AWSLQ: Activation-Wise Sparse-Ladder Quantization for Accurate W4A4 Single-Load Decoding

Anonymous authors
Paper under double-blind review

Abstract

On FP4-native GPUs, deployed serving stacks already run W4A4—FP4 weights times FP4 activations—at decode, yet the quantization literature retreats to 16-bit decode because W4A4 is where accuracy collapses. Small-batch decode is bound by streaming the weights, so its tensor cores sit mostly idle. AWSLQ, a training-free W4A4 scheme that requires no external calibration corpus, spends that idle compute: it multiplies each weight tile by several FP4 activation terms instead of one, here a *ladder* of N structured-sparse NVFP4 operands, peeled online, that share one weight load. Two-operand ablations had concluded that more sparse computation does not recover the loss; the knee is at $N{=}3$, and by $N{\approx}4$ the ladder is on the W4A16 floor, to within $+0.005$ perplexity on ten of eleven models. A selective reverse-smoothing fold, in the direction SmoothQuant advises against, lowers the weight-only floor itself; the ladder absorbs the activation outliers the fold creates and lands on the lowered floor, while a single dense term stays above it; whether to fold is decided per model from its own generated text at preparation time. A fused Blackwell kernel runs the ladder as one M-stacked sparse GEMM into a single accumulator and beats cuBLASLt's FP4 GEMM across a Llama-3.1-8B decode block at $M{=}16$–64 ($1.26\times$ to $1.07\times$ on an RTX 5090, $1.08\times$ at $M{=}16$ on a B200). AWSLQ matches or beats every published W4A4 method, calibrated or not, on 16 of the 18 model–dataset cells with a published comparator, at coarser block-32 scales, and on GSM8K it matches the accuracy of retreating to 16-bit decode activations. The accuracy comes from the number of FP4 terms that share one weight load. A sparse rung performs half the multiply-accumulates of a dense term, so the ladder also offers an intermediate arithmetic budget, three rungs at one and a half dense terms, which chooses the refined half of each token's channels online.

1 Introduction

LLM serving has two phases with different bottlenecks: prefill is one compute-bound pass over the prompt, and decode is thousands of memory-bound passes, one per generated token, which dominate serving cost (DeepSeek's disclosed V3/R1 deployment assigns 18 of the 22 nodes in each unit to decode (DeepSeek-AI, 2025)). On Blackwell GPUs, W4A4, FP4 weights times FP4 activations, is already the production decode format: NVIDIA's minimum-latency DeepSeek-R1 deployment runs its FFN and MoE GEMMs in FP4×FP4 at every decode step (NVIDIA TensorRT-LLM, 2025), and the major serving stacks ship NVFP4 checkpoints whose activations are quantized to FP4 at run time (LMSYS Org, 2025; NVIDIA, 2026b;c;d). At batch one a decode GEMM is limited by its weight stream, and weight-only W4A16 moves the same 4-bit weights as W4A4, so the two have similar weight-bandwidth lower bounds. From batch ≈16–32 the multiply-accumulates start to matter: W4A16 must expand every weight to BF16 for tensor cores that run at a quarter of the FP4 rate, so BF16 arithmetic can become limiting while the faster FP4 tensor cores still have unused throughput. NVIDIA's guidance switches to activation quantization at batch ≥ 16 (NVIDIA Model Optimizer, 2026), where Marlin's near-ideal region ends (Frantar et al., 2025); Section 6 measures the crossover.

Phase-specialized research systems avoid exactly this operating point (Figure 1): Mix-Quant (Lu et al., 2026b) quantizes prefill to NVFP4 but keeps decode in BF16 because "decoding is much more error-sensitive"; QSpec (Zhao et al., 2025a) permits W4A4 only for speculative drafts that a W4A16 pass verifies (a ∼30% reasoning loss for naive W4A4 against ∼4% for W4A16); and Disaggregated

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

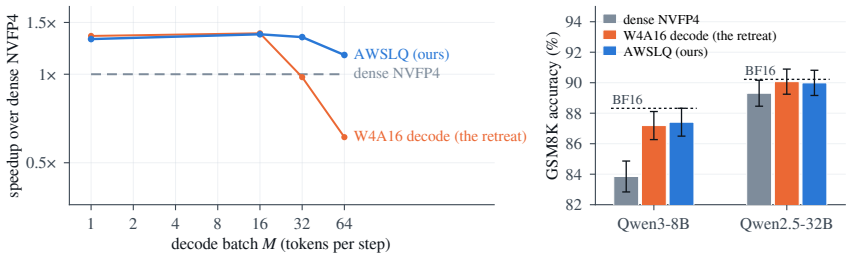


Figure 1: The three decode formats that can follow the same NVFP4 prefill. Left: per-GEMM speed relative to cuBLASLt's dense NVFP4 kernel at the same batch (geometric mean over four Llama-3.1-8B layers, RTX 5090, cold L2; three shapes at $M{=}1$, Appendix D.5). Right: GSM8K accuracy (8-shot, full test set) with binomial standard errors, BF16 dotted. The retreat to 16-bit decode activations keeps accuracy but must expand every weight to BF16, so it loses its lead at $M{=}32$; dense NVFP4 keeps its speed and loses 4.5 points on Qwen3-8B; AWSLQ (SL4) ties the retreat within one standard error and is the fastest of the three from $M{=}16$ to $M{=}64$.

Quantization (Panferov et al., 2026) measures that quantizing decode alone costs two to four times the accuracy loss of quantizing prefill alone, and prescribes weight-only decode.

Our approach and contributions. Memory-bound decode leaves tensor-core throughput unused; we spend it on *more operands*: an activation tile is peeled online into a ladder of N structured-sparse (4:8) NVFP4 operands, each rung the top-magnitude half of the running residual, and all N rungs multiply the *same* weight tile inside one kernel. (1) **Saturation.** The knee is at $N{=}3$, and by $N{\approx}4$ the ladder is within $+0.005$ PPL of W4A16 on ten of eleven models and within $+0.02$ on all; two dense terms on the same load reach the same floor at the same arithmetic budget, and the sparse rung adds a point between them: half a dense term whose kept half the prepare kernel chooses per token, so three rungs sit at a budget and an allocation that none of the matched dense configurations we evaluate realizes (Section 3.2). (2) **A selective reverse fold** lowers the weight-only floor itself; the rungs absorb the outliers it creates, so the ladder lands on the lowered floor while a single dense term stays 0.19–0.48 PPL above it, and a preparation-time probe on self-generated text selects the fold per model (Section 3.3). (3) **A fused multi-term NVFP4 kernel**: one weight load, one accumulator, up to four rungs via M-stacking, bit-exact, and $1.26\times$ to $1.07\times$ faster than cuBLASLt's FP4 GEMM across a Llama-3.1-8B decode block at $M{=}16$–64 on an RTX 5090, $1.08\times$ at $M{=}16$ on a B200 (Sections 4 and 6). (4) **Results.** Zero-calibration W4A4 that matches or beats every published W4A4 method on 16 of the 18 model–dataset cells with a published comparator; on GSM8K the ladder at decode matches the 16-bit-decode retreat, and on MATH-500 it beats uniform dense NVFP4 on both models tested (Section 5).

2 Related Work

Decode precision. Every phase-specialized system we know of—Mix-Quant, HMA-Serve (Wei et al., 2026), QSpec, Lynx (Han et al., 2026) and Disaggregated Quantization (Panferov et al., 2026)—restores full precision at decode; vendor stacks run one uniform NVFP4 format with no controlled public ablation of decode-side accuracy and treat sparse execution per layer as optional (NVIDIA, 2026e); per-phase dispatch itself is prior art we adopt unchanged (Mix-Quant, PMPD (Chen et al., 2025a), FlexQuant (Liu et al., 2025a)).

