Jie Chen, Jieyu Yang, Jin Chen, Jingde Zhang, Jingzi Zhou, Jinqian Wang, Jinyang Liu, JinZhao Sun, Junhua Ling, Junmin Zheng, Kaicheng Yang, Ke Xu, Le Su, Leyi Xia, Liangfeng Ding, Lin Zhuo, Linwang Ma, Linyan Zhu, Liyu Cai, Luqi Yao, M. K. Zhang, Meng Li, Miao Lin, Miaojun Wang, Min Zhang, Mingming Li, Mingming Wang, Mingze Yin, Minmin Han, Nan Cao, Ning Wang, Ningxin Ma, Panpan Wang, Peihan Lin, Peng Sun, Peng Zhang, Qian Ying, Qiang Xiang, Qiao Wang, Qingmiao Mao, Qiwei Jiang, Rongli Jin, Ruyi Chen, Sha Tao, Shangmian Sun, Shaoqing Wu, Shichao Zou, Si Lei, Tianyang Zhang, Tianyu Sun, Tingting Yin, W. L. Xiao, Wei An, Wei Li, Wei Wang, Weiwei Lin, Wenqing Hou, X. Lin, Xiangfei Meng, Xianzhu Huang, Xiao Peng, Xiaoqian Li, Xiaoting Zhang, Xiaowen Sun, Xiaoxiang Wang, Xiaoyu Ye, Xinrou Zhang, Xinyu Zhang, Xue Cao, Xueyin Chen, Yanan Zhou, Yanhong Xu, Yao Xia, Yao Xu, Yi Shao, Yihong Zhang, Yiling Ma, Ying Tang, Yining Lou, Yiru Chen, Yishi Piao, Yixuan Chen, Yong Xiong, Yuchen Xuan, Yuehan Yang, Yuer Xu, Yukun Zha, Yunxian Ma, Yuping Lin, Yuting Yan, Yutong Xie, Yuwen Sheng, Yuxuan Zhu, Zekai Zhang, Zhe Ju, Zhenzhen Lin, Zheren Gao, Zheyang Sun, Zhigang Yan, Zhongyu Wu, Zi Wang, Zihua Qu, Ziling Yan, and Ziyi Wan. Deepseek-v4.1-flash: Pushing the limits of kv cache compression, 2026a. URL https://arxiv.org/abs/2609.19969.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang

Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026b. URL https://arxiv.org/abs/2606.19348.

Vage Egiazarian, Roberto L. Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Bridging the gap between promise and performance for microscaling fp4 quantization, 2026. URL https://arxiv.org/abs/2509.23202.

Tanmaey Gupta, Hayden Prairie, Xiaoxia Wu, Reyna Abhyankar, Qingyang Wu, Austin Silveria, Pragaash Ponnusamy, Jue Wang, Ben Athiwaratkun, Leon Song, Tri Dao, Daniel Y. Fu, and Chris De Sa. Search your block floating point scales!, 2026. URL https://arxiv.org/abs/2605.12464.

Yefei He, Luoming Zhang, Weijia Wu, Jing Liu, Hong Zhou, and Bohan Zhuang. Zipcache: Accurate and efficient kv cache quantization with salient token identification, 2024. URL https://arxiv.org/abs/2405.14256.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization, 2025. URL https://arxiv.org/abs/2401.18079.

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?, 2024. URL https://arxiv.org/abs/2404.06654.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving, 2025. URL https://arxiv.org/abs/2405.04532.

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, 2025. URL https://arxiv.org/abs/2405.16406.

Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, Stosic Dusan, Venmugil Elango, Maximilian Golub, Alexander Heinecke, Phil James-Roxby, Dharmesh Jani, Gaurav Kolhe, Martin Langhammer, Ada Li, Levi Melnick, Maral Mesmakhosroshahi, Andres Rodriguez, Michael Schulte, Rasoul Shafipour, Lei Shao, Michael Siu, Pradeep Dubey, Paulius Micikevicius, Maxim Naumov, Colin Verrilli, Ralph Wittig, Doug Burger, and Eric Chung. Microscaling data formats for deep learning, 2023. URL https://arxiv.org/abs/2310.10537.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision, 2024. URL https://arxiv.org/abs/2407.08608.