Multi-term activations. Multi-term residual refinement of activations is REx (Yvinec et al., 2023), data-free and on integer grids; RFQ (Rahman et al., 2026) and ARCQuant (Meng et al., 2026a) add a dense FP4 residual as a second GEMM, and Multi-Scale Dequant (Zheng et al., 2026) stacks dense components along rows to load each weight tile once. The greedy top-2-of-4-pairs peel is the decomposition of TASD (Jeong et al., 2025), which defines the series for any number of terms but evaluates every network at two, on calibration data; ours is its block-scaled FP4 instance at $N \geq 3$ on one weight load, without calibration. SharQ (Meng et al., 2026b) shares one FP4 weight payload between an outlier-sparse NVFP4 operand and a dense NVFP4 residual and finds on Llama-3.1-8B

that two dense paths beat sparse plus dense while two sparse paths do not improve on one dense term; both outcomes are consistent with the two-rung lemma (Appendix A.2) and the block-32 scale of the sparse instruction (Section 3.1). SDQ (Jeong et al., 2024), MX+ (Lee et al., 2025), HiFloat4's Rollout-ResQ (Mak et al., 2026) and SPARQLe (Parvathy et al., 2026) also stop at two terms; none runs more than two activation terms under block-scaled FP4 on current GPUs. R-Sparse (Zhang et al., 2025), TEAL (Liu et al., 2025b) and Alanova et al. (2026) run or emulate activation sparsity as a single term; we run a ladder of such terms on a production GPU and, as TEAL does, compare it with a matched dense build.

Transforms and formats. Our fold is reverse smoothing, the α=0 setting that SmoothQuant (Xiao et al., 2023) rejects and AMD's Quark documents for weight-only quantization (AMD, 2026); the transforms that migrate difficulty into an absorber or learn it from data, and the rotation methods, are compared in Appendix B. RaZeR (Chen et al., 2025b) changes the FP4 element format and ARCQuant appends quantized residual channels; we compare against both in Section 5.

Kernels. "One weight load, N MMAs, one kernel" has precedent as an idea (EmuGEMM (Lu et al., 2026a), MSD (Zheng et al., 2026), cuBLAS BF16x9), as does stacking operands along M (ABQ-LLM (Zeng et al., 2024), PrecisionBatching (Lam et al., 2020), ReSET (Lee et al., 2026)); ours is the bit-exact, single-accumulator, $N \geq 3$ structured-sparse, block-scaled instance on Blackwell tensor cores; the nearest published kernel is MX+ (Lee et al., 2025), whose second `mma.sp` reuses one weight load at N=2, while QuTLASS (Egiazarian et al., 2026) and APEX4 (Guo et al., 2026) compute one operand per load and SlideSparse (Shao et al., 2026) maps milder weight-sparsity patterns onto the 2:4 instruction.

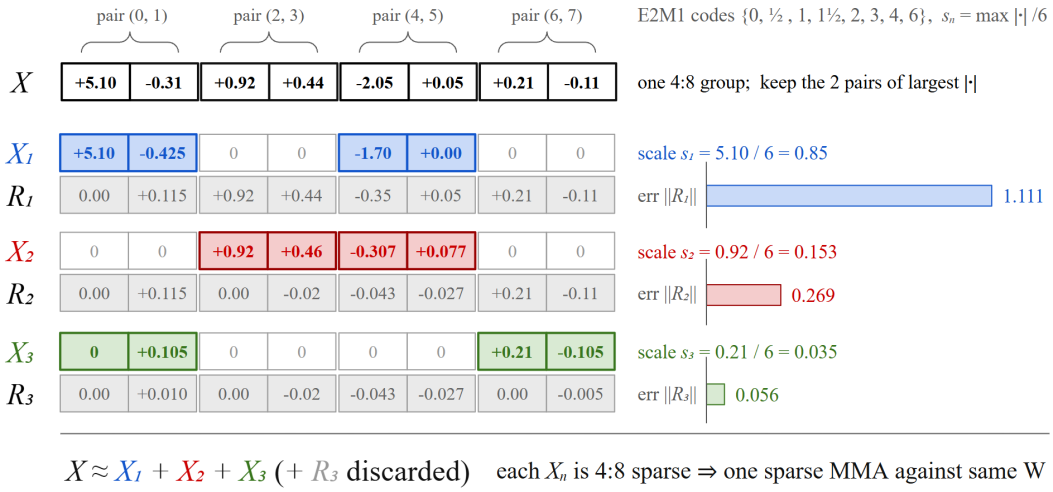


Figure 2: The greedy pair-granular peel on one group of eight activations: each rung keeps the two of four adjacent pairs with the largest magnitude, quantizes them to NVFP4 under its own scale, and hands the residual to the next rung (residual norm $1.11 \rightarrow 0.27 \rightarrow 0.056$; scales shown without E4M3 rounding). First pair: $s_1 = 5.10/6 = 0.85$; $-0.31/0.85 = -0.365$ rounds to the E2M1 code -0.5, so $-0.5 \times 0.85 = -0.425$ leaves $-0.31 + 0.425 = +0.115$ in the residual.

3 The Ladder, the Fold, and the Self-Probe

AWSLQ keeps W4A4 accurate at decode with no training and no external calibration data. It replaces the single dense quantization of each activation tile with a ladder of N structured-sparse operands that share one weight load (Section 3.1); the ladder converges to the weight-only reference by four rungs (Section 3.2); a reverse fold then lowers that reference, which the ladder can afford because its rungs absorb the outliers the fold creates (Section 3.3); and a preparation-time probe on self-generated text selects the fold per model (Section 3.4).

3.1 From one peel to an N-operand ladder

Let $X \in \mathbb{R}^{M\times K}$ be the activation tile entering a linear layer with weight $W \in \mathbb{R}^{K\times D}$, at decode batch M. Activation quantization $Q(\cdot)$ maps each 32-element block to E2M1 values under an E4M3 block scale; W4A4 denotes the operand precision of each multiplication, the activation being represented by several FP4 terms, and $\widehat{W}$ is the dequantized FP4 weight. The block is 32 because the sparse MMA carries four scale factors per 128 logical elements of K and has no eight-scale variant, so each scale spans 32 elements where dense NVFP4 covers 16 (Appendix D.6). The FP4 sparse tensor core consumes sparsity at *pair* granularity: every group of eight consecutive elements comprises four adjacent pairs, of which two may be non-zero.

Rung 1 keeps, in every group, the two pairs with the largest magnitude, the larger absolute value of a pair's two elements, and quantizes them (Figure 2); rung n repeats the peel on the previous residual ($\tilde{A}_n$ is A_n dequantized):

$$R_0 = X, \qquad A_n = Q\big(\mathrm{top}^{\mathrm{pair}}_{2:4}(R_{n-1})\big), \qquad R_n = R_{n-1} - \tilde{A}_n, \qquad Y = \sum_{n=1}^{N} \tilde{A}_n \widehat{W}. \tag{1}$$

Every A_n is 4:8-sparse at element level (2:4 over pairs), exactly the pattern the sparse FP4 MMA consumes with a 2-bit index per kept pair, so the output is N sparse MMAs against *one* weight tile. Selection is greedy magnitude, per token, online, and stateless: no calibration, no learned parameters, nothing stored between requests. The rungs add on-chip work but almost no memory traffic: the weights stream once, and at decode batch the ladder's activation payload is a couple of percent of the weight bytes (Appendix D.6). We write SLN (SL3, SL4, …) for the sparse ladder with N rungs; Section 4 computes Equation 1 with a single weight load.

3.2 Saturation: the ladder reaches the weight-only floor

Depth is the ladder's one parameter, and it needs no tuning: as the ladder deepens from $N{=}2$ to $N{=}8$ it converges, on every one of eleven models, to the W4A16 floor, the perplexity of the same model with the same quantized weights and transform and the activations left in 16-bit (Figure 8; Table 4 in Appendix A.1). At $N{=}2$, where the published FP4 evaluations stopped, the ladder sits 0.08–0.35 PPL above the floor on ten of the eleven models and 0.6 on Gemma 4; the third rung captures 84–96% of that gap, the fourth brings every model to within $+0.02$ PPL of the floor, and deeper rungs change perplexity by less than 0.005 on every model but Gemma 4. The same trend holds on a GPTQ weight track and on generative and zero-shot accuracy (Appendix A.4, Section 5.2); at four rungs the loss that remains is the weights' own, which is what the fold addresses (Section 3.3). Saturation also settles two design choices: a further rung beats the best two-operand selection rule we found by search, and adapting the depth per token, layer or expert leaves little headroom for accuracy (Appendices A.6 and A.7).

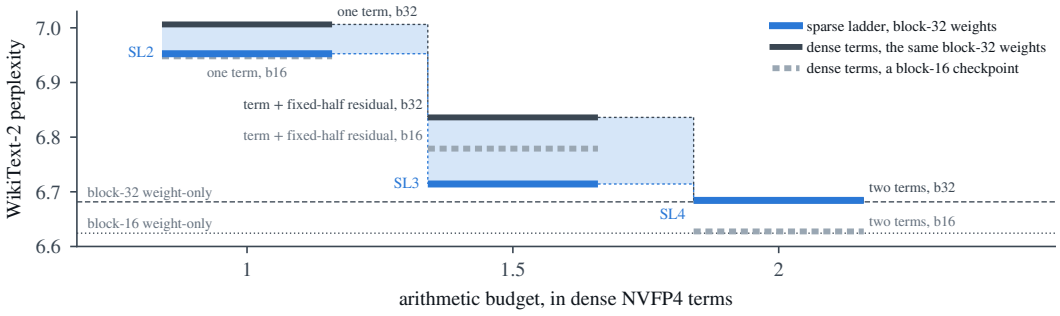
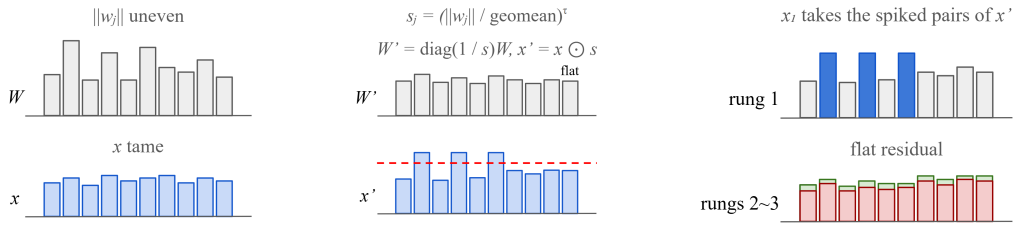


Figure 3: Accuracy against arithmetic budget on Llama-3.1-8B (WikiText-2; a rung is half a dense term). Blue: the ladder, block-32 weights; dark grey: dense terms on the same weight; light grey: a block-16 checkpoint, a scale the sparse instruction cannot use. The band: where the ladder lies below the same-weight dense frontier. Eight models: Appendix A.3; kernel costs: Appendix D.7.

What the rung buys. Two dense residual terms reach the same floor and small-batch latency as four rungs; sparse rungs add an intermediate arithmetic budget, three rungs at one and a half dense

terms with the residual pairs chosen per token. Two rungs contract a block's error by the factor of one dense term (Appendix A.2), so the floor is set by how many terms share the weight load, not by their structure: a base term and a residual (Yvinec et al., 2023; Rahman et al., 2026) tie SL4 within 0.008 PPL at block-32 weights (Table 4). A rung performs half the multiply-accumulates of a dense term (Figure 3), and each rung refines the half of a token's channels that carries the residual, a choice the prepare kernel makes per token and encodes in the instruction's 2:4 metadata. A dense configuration of that budget refines a fixed half of the channels, or every channel on alternate layers; the ladder matches or beats the tested ones on six of the eight models, by up to 0.07 PPL, and trails on the two Qwen3 models whose block-16 weight advantage is largest (Appendix A.3).



(a) Before the fold: the row norms of W vary widely while the activations that multiply them are comparatively flat.

(b) Reverse fold: row norms divided by s_j flatten; the activation channels, multiplied by s_j, exceed one dense block scale (dashed).

(c) With the ladder: the first rung captures the largest of those pairs and leaves a small residual for the remaining rungs.

Figure 4: Reverse smoothing and why the ladder makes it viable. The fold is applied to up, down, o, and v.

3.3 The selective reverse fold

Since the ladder already sits on the weight-only floor, further gains must lower the floor itself (Figure 4). For each input channel j of a projection, with $w_j = W_{j,:}$ its row of W, define

$$s_j = \left(\|w_j\|_2 \,/\, \text{geomean}_{j'} \|w_{j'}\|_2 \right)^{\tau}, \qquad W' = \text{diag}(1/s)\, W, \qquad x' = x \odot s, \tag{2}$$

so that $x'W' = xW$ exactly. The checkpoint stores W' and the ladder's prepare kernel applies the prescale, so a folded projection costs one elementwise multiply in its own prepare pass, and the projections that share its normalization are untouched; the identity holds at full precision. This is SmoothQuant's α=0 corner with the channel norm in place of the maximum, the end of the dial SmoothQuant advises against because it pushes magnitude onto the activations, which dense W4A4 quantizes poorly; the ladder absorbs it, because its first rung takes the largest of the pairs the fold enlarges and leaves a residual of small dynamic range (Appendix B.1). This is the *pairing* of the fold with a multi-term ladder: the fold moves the weight-only floor, and the ladder follows the moved floor to within 0.003 PPL, whereas a single dense NVFP4 term keeps a gap of 0.19–0.48 PPL to it (Appendix B); two dense terms follow it as well, within 0.002 PPL of four rungs on Llama-3.1-8B and Qwen3-8B. We fold up, down, o and v and leave gate, q and k unfolded, a rule set by where the fold moves the quantization error (Appendix B).

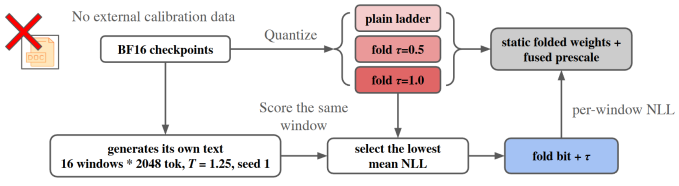


Figure 5: The preparation-time self-probe. The BF16 checkpoint generates its own probe text; the three candidate configurations are quantized and score it, and the one with the lowest mean NLL is kept. Serving carries no per-request cost.

3.4 The self-probe: choosing the fold from self-generated text

Whether the fold helps, and at what strength τ, is a property of the checkpoint: it helps ten of the thirteen models of Table 1, hurts both Llama-2 models, and is neutral on Qwen2.5-32B. Instead of an external calibration corpus, we select the fold using text generated by the model itself (Figure 5): at preparation time the BF16 model generates 16 windows of free text, self-generated data in the manner of LLM-QAT (Liu et al., 2023); the three candidates, plain ladder, fold τ=0.5 and fold τ=1.0, score those windows, and the one with the lowest mean per-window NLL is kept (protocol in Appendix C.4). Against full-evaluation ground truth on 18 models the probe never chose a harmful fold, declined it on the two Llama-2 models where it hurts, and matched the oracle τ on six of the eight held-out models, capturing about three quarters of the available fold gain (Table 12, Appendix C.4).

4 One Weight Load, N Sparse Operands: the Kernel

Equation 1 is only useful if all N rungs can be paid for with a single weight stream; executed as N separate GEMMs, as SharQ's released path does at N=2, the ladder costs $N\times$ the vendor's latency. Our CUTLASS-based kernels target sm_120a, the RTX 5090's block-scaled FP4 tensor cores (tiling in Appendix D.1), and pay for the rungs in two ways (Figure 6).

Fused kernel and its shared-memory fill. The direct way to share one weight load is to fuse the rungs into one kernel: an N-templated mainloop loads one NVFP4 weight tile into shared memory and issues the N sparse MMAs into a single FP32 accumulator, bit-exact through N=4. Fusion alone gains $1.41\times$ at N=3 over three launches, but each operand's tile must be filled into shared memory before its MMA can issue, and at M=16 the fused ladder pays 8–64% per added operand across the five decode shapes (Appendix D.1).