Yuantian Shao, Peisong Wang, Yuanteng Chen, Chang Xu, Zhihui Wei, and Jian Cheng. Block rotation is all you need for mxfp4 quantization, 2025. URL https://arxiv.org/abs/2511.04214.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks, 2024. URL https://arxiv.org/abs/2402.04396.

Jintao Zhang, Haofeng Huang, Pengle Zhang, Jia Wei, Jun Zhu, and Jianfei Chen. Sageattention2: Efficient attention with thorough outlier smoothing and per-thread int4 quantization, 2025a. URL https://arxiv.org/abs/2411.10958.

Jintao Zhang, Jia Wei, Haofeng Huang, Pengle Zhang, Jun Zhu, and Jianfei Chen. Sageattention: Accurate 8-bit attention for plug-and-play inference acceleration, 2025b. URL https://arxiv.org/abs/2410.02367.

Jintao Zhang, Jia Wei, Pengle Zhang, Xiaoming Xu, Haofeng Huang, Haoxu Wang, Kai Jiang, Jianfei Chen, and Jun Zhu. Sageattention3: Microscaling fp4 attention for inference and an exploration of 8-bit training, 2026a. URL https://arxiv.org/abs/2505.11594.

Peiyuan Zhang, Matthew Noto, Wenxuan Tan, Chengquan Jiang, Will Lin, Wei Zhou, and Hao Zhang. Attn-qat: 4-bit attention with quantization-aware training, 2026b. URL https://arxiv.org/abs/2603.00040.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving, 2024. URL https://arxiv.org/abs/2310.19102.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi : Plug-and-play 2bit kv cache quantization with streaming asymmetric quantization. 2023. doi: 10.13140/RG.2.2.28167.37282. URL https://www.researchgate.net/doi/10.13140/RG.2.2.28167.37282.

A Proof of Proposition 1

Setting. The cache stores codes $q_{t,c} \in \mathrm{E2M1}$ and scales with $\hat{C}_{t,c} = S_{t,c}\, q_{t,c}$. A block-scaled MMA computes Eq. (2). A layout is *directly consumable* by a GEMM if the codes q enter the MMA unchanged as one operand and the GEMM result equals the intended contraction of $\hat{C}$ after (a) choosing the block scales of both operands, (b) multiplying the entries of the other, transient operand by arbitrary factors before it is quantized, and (c) multiplying the output by a static factor. The scales of token t are fixed when t is written, as a function of C_t only (*write-once*).

QK. The logits are $s_t = \tau \sum_c \mathbf{q}_c \, S_{t,c} \, q_{t,c}$. Map the cache to operand B with $n = t$ and $k = c$. If $S_{t,c} = u(t, \lfloor c/16 \rfloor)$, set $S^B_{t,\lfloor c/16 \rfloor} = u(t, \lfloor c/16 \rfloor)$; the query supplies A with its own block scales. This proves (i). If $S_{t,c} = \alpha_t b_c$, set $S^B_{t,\cdot} = \alpha_t$ and multiply the query entries by b_c before quantizing them (step b), so the form of (iii) is consumable by QK.

PV, sufficiency. The output is $o_{h,c} = \sum_t x_{h,t} \, S_{t,c} \, q_{t,c}$ for query row h, with $x = p$. Map the cache to operand A with $m = c$ and $k = t$, and the probabilities to operand B with $n = h$. If $S_{t,c} = \alpha_t b_c$, set $S^A \equiv 1$, multiply $x_{h,t}$ by α_t before quantization (step b), and multiply the output row c by b_c (step c).

PV, necessity. Any consumable realization computes $\sum_t S^A_{c,\lfloor t/16 \rfloor} \, q_{t,c} \, S^B_{h,\lfloor t/16 \rfloor} \, g_{h,t} \, x_{h,t} \, r_c$, where g is the factor applied to the transient operand and r the static output factor; g cannot depend on c because the transient operand has no channel index, and r cannot depend on t because it is applied after the sum. Matching this to $\sum_t x_{h,t} S_{t,c} q_{t,c}$ for all codes and all x requires $S_{t,c} = S^A_{c,\lfloor t/16 \rfloor} \, \gamma_t \, r_c$, where γ_t collects the token-dependent factors ($S^B g$ must not depend on h, since S does not). Under write-once, $S^A_{c,\lfloor t/16 \rfloor}$ is shared by 16 tokens yet must be determined by each token's own latent; for arbitrary latents it is therefore independent of the tokens, i.e. static. Absorbing it into r_c gives $S_{t,c} = \alpha_t b_c$. Statement (iii) combines (ii) with the QK case above. □