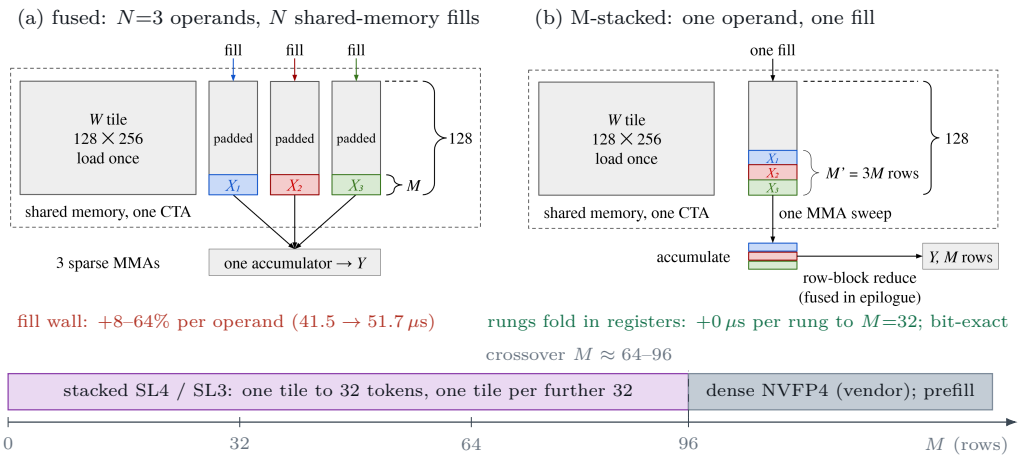


Figure 6: Paying for the rungs. (a) Fused: one shared-memory fill per operand before its MMA. (b) M-stacked: the rungs become extra rows of *one* operand against the same weight tile, with one MMA sweep, one accumulator, and a row-block reduce fused into the epilogue. The strip shows the kernel deployed at each M.

M-stacking. M-stacking places the rungs in the rows a decode tile leaves empty: the N rungs are placed as N blocks of extra rows of *one* A operand, so one accumulator tensor holds the rung-specific output rows, which are summed in registers before the output store; the metadata and scale layouts are consistent under row concatenation, and the result is bit-exact. For SL4, 32 tokens occupy 32×4=128 stacked rows, one tile; the rungs of a token occupy the rows one MMA thread owns, and tokens beyond 32 go to further 128-row tiles, each holding every rung of its own tokens. An extra rung costs no measurable time up to M=32, about one timer tick per shape; beyond M=64 the rows themselves, not the rungs, set the price. The stacking is indifferent to the operand's structure: a matched dense build, two terms on $2M$ rows, runs the Llama-3.1-8B block within 3% of SL4 at $M \leq 32$. At

Table 1: Perplexity against every published W4A4 number at RaZeR's protocol (token perplexity, full WikiText-2 test stream and 256 random C4 crops at sequence 2048, lm_head and embeddings BF16, round-to-nearest weights). Published rows use block-16 scales (SharQ's sparse operand: block 32) and are calibrated except SharQ and Four-over-Six; AWSLQ (SL4 with the per-model fold of Table 12, shaded) is zero-calibration at block 32; the W4A16 reference uses unfolded block-32 weights, whereas AWSLQ includes the selected fold. Values are rounded to two decimals (full precision in Appendix C.5). Bold marks the lowest W4A4 perplexity in a column; rows above the dashed rules are references. *Four-over-Six as re-implemented by RaZeR. †Below BF16 (Appendix A.5).

	Llama-2		Llama-3		Qwen2.5		Qwen3				Qwen3.5		Gemma 4
Method	7B	13B	3.1-8B	3.2-3B	7B	32B	4B	8B	14B	32B	4B	9B	12B
WikiText-2													
BF16	5.47	4.89	6.24	7.82	6.85	5.02	13.66	9.73	8.65	7.61	8.06	6.98	8.53
W4A16, unfolded	5.66	4.99	6.68	8.32	7.12	5.25	13.88	9.95	8.96	7.86	8.42	7.22	10.15
dense NVFP4	5.77	5.07	6.95	8.61	7.29	5.41	13.94	10.05	8.94	8.01	8.66	7.38	11.13
RaZeR	**5.66**	5.01	6.74	8.37	—	—	13.82	**9.83**	8.86	7.84	—	—	—
Four-over-Six*	5.72	5.06	6.88	8.53	—	—	13.88	10.00	**8.83**	7.91	—	—	—
SharQ	—	—	6.73	—	7.15	—	—	—	—	—	—	—	—
ARCQuant	—	—	6.87	—	7.28	5.38	—	—	—	—	—	—	—
AWSLQ	**5.66**	**4.99**	**6.68**	**8.27**	**7.12**	**5.25**	**13.55**†	9.90	8.86	**7.81**	**8.42**	**7.20**	**9.33**
C4													
BF16	6.97	6.47	8.96	10.44	10.44	8.95	16.64	13.30	12.02	10.78	11.55	10.34	11.32
W4A16, unfolded	7.18	6.58	9.58	11.21	10.73	9.09	16.98	13.61	12.35	11.02	11.95	10.64	12.74
dense NVFP4	7.29	6.66	9.94	11.65	10.91	9.21	17.26	13.78	12.41	11.16	12.26	10.82	13.56
RaZeR	7.19	6.59	9.63	11.27	—	—	17.11	**13.60**	12.26	11.02	—	—	—
Four-over-Six*	7.25	6.63	9.83	11.53	—	—	17.21	13.73	12.33	11.14	—	—	—
AWSLQ	**7.18**	**6.58**	**9.54**	**11.17**	**10.71**	**9.09**	**16.91**	13.62	**12.24**	**10.96**	**11.95**	**10.61**	**12.10**

compute-bound shapes the sparse instruction delivers twice the dense rate, but CUTLASS's sparse mainloop keeps the tensor pipe busy half the time against three quarters for the dense one, so three fused rungs run only 1.11–$1.15\times$ faster than a two-term dense pair (Appendix D.7).

Split-K, prepare, dispatch. A two-pass split-K pays off at long K and few output features, and the prepare kernel, one cooperative launch per activation tensor, quantizes online and leaves the next residual behind, bit-exact against the reference (Appendix D.6). Which kernel executes the ladder is chosen at run time, per layer shape and per M: the stacked kernel, SL4 or SL3, at every decode M; for a given N the stacked and fused kernels give identical outputs, so that choice leaves the numerical configuration unchanged. Two switches do change it: from $M \approx 96$ the vendor's single-term dense kernel is faster and the dispatch may return it, and at prefill the dispatch runs dense NVFP4 on the same weights, as the vendor stacks do (Mix-Quant's setting); the accuracy experiments apply the ladder at every decode step, the speed figures show it at every plotted M, and without the prefill switch the fused ladder would cost 1.62–$1.92\times$ the dense GEMM.

5 Accuracy

Protocol. Competitor numbers are the papers' published values at matched protocol (RaZeR (Chen et al., 2025b), ARCQuant (Meng et al., 2026a), SharQ (Meng et al., 2026b); calibration status in Table 1), and our BF16 numbers reproduce theirs to within 0.011 PPL on every shared model; for perplexity comparisons we treat $|\Delta\text{PPL}| < 0.02$ as a tie, about twice that agreement. Perplexity and log-likelihood evaluations apply the specified activation quantizer throughout; generation benchmarks use dense NVFP4 prefill followed by the specified decode configuration. Accuracy tasks use `lm-eval` 0.4.9.2 (Appendix C.4). Perplexity is simulated, the kernel verified bit-exact separately; saturation experiments hold the quantized weights and transform fixed, and task comparisons use the configuration each caption states.