Layouts outside the premises. Dropping write-once admits a 16-token × 16-channel tile scale, which is legal for both GEMMs but ties the scales of 16 consecutive tokens together, so tokens can no longer be written independently to a paged cache (a residual buffer as in KIVI (Zirui Liu et al., 2023) would be needed). Dropping direct consumption admits the $288\,\mathrm{B}/\text{token}$ per-16-channel format on PV at the cost of either dequantizing and re-quantizing V along tokens (about $8\times$ the arithmetic; 0.083–$0.22\times$ the throughput in two independent implementations) or re-quantizing P once per channel block (32 times at MMA $N{=}16$). A static per-channel vector b is permitted by the proposition; NibbleMLA does not need one because the co-rotation already equalizes the channels ($b \equiv 1$).

B Full Results

B.1 RULER at 128K

Tables 3 and 4 give the full 13-task RULER (Hsieh et al., 2024) results at 128K (100 samples per task, 1300 per method; chat template, non-thinking; greedy; at most 128 new tokens; RULER's string-match recall). No configuration produced empty responses.

Table 3: RULER at 128K by task category: NIAH (eight needle tasks), VT (variable tracking), CWE/FWE (common/frequent-word extraction), QA (two QA tasks). Mean recall, exact match, and the empty-response rate.

Method	NIAH	VT	CWE	FWE	QA	Overall	Exact	Empty
`fp8` (BF16 attn.)	0.995	0.992	0.838	0.983	0.596	0.920	0.902	0%
`sage3` (BF16 attn.)	0.989	0.954	0.740	0.990	0.619	0.910	0.885	0%
`group16_fp4` (BF16 attn.)	0.993	0.982	0.791	0.970	0.598	0.914	0.893	0%
`scalesearch_fp4` (BF16 attn.)	0.991	0.964	0.817	0.983	0.623	0.918	0.889	0%
`naive_fp4` (BF16 attn.)	0.997	0.982	0.741	0.957	0.615	0.914	0.885	0%
NibbleMLA (FP4 attn.)	0.995	0.928	0.800	0.930	0.626	0.913	0.885	0%

B.2 Reasoning with paired statistics

Table 5 lists, for every method, the accuracy with its Wilson 95% interval, the discordant pairs against FP8 (items the method solves and FP8 does not / vice versa), and the paired difference $d = (w - \ell)/n$ with a 95% Wald interval, $\mathrm{SE} = \sqrt{(w + \ell) - (w - \ell)^2/n} \,/\, n$, together with the exact McNemar p. AIME contains 60 problems, so its intervals are wide for every method.

Table 4: RULER at 128K per task (mean recall; s = single, mk = multikey, mq = multiquery, mv = multivalue).

	NIAH								VT	CWE	FWE	QA	
Method	s1	s2	s3	mk1	mk2	mk3	mq	mv	vt	cwe	fwe	qa1	qa2
fp8	0.980	1.00	1.00	1.00	1.00	0.990	1.00	0.990	0.992	0.838	0.983	0.642	0.550
sage3	0.980	0.980	0.990	1.00	1.00	1.00	1.00	0.960	0.954	0.740	0.990	0.688	0.550
NibbleMLA	0.980	1.00	1.00	1.00	1.00	0.990	1.00	0.993	0.928	0.800	0.930	0.662	0.590

Table 5: GSM8K (n=500) and AIME (n=60) with paired statistics against FP8. w/ℓ: discordant pairs; d: paired difference in points with 95% interval; p: exact McNemar.

	GSM8K			AIME		
Method	Acc. [Wilson]	w/ℓ	d [95%]; p	Acc. [Wilson]	w/ℓ	d [95%]; p
fp8	0.958 [0.937,0.972]	–	–	0.483 [0.362,0.607]	–	–
sage3	0.944 [0.920,0.961]	3/10	-1.4 [$-2.8, +0.0$]; 0.092	0.417 [0.301,0.543]	2/6	-6.7 [$-15.8, +2.4$]; 0.289
group16_fp4	0.930 [0.904,0.949]	0/14	-2.8 [$-4.2, -1.4$]; 1.2×10^{-4}	0.450 [0.331,0.575]	1/3	-3.3 [$-9.8, +3.1$]; 0.625
scalesearch_fp4	0.918 [0.891,0.939]	1/21	-4.0 [$-5.8, -2.2$]; 1.1×10^{-5}	0.383 [0.271,0.510]	2/8	-10.0 [$-20.0, +0.0$]; 0.109
naive_fp4	0.912 [0.884,0.934]	1/24	-4.6 [$-6.5, -2.7$]; 1.5×10^{-6}	0.417 [0.301,0.543]	1/5	-6.7 [$-14.5, +1.2$]; 0.219
NibbleMLA	0.960 [0.939,0.974]	4/3	$+0.2$ [$-0.8, +1.2$]; 1.000	0.467 [0.346,0.591]	4/5	-1.7 [$-11.5, +8.1$]; 1.000