5.1 Perplexity head-to-head

The comparisons in Table 1 differ in calibration requirements and scale-block size: the calibrated rows fit their transforms on data (RaZeR on the Pile, ARCQuant on WikiText-2), and every published row quantizes at block 16 (SharQ's sparse operand at 32), whereas AWSLQ uses no calibration and block-32 scales. AWSLQ still matches or beats every published method's number on eight of the

Table 2: All W4A4 arms use dense FP4 activation quantization at prefill; weight block sizes and folding follow each configuration, and the decode activation scheme varies as shown. The W4A16 rows leave the activations in 16-bit, at decode on the left and in both phases on the right; our arms are shaded. Left: GSM8K 8-shot CoT accuracy (%), pinned `lm-eval`, full test set, flexible extraction; bold marks the best quantized cell in each column; BF16 above the dashed rule is a reference. The W4A16 row keeps the 4-bit weights, the cheapest form of the retreat (QSpec's verifier); Mix-Quant and HMA-Serve decode with BF16 weights. [a]Qwen2.5-32B's self-probe declines the fold, so its ladder arms are the plain ladder. Right: MATH-500 under `math_verify` (mathematical equivalence); greedy, and mean±sd of three sampled runs (temperature 0.6, top-p 0.95); strict-extraction cells in Appendix C.3.

GSM8K arm	Qwen3-8B	Qwen3-14B	Qwen2.5-32B[a]
BF16	88.32	89.31	90.22
dense NVFP4	83.85	85.90	89.31
W4A16 decode	87.19	85.67	90.07
fold + SL3 decode	**87.87**	85.75	**90.52**
fold + SL4 decode	87.41	**86.66**	89.99

	Qwen3-8B		Qwen3-14B
MATH-500 arm	greedy	sampled	greedy
BF16	53.0	51.6 ± 0.5	52.8
dense NVFP4	49.6	48.3 ± 0.8	50.8
W4A16 both phases	50.4	$\mathbf{50.4} \pm 1.1$	51.2
fold + SL3	**52.0**	49.3 ± 0.4	54.2
fold + SL4	50.6	48.1 ± 1.0	**55.4**

Table 3: Long context. RULER on Qwen3-8B, all thirteen tasks, 100 samples per length, each cell the mean over 8K/16K/32K; every W4A4 arm runs the deployed dense NVFP4 prefill, the W4A16 reference leaves the activations in 16-bit in both phases at block-16 weights; bold marks the best quantized cell per column, BF16 above the dashed rule is a reference, VT = variable tracking, last column = suite mean. Needle retrieval is saturated; the arms differ on aggregation and question answering (Appendix C.4).

	NIAH single			NIAH multi-key			NIAH multi-		tracking	word extraction		QA		all
Arm	1	2	3	1	2	3	query	value	VT	frequent	common	SQuAD	HotpotQA	mean
BF16	1.000	1.000	1.000	1.000	1.000	0.993	1.000	0.991	0.999	0.944	0.818	0.723	0.577	0.927
dense NVFP4	**1.000**	**1.000**	**1.000**	0.997	0.980	0.973	0.995	0.987	**0.998**	0.926	0.779	0.673	0.523	0.910
W4A16 reference	**1.000**	**1.000**	**1.000**	**1.000**	**0.997**	0.963	**0.999**	0.993	**0.998**	**0.931**	**0.798**	0.663	0.570	**0.916**
fold + SL3	**1.000**	**1.000**	**1.000**	0.997	0.987	**0.987**	0.998	0.992	0.997	0.899	0.750	**0.691**	0.567	0.913
fold + SL4	**1.000**	**1.000**	**1.000**	**1.000**	0.990	0.973	0.997	**0.996**	0.997	0.910	0.728	0.686	**0.590**	0.913

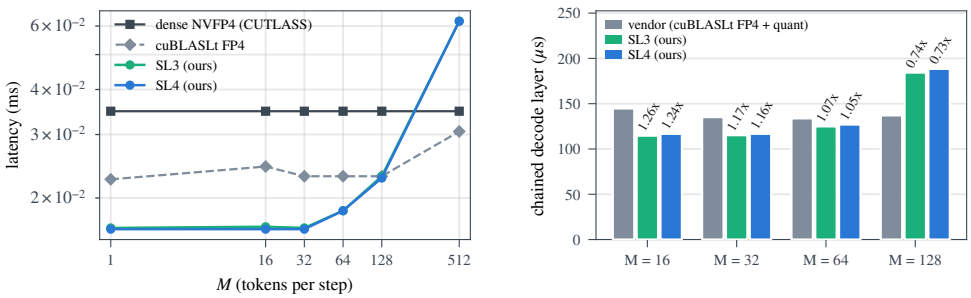
ten WikiText-2 columns and on all eight C4 columns (mean over RaZeR's eight models 8.21 vs. 8.27 on WikiText-2 and 11.03 vs. 11.08 on C4); the exceptions are Qwen3-8B (RaZeR 0.07 lower) and Qwen3-14B (Four-over-Six 0.03 lower), both ties on C4. Two dense paths, SharQ's ablation (6.63) and our own pair at block 16, sit below the Llama-3.1-8B cell, at the block-16 floor the sparse instruction cannot use (Appendix A.1). The fold accounts for much of the margin, lowering WikiText-2 perplexity by a further 0.04–0.84 over the plain ladder on six of the ten models whose probe selected it, and carrying over to C4 on all but Qwen3-8B. On a GPTQ (Frantar et al., 2023) weight track Llama-3.1-8B reaches 6.5441 at full W4A4, below RaZeR's weight-only GPTQ and AWQ (Appendix A.4); Qwen3-4B lands below its own BF16 perplexity, a property of the checkpoint (Appendix A.5).

5.2 Generative, zero-shot, and long-context accuracy

On GSM8K neither ladder arm trails 16-bit decode activations by more than 0.1 points on any of the three models, while uniform dense W4A4 loses 4.5 points on Qwen3-8B (Appendix C.1–C.4). On MATH-500 it clears the weight-only floor at greedy on both models, by 3.0–4.2 points on Qwen3-14B where it also beats BF16; under sampling the floor leads on Qwen3-8B. On zero-shot tasks neither arm falls more than half a point below the floor on any of the ten models, and the folded models sit at or above it on all but two. On MMLU 5-shot the ladder sits on the floor. On RULER both arms sit above uniform dense, within one standard error of the floor.

6 Speed

Decode speedup. We measure decode latency on an RTX 5090 per GEMM at five layer shapes (four Llama-3.1-8B layers and the 2560×9728 Qwen3-4B down projection) and on the chained



(a) Per-GEMM latency, D=2560, K=9728.

(b) Llama-3.1-8B block: four linear projections within one Transformer block, chained.

Figure 7: Cold-L2 decode latency on an RTX 5090; SL3/SL4 are the stacked kernel at every M (one tile per 32 tokens beyond the first).

Llama-3.1-8B block, against cuBLASLt's FP4 kernel with our own activation quantizer, CUTLASS's dense NVFP4 kernel, and Marlin (Frantar et al., 2025) for weight-only W4A16 (Figure 7; details in Appendix D). On the chained decode block, SL3 runs 1.26× faster than cuBLASLt-FP4 at M=16 (114.7 vs. 144.7 μs), 1.17× at M=32 and 1.07× at M=64; SL4, the operating point of the accuracy tables, is within 2% of SL3, the half step's point (Section 3.2). Per GEMM, the ladder beats cuBLASLt-FP4 at every layer shape, by 1.29–1.51× at M=16 and 1.07–1.27× at M=64. The speedup comes from streaming the weights closer to the memory bound, 42–78% of it at M=16 against the vendor's 28–57%, a wider margin than the 5.6% block-32 weights save in bytes (Appendix D.6). The ladder's preparation costs 18.0 μs per block against 12.6 μs for the vendor's quantizer, so the gain is in the GEMMs (Appendix D.3). The crossover is between M=64 and 96 (Figure 7): the stacked ladder is 1.08× slower than the vendor at M=96 and 1.34× at M=128, where the dispatch returns the vendor's kernel.

Against weight-only W4A16. Marlin, the fastest published W4A16 kernel (Table 18), moves the same weight bytes as FP4 at $M \leq 16$ and is 1.35–1.38× faster than the vendor's FP4 kernel there, with stacked SL4 within 3% of it. From $M \geq 32$ it pays for expanding every weight to BF16 for tensor cores at a quarter of the FP4 rate, and the stacked ladder is 1.88× faster at M=32 and 2.36× at M=64.

B200. The design ports to B200 (sm_100a) with bit-identical outputs: on the chained Llama-3.1-8B block, which is bound by launch overhead, stacked SL3 runs 1.08× faster than cuBLASLt-FP4 at M=16 and 1.03–1.05× at M=32, SL4 ties the vendor, and the lead is 1.02–1.06× when the vendor's quantizer is launched as ours is. The SL3 arm there reuses the previous token's per-tensor scale, as deployed stacks reuse a static one (Appendix D.4).

7 Limitations and Conclusion

The speed claim is scoped to $M \leq 64$, past one tile per 32 tokens the rows set the price, and we accelerate the linear layers only. At block-32 weights four rungs and a dense residual pair converge in accuracy and in weight-bound latency: the term count sets the floor, and only the dense instruction may use the block-16 weight scale that lowers it further. What the rung adds is the half step, three rungs at one and a half dense terms with the refined half chosen per token.

AWSLQ spends the tensor-core throughput that memory-bound decode leaves unused on several FP4 activation terms per weight load, here a ladder of structured-sparse NVFP4 rungs: it reaches the weight-only floor by four rungs or two dense terms, lowers that floor with a fold whose outliers the rungs absorb, beats cuBLASLt's FP4 GEMM at decode up to M=64, and past the knee at N=3 gains nothing measurable on GSM8K from 16-bit decode. Better sparse operand prefetching may turn the half step's budget into latency savings (Appendix D.7). The dense pair is compared in perplexity and kernel latency; task-level comparison is future work.

Reproducibility statement

We will release the kernel source (built on unmodified CUTLASS), the prepare kernel, the fold and self-probe scripts, the Marlin sm_120 port, and every result file, including the latency logs with their protocol (cold-L2 weight ring, iteration counts, spread across process replicas, cuBLASLt version and algorithm). Accuracy numbers come from `lm-eval` 0.4.9.2 with the three patches listed in Appendix C.4, which also gives the self-probe protocol.

Ethics statement

This work concerns the efficiency of LLM inference and introduces no new data, models, or capabilities. The evaluated checkpoints are public and used under their licenses.

AI use statement

We used generative AI tools for editing prose, writing analysis and plotting scripts, and drafting figure code. All AI-assisted code was verified against the bit-exact host references and result files described above. All AI-assisted text was reviewed by the authors, who take responsibility for the final content.

References

Shirin Alanova, Kristina Kazistova, Ekaterina Galaeva, Alina Kostromina, Vladimir Smirnov, Redko Dmitry, Alexey Dontsov, Maxim Zhelnin, Evgeny Burnaev, and Egor Shvetsov. Motivating next-gen accelerators with flexible N:M activation sparsity via benchmarking lightweight post-training sparsification approaches. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, 2026. arXiv:2509.22166.

AMD. SmoothQuant in AMD Quark: documentation. `https://quark.docs.amd.com/latest/pytorch/smoothquant.html`, 2026.

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 *NeurIPS*, 2024. arXiv:2404.00456.

Hao Mark Chen, Fuwen Tan, Alexandros Kouris, Royson Lee, Hongxiang Fan, and Stylianos I. Venieris. Progressive mixed-precision decoding for efficient LLM inference. In *ICLR*, 2025a. arXiv:2410.13461.

Yuzong Chen, Xilai Dai, Jake Hyun, Chi-Chih Chang, Wonsuk Jang, Yuheng Wu, Thierry Tambe, Jae-sun Seo, and Mohamed S. Abdelfattah. RaZeR: Pushing the limits of NVFP4 quantization with redundant zero remapping. *arXiv preprint arXiv:2501.04052*, 2025b. v3, January 2026.

Jack Cook, Junxian Guo, Guangxuan Xiao, Yujun Lin, Keith Wyss, Mahdi Nazemi, Asit Mishra, Carlo del Mundo, Tijmen Blankevoort, and Song Han. Four over six: More accurate NVFP4 quantization with adaptive block scaling. *arXiv preprint arXiv:2512.02010*, 2025. v5, May 2026.

DeepSeek-AI. Day 6: DeepSeek-V3/R1 inference system overview. `https://github.com/deepseek-ai/open-infra-index/blob/main/202502OpenSourceWeek/day_6_one_more_thing_deepseekV3R1_inference_system_overview.md`, 2025.

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. In *ICLR*, 2026. arXiv:2509.23202.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *ICLR*, 2023. arXiv:2210.17323.

Elias Frantar, Roberto L. Castro, Jiale Chen, Torsten Hoefler, and Dan Alistarh. MARLIN: Mixed-precision auto-regressive parallel inference on large language models. In *PPoPP*, 2025. arXiv:2408.11743.

Hong Guo, Nianhui Guo, Weixing Wang, Jona Otholt, Christoph Meinel, and Haojin Yang. APEX4: Efficient pure W4A4 LLM inference via intra-SM compute rebalancing. *arXiv preprint arXiv:2606.08761*, 2026.

Wenchen Han, Gingfung Matthew Yeung, Marco Barletta, William Toner, Amory Hoste, and Adam Barker. Lynx: Progressive speculative quantization for accelerating KV transfer in long-context inference. *arXiv preprint arXiv:2607.01831*, 2026.

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 *NeurIPS*, 2024. arXiv:2401.18079.

Yangjia Hu, Haodong Wang, Zicong Hong, Qianli Liu, Quanxin Shou, Jian Lin, Song Guo, Xiaowei Shen, Xiangjun Huang, Dian Wang, and Jian Yang. MosaicQuant: Inlier-outlier disaggregation for unified 4-bit LLM quantization. *arXiv preprint arXiv:2606.15652*, 2026.

Geonhwa Jeong, Po-An Tsai, Stephen W. Keckler, and Tushar Krishna. SDQ: Sparse decomposed quantization for LLM inference. *arXiv preprint arXiv:2406.13868*, 2024.

Geonhwa Jeong, Po-An Tsai, Abhimanyu Rajeshkumar Bambhaniya, Stephen W. Keckler, and Tushar Krishna. Enabling unstructured sparse acceleration on structured sparse accelerators. In *MLSys*, 2025. arXiv:2403.07953.

Maximilian Lam, Zachary Yedidia, Colby Banbury, and Vijay Janapa Reddi. Quantized neural network inference with precision batching. *arXiv preprint arXiv:2003.00822*, 2020.

Jungi Lee, Junyong Park, Soohyun Cha, Jaehoon Cho, and Jaewoong Sim. MX+: Pushing the limits of Microscaling formats for efficient large language model serving. In *MICRO*, 2025. arXiv:2510.14557.

Sihwa Lee, Janghwan Lee, Donghoon Yoo, Jae Gon Kim, Hanyul Ryu, Soojung Ryu, and Jungwook Choi. ReSET: Accurate latency-critical NVFP4 reasoning via step-aware temperature scaling. *arXiv preprint arXiv:2606.13233*, 2026.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. SVDQuant: Absorbing outliers by low-rank components for 4-bit diffusion models. In *ICLR*, 2025. arXiv:2411.05007.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *MLSys*, 2024. arXiv:2306.00978.

Fangxin Liu, Zongwu Wang, Jinhong Xia, Junping Zhao, Shouren Zhao, Jinjin Li, Jian Liu, Li Jiang, and Haibing Guan. FlexQuant: A flexible and efficient dynamic precision switching framework for LLM quantization. In *Findings of EMNLP*, 2025a. arXiv:2506.12024.

James Liu, Pragaash Ponnusamy, Tianle Cai, Han Guo, Yoon Kim, and Ben Athiwaratkun. Training-free activation sparsity in large language models. In *ICLR*, 2025b. arXiv:2408.14690.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. LLM-QAT: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*, 2023.

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 *ICLR*, 2025c. arXiv:2405.16406.

LMSYS Org. Deploying DeepSeek on GB200 NVL72 with PD and large scale EP (Part II): 3.8× prefill, 4.8× decode throughput. `https://www.lmsys.org/blog/2025-09-25-gb200-part-2/`, 2025.

Denghui Lu, Alexander Maeder, Mathieu Luisier, and Alexandros Nikolaos Ziogas. EmuGEMM: Fused Tensor Core kernels for precision emulation in matrix multiplication. *arXiv preprint arXiv:2606.25453*, 2026a.