B.3 LongBench-v2

Protocol. Some LongBench-v2 (Bai et al., 2025) contexts exceed the model's 128K window. We truncate every context to 128K tokens by removing the middle (keeping head and tail), so that all 503 questions complete for all methods under the same budget. Table 6 breaks the headline result down by context length. NibbleMLA differs from FP8 on 85 questions (45 wins, 40 losses; McNemar $p = 0.665$; Wilson intervals FP8 $[.432, .519]$, NibbleMLA $[.442, .529]$).

Table 6: LongBench-v2 by context-length bucket (one request in flight, 128K truncation, 503 questions; short/medium/long $= 180/215/108$). No difference to FP8 is significant.

Method	Short	Medium	Long	Overall
fp8	0.539	0.451	0.417	0.475
sage3	0.517	0.405	0.444	0.453
group16_fp4	0.528	0.470	0.361	0.467
scalesearch_fp4	0.539	0.447	0.389	0.467
naive_fp4	0.522	0.414	0.407	0.451
NibbleMLA	0.522	0.470	0.454	0.485

C The Co-Rotation in Detail

H_{512} is the normalized 512×512 Sylvester–Hadamard matrix, applied with the fast Walsh–Hadamard transform (9 butterfly stages). It is applied to each latent when the entry is written, fused with the RoPE step, and to each query before QK; the attention output is mapped back with $H_{512}^{\top} = H_{512}$. Figure 3 complements Figure 1 on the same layer: the co-rotation compresses the per-channel range, reduces the share of entries that round to code 0 from 24.4% to 10.4%, and nearly halves the per-token reconstruction error under a single scale.

D Kernel Details

Measurement protocol. All three kernels are timed directly on the same Blackwell GPU (152 SMs, CUDA 13.0) with kernel-only Kineto traces, GPU-solo, at s_{kv}=8192, d=512, 64 query heads; each configuration runs 100 times in each of three interleaved rounds and we report the minimum. Within-session timing variation is 0.1–0.4%, so the per-shape numbers in Table 2 are reproducible to well under a percent. Averaged over the nine shapes, NibbleMLA runs in 20.4 µs against 24.3 for FP8 ($1.19\times$) and 21.7 for the official FP4 kernel ($1.06\times$); against the official kernel it is faster on

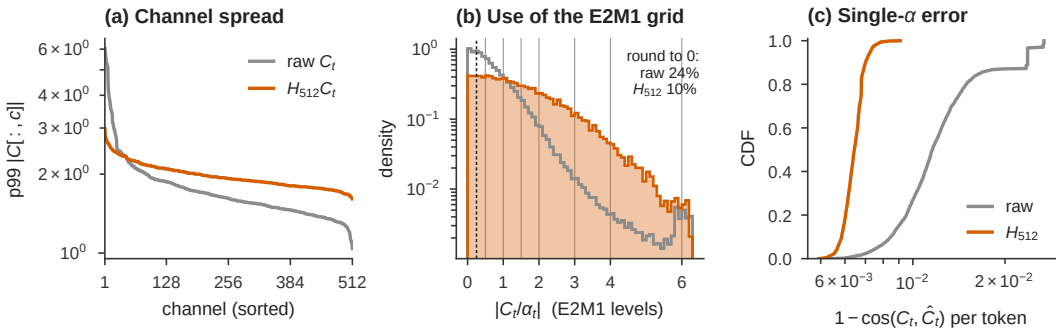

Figure 3: Layer 32, 2000 tokens, one scale per token ($\alpha_t = \max_c |x_{t,c}|/6$). *(a)* Per-channel p99 magnitude, channels sorted: H_{512} removes the heavy head of outlier channels. *(b)* Scaled entries $|C_t/\alpha_t|$ against the E2M1 levels: without H_{512}, 24.4% of entries (25.1% of content channels) round to code 0; with H_{512}, 10.4%. *(c)* Per-token reconstruction error $1 - \cos(C_t, \hat{C}_t)$: median 1.2×10^{-2} without H_{512} and 6.4×10^{-3} with it.