Haiquan Lu, Zigeng Chen, Gongfan Fang, Xinyin Ma, and Xinchao Wang. Mix-Quant: Quantized prefilling, precise decoding for agentic LLMs. *arXiv preprint arXiv:2605.20315*, 2026b.

Hei Yi Mak, Shadan Golestan, Hoang Le, Mehran Taghian Jazi, Yunke Peng, Yaoyuan Wang, Yao Wang, Junsong Wang, Tianchi Hu, Fengchen He, Guipeng Hu, Tanzila Rahman, and Anandharaju Durai Raju. HiFloat4 format for end-to-end reinforcement learning post-training of large language models. *arXiv preprint arXiv:2607.26515*, 2026.

Haoqian Meng, Yilun Luo, Yafei Zhao, Wenyuan Liu, Peng Zhang, and Xindian Ma. ARCQuant: Boosting NVFP4 quantization with augmented residual channels for LLMs. In *ACL*, 2026a. arXiv:2601.07475.

Haoqian Meng, Yilun Luo, Yafei Zhao, Wenyuan Liu, Huaqing Zheng, Xindian Ma, and Peng Zhang. SharQ: Bridging activation sparsity and FP4 quantization for LLM inference. *arXiv preprint arXiv:2606.26587*, 2026b.

NVIDIA. CUTLASS documentation: Blackwell functionality. `https://github.com/NVIDIA/cutlass/blob/main/media/docs/cpp/blackwell_functionality.md`, 2026a.

NVIDIA. NVIDIA Nemotron 3 Ultra. `https://research.nvidia.com/labs/nemotron/Nemotron-3-Ultra/`, 2026b.

NVIDIA. 3 ways NVFP4 accelerates AI training and inference. `https://developer.nvidia.com/blog/3-ways-nvfp4-accelerates-ai-training-and-inference/`, 2026c.

NVIDIA. nvidia/DeepSeek-V4-Pro-NVFP4 model card. `https://huggingface.co/nvidia/DeepSeek-V4-Pro-NVFP4`, 2026d.

NVIDIA. NVIDIA TensorRT documentation: Data formats and tensors, structured sparsity. `https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/data-formats-tensors.html`, 2026e.

NVIDIA Model Optimizer. Best practices to choose the right quantization methods. `https://nvidia.github.io/Model-Optimizer/guides/_choosing_quant_methods.html`, 2026.

NVIDIA TensorRT-LLM. Pushing latency boundaries: Optimizing DeepSeek-R1 performance on NVIDIA B200 GPUs. `https://nvidia.github.io/TensorRT-LLM/blogs/tech_blog/blog1_Pushing_Latency_Boundaries_Optimizing_DeepSeek-R1_Performance_on_NVIDIA_B200_GPUs.html`, 2025.

Andrei Panferov, Maximilian Kleinegger, Sweta Priyadarshi, Tijmen Blankevoort, and Dan Alistarh. Disaggregated quantization: Specializing LLM prefill and decode. *arXiv preprint arXiv:2609.26333*, 2026.

Aradhana Mohan Parvathy, Soumendu Kumar Ghosh, Shamik Kundu, Arnab Raha, Souvik Kundu, Deepak A. Mathaikutty, and Anand Raghunathan. SPARQLe: Sub-precision activation representation for quantized LLM inference. *arXiv preprint arXiv:2606.00365*, 2026.

Tanzila Rahman, Mehran Taghian Jazi, Yunke Peng, Zhuang Ma, Anandharaju Durai Raju, Yao Wang, Xing Huang, Hei Yi Mak, Shadan Golestan, Hoang Le, Yonghan Dong, Wei Guo, and Yaoyuan Wang. Activation outliers matter: Robust recovery for quantized multimodal LLMs. *arXiv preprint arXiv:2608.26581*, 2026.

SceneWorks. Fused NVFP4 decode GEMV for one to eight rows (candle-llm pull request 1027). `https://github.com/SceneWorks/inference/pull/1027`, 2026.

secYOUre. nvfp4bench: instruction-level NVFP4 tensor-core microbenchmarks. `https://github.com/secYOUre/nvfp4bench`, 2026.

Hanyong Shao, Yingbo Hao, Ting Song, Yan Xia, Di Zhang, Shaohan Huang, Xun Wu, Songchen Xu, Le Xu, Li Dong, Zewen Chi, Yi Zou, and Furu Wei. SlideSparse: Fast and flexible (2N-2):2N structured sparsity. In *ICML*, 2026. arXiv:2603.05232.

Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. OmniQuant: Omnidirectionally calibrated quantization for large language models. In *ICLR*, 2024. arXiv:2308.13137.

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 *ICML*, 2025. arXiv:2410.09426.

vLLM Project. FP4 quantization with NVFP4. LLM Compressor documentation, `https://docs.vllm.ai/projects/llm-compressor/en/latest/examples/quantization_w4a4_fp4/`, 2026a.

vLLM Project. vLLM ModelOpt NVFP4 linear method (`vllm/model_executor/layers/quantization/modelopt.py`). `https://github.com/vllm-project/vllm/blob/main/vllm/model_executor/layers/quantization/modelopt.py`, 2026b.

Zhixiang Wei, Yun Wang, James Yen, Mingyuan Xia, and Zhengwei Qi. HBM is not all you need: Efficient disaggregated LLM serving across memory-heterogeneous accelerators. *arXiv preprint arXiv:2606.29986*, 2026.

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. arXiv:2211.10438.

Edouard Yvinec, Arnaud Dapogny, Matthieu Cord, and Kevin Bailly. REx: Data-free residual quantization error expansion. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Chao Zeng, Songwei Liu, Yusheng Xie, Hong Liu, Xiaojian Wang, Miao Wei, Shu Yang, Fangmin Chen, and Xing Mei. ABQ-LLM: Arbitrary-bit quantized inference acceleration for large language models. *arXiv preprint arXiv:2408.08554*, 2024.

Zhenyu Zhang, Zechun Liu, Yuandong Tian, Harshit Khaitan, Zhangyang Wang, and Steven Li. R-Sparse: Rank-aware activation sparsity for efficient LLM inference. In *ICLR*, 2025. arXiv:2504.19449.

Juntao Zhao, Wenhao Lu, Sheng Wang, Lingpeng Kong, and Chuan Wu. QSpec: Speculative decoding with complementary quantization schemes. In *EMNLP*, 2025a. arXiv:2410.11305.

Weibo Zhao, Yubin Shi, Xinyu Lyu, Wanchen Sui, Shen Li, and Yong Li. ASER: Activation smoothing and error reconstruction for large language model quantization. In *AAAI*, 2025b. arXiv:2411.07762.

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. In *MLSys*, 2024. arXiv:2310.19102.

Lingchao Zheng, Yuwei Fan, Jun Li, Chengqiu Hu, Qichen Liao, Junyi Fan, Rui Shi, and Fangzheng Miao. Multi-Scale Dequant: Eliminating dequantization bottleneck via activation decomposition for efficient LLM inference. *arXiv preprint arXiv:2605.13915*, 2026.

Dawei Zhu, Nan Yang, Liang Wang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. PoSE: Efficient context window extension of LLMs via positional skip-wise training. In *ICLR*, 2024. arXiv:2309.10400.

A Saturation and Ablations

A.1 Saturation

Table 4: WikiText-2 perplexity (full test stream, sequence 2048, round-to-nearest weights) as the ladder deepens. The W4A16 row keeps the same 4-bit weights with the activations left in 16-bit, the reference the ladder converges to; [R]the block-16 weight-only perplexity RaZeR publishes. By $N{=}4$ every model is within $+0.02$ of the floor and all but Gemma 4 within $+0.005$; deeper rungs change perplexity by less than 0.005 except on Gemma 4 (0.03). All quantized rows but [R] and the block-16 dense row use block-32 scales; the deployed depth is shaded. The last two rows are two dense NVFP4 residual terms on one weight load, the arithmetic budget of four rungs: they land on the ladder's floor at block 32 and, wherever block-16 weight scales are the better format, below it.