Algorithm 1: NibbleMLA decode attention (one query head, one tile of cache entries).

1. **Gather.** By page index, load the E2M1 codes $\{q_t\}$ and scales $\{\alpha_t\}$ of the selected entries in place; nothing is dequantized.
2. **QK.** Compute $\tau\, \tilde{\mathbf{q}}^\top (\alpha_t q_t)$ as a block-scaled FP4 MMA: the rotated query as a two-term E2M1 residual with per-16-channel E4M3 scales, the cache codes with block scale α_t on all channel blocks; FP32 accumulation; add a_t.
3. **Softmax.** One FP32 online softmax over all branches and the sink. Form $x_t = p_t\, \alpha_t$ and quantize it to a two-term NVFP4 residual with per-16-token E4M3 scales and unit row scale.
4. **PV.** Compute $\sum_t x_t\, q_t$ as a block-scaled FP4 MMA with unit scale on the cache operand; FP32 accumulation.
5. **Finalize.** Normalize, map back with $H_{512}^\top$, undo RoPE on the rotary slice, and apply the output projection.

Figure 4: The decode kernel. Every accuracy result in the paper uses exactly these numerics.

six of the nine shapes and behind on the other three by less than 1.3%, and it stays faster on average (21.6 vs 22.1 µs) even with that kernel's slow shape at $(8, 512)$ dropped. The kernel passes an output-accuracy gate at every one of 20 shapes (minimum row cosine 0.99985 against the reference, gate 0.999) and is bit-identical to the version used for the accuracy experiments. Removing the per-row scale reduction of §4.3 reduced latency by 23.1% at batch 128, top-k 2048.

E Byte Accounting

Per-layer bytes. Table 7 lists the per-layer footprint of the main latent cache. Both FP4 formats store 512 E2M1 codes (256 B) and differ in scale metadata and in how attention is computed. DeepSeek-V4.1-Flash (DeepSeek-AI et al., 2026a) reports a whole-model figure of 890 B/token, which covers the cache of the entire model, including the indexer keys, and relies mainly on cross-layer KV reuse; it is not a per-layer format number. Our experiments use DeepSeek-V4-Flash, which has no cross-layer reuse. The 257 B/token format is orthogonal to such reuse and could be combined with it; we make no whole-model comparison with DeepSeek-V4.1.

Table 7: Per-layer bytes per token of the main latent cache and attention precision.

Format	Bytes/token	Attention
BF16 latent	1024	BF16
FP8 (DeepSeek-V4-Flash production)	584	BF16
FP4, per-16-channel E4M3 scales (DeepSeek-V4.1)	288	BF16 (dequantized)
NibbleMLA: FP4, one E4M3 scale per token	257	FP4 (both GEMMs)

F CAPABILITY COMPARISON

Table 8 places each method by what it quantizes. KV-cache quantization methods store keys and values in low precision but compute attention in higher precision; FP4 attention methods quantize transient operands and store no compressed cache; the DeepSeek-V4.1 cache stores the MLA latent in FP4 but dequantizes it.

Table 8: Capability comparison. "$\leq$4-bit cache": stored KV or latent at four bits or fewer (INT4 or FP4). "FP4 QK/PV": the GEMM runs on FP4 tensor cores. "Shared latent": designed for a single $K{=}V$ MLA latent.

Method	$\leq$4-bit cache	FP4 QK	FP4 PV	Shared latent
KIVI (Zirui Liu et al., 2023), KVQuant (Hooper et al., 2025)	✓	×	×	×
QServe (Lin et al., 2025), Atom (Zhao et al., 2024)	✓	×	×	×
ZipCache (He et al., 2024)	✓	×	×	×
QuaRot (Ashkboos et al., 2024)	✓	×	×	×
SageAttention3 (Zhang et al., 2026a)	×	✓	✓	×
Attn-QAT (Zhang et al., 2026b)	×	✓	✓	×
DeepSeek-V4.1 FP4 cache (DeepSeek-AI et al., 2026a)	✓	×	×	✓
NibbleMLA	✓	✓	✓	✓