	Llama-2		Llama-3		Qwen2.5	Qwen3			Qwen3.5		Gemma 4
Arm	7B	13B	3.1-8B	3.2-3B	7B	4B	8B	14B	4B	9B	12B
BF16	5.4734	4.8854	6.2397	7.8165	6.8500	13.6590	9.7259	8.6474	8.0554	6.9823	8.5319
W4A16 floor	5.6560	4.9906	6.6816	8.3193	7.1181	13.8835	9.9462	8.9579	8.4219	7.2175	10.1529
W4A16 block 16[R]	5.63	4.98	6.63	8.24	—	13.83	9.92	8.80	—	—	—
$N{=}2$	5.7654	5.0736	6.9526	8.6471	7.2948	14.2327	10.0646	9.0692	8.6826	7.3564	10.7630
$N{=}3$	5.6694	5.0014	6.7143	8.3561	7.1404	13.9323	9.9655	8.9623	8.4501	7.2301	10.2206
$N{=}4$	5.6567	4.9913	6.6843	8.3219	7.1194	13.8880	9.9488	8.9507	8.4231	7.2182	10.1711
$N{=}6$	5.6565	4.9905	6.6826	8.3190	7.1187	13.8869	9.9468	8.9503	8.4206	7.2175	10.1375
$N{=}8$	5.6563	4.9907	6.6829	8.3201	7.1181	13.8889	9.9463	8.9514	8.4211	7.2172	10.1528
gap($N{=}4$)	$+0.0007$	$+0.0007$	$+0.0027$	$+0.0026$	$+0.0013$	$+0.0045$	$+0.0026$	-0.0072	$+0.0012$	$+0.0007$	$+0.0182$
two dense terms, block 32	—	—	6.6844	—	—	—	9.9471	—	—	—	10.1634
two dense terms, block 16	—	—	6.6275	—	—	—	9.9238	—	—	—	10.3208

The floor depends on the weight block: the sparse instruction requires block-32 weight scales, and RaZeR's block-16 weight-only numbers sit 0.01–0.16 lower on all seven models with a published value, most on Qwen3-14B, which is why uniform dense at block 16 sits below the block-32 floor there (Table 1). Two dense terms on one load make the same point on our own protocol: at block 32 they land within 0.008 PPL of the ladder on all three models, and at block 16 within 0.004 of the block-16 floor, as SharQ's two dense paths do (6.63); the gap between the two floors therefore comes from the weight block, which the sparse instruction fixes at 32. On Gemma 4 block-16 weights are the worse format for every arm (weight-only floor 10.3010 against 10.1529), and the dense pair at block 16 (10.3208) sits above the ladder there.

A.2 Two rungs against one dense term

The tie between four rungs and a dense pair in Table 4 is consistent with a worst-case bound: two greedy rungs contract a block's error at least as much as one dense term. Take one 32-element block with maximum m, and let its scale codes stay in UE4M3's normal range. A dense NVFP4 term quantizes the block to E2M1 codes under the scale $m/6$: the largest code is 6 and the widest step 2, so every element is represented to within $m/6$, and to within $17m/96$ once the scale itself is rounded to E4M3, whose worst case is a factor $17/16$. Two greedy rungs reach the same bound. Rung 1 keeps the two of four pairs with the largest absolute maximum in every group of eight, so the pair holding m is kept and its scale is again $m/6$; the kept elements leave residuals no larger than $17m/96$, and each dropped pair has an absolute maximum no larger than the kept ones. Rung 2 keeps the two pairs of largest residual maximum, represents them to within $17/96$ of that maximum, which is at most m, and drops the rest; a pair dropped by both rungs was outranked in rung 2 by at least one pair that rung 1 had kept, whose residual is at most $17m/96$. Every element therefore ends within $17m/96$ of its value: two rungs contract a block's maximum error by at least $96/17 \approx 5.65$, the factor of one dense term. The bound is tight; on Llama-3.1-8B activations the largest measured ratio is 0.1764 against 0.1771. At even depth the two forms are at parity in the worst case, and the last rows of Table 4 show the parity on average.

A.3 The half step

The half step is the one budget where the ladder's per-token choice of the refined half can show, so it is where the sparse structure is tested against dense configurations of the same cost. At an arithmetic budget of one and a half dense terms, SL3 at block-32 weights is compared with three of them,

each at block-16 weights (Table 5): one dense term plus a residual term on a fixed half of the input channels, either an arbitrary half or the block-aligned half of largest weight-column energy, and two dense terms on even layers with one on odd. The three dense configurations agree within 0.02 on six of the eight models and within 0.05 on all, so which fixed half carries the residual matters little; what separates the ladder from them is that its half is chosen per token. The two models where the dense configurations lead, Qwen3-4B and Qwen3-14B, are the two with the largest block-16 weight advantage (0.31 and 0.19 PPL between one dense term on block-16 and on block-32 weights, against 0.02–0.09 elsewhere). With each model's probe-chosen fold the ladder keeps its lead on Llama-3.1-8B (6.70 against 6.75–6.77) and loses it on Qwen3-8B (9.92 against 9.83–9.87), where the fold widens the block-16 weight advantage. On the block-32 weight the deployment stores, the same three dense configurations give 6.8361, 6.8389 and 6.8568 on Llama-3.1-8B, 0.12 PPL above SL3 (Figure 3).

Table 5: The half step: WikiText-2 perplexity (full test stream, no fold) at an arithmetic budget of one and a half dense terms. SL3 uses block-32 weights; the dense configurations use block-16 weights. Bold marks the lowest value in a column.

	Llama-2		Llama-3		Qwen2.5	Qwen3		
Configuration at 1.5 dense terms	7B	13B	3.1-8B	3.2-3B	7B	4B	8B	14B
SL3, half chosen per token	**5.6694**	**5.0014**	**6.7143**	**8.3561**	**7.1404**	13.9323	9.9655	8.9623
dense term + residual on an arbitrary half	5.6962	5.0213	6.7831	8.4188	7.1857	13.7425	9.9989	**8.8428**
dense term + residual on the largest-energy half	5.6983	5.0204	6.7790	8.4210	7.1940	**13.7118**	9.9783	8.8560
two dense terms on even layers, one on odd	5.6937	5.0176	6.7980	8.4180	7.1961	13.7451	**9.9537**	8.8583

A.4 Saturation on a GPTQ weight track

The sweep of Table 4 is repeated in Table 6 on seven of its eleven models with GPTQ-quantized weights (Frantar et al., 2023) (a comparator weight track, not part of AWSLQ). The ladder reaches each track's own floor. Small negative gaps reflect interaction between the weight and activation errors. GPTQ itself hurts Llama-3.2-3B by $+1.40$ PPL at the floor and by $+1.39$–1.51 at every depth $N \geq 3$, and the ladder converges to that worse floor as well.

Table 6: WikiText-2 perplexity (full test stream, sequence 2048) with GPTQ weights as the ladder deepens. The W4A16 row is the GPTQ weight-only floor: 4-bit weights with the activations left in 16-bit; the deployed depth is shaded. The $N{=}2$ row is a two-way split of the pairs rather than the recursive peel of Equation 1.

	Llama-2		Llama-3		Qwen2.5	Qwen3	
Arm	7B	13B	3.1-8B	3.2-3B	7B	4B	8B
BF16	5.4734	4.8854	6.2397	7.8165	6.8500	13.6590	9.7259
W4A16 (GPTQ)	5.6119	4.9590	6.5424	9.7170	7.0039	14.0662	9.9327
$N{=}2$	5.7936	5.0500	6.8578	11.1882	7.2376	14.2878	10.1067
$N{=}3$	5.6193	4.9698	6.5734	9.8653	7.0224	14.0940	9.9486
$N{=}4$	5.6105	4.9599	6.5441	9.7587	7.0042	14.0643	9.9344
$N{=}6$	5.6109	4.9589	6.5418	9.7384	7.0036	14.0620	9.9302
$N{=}8$	5.6123	4.9591	6.5422	9.7141	7.0033	14.0645	9.9340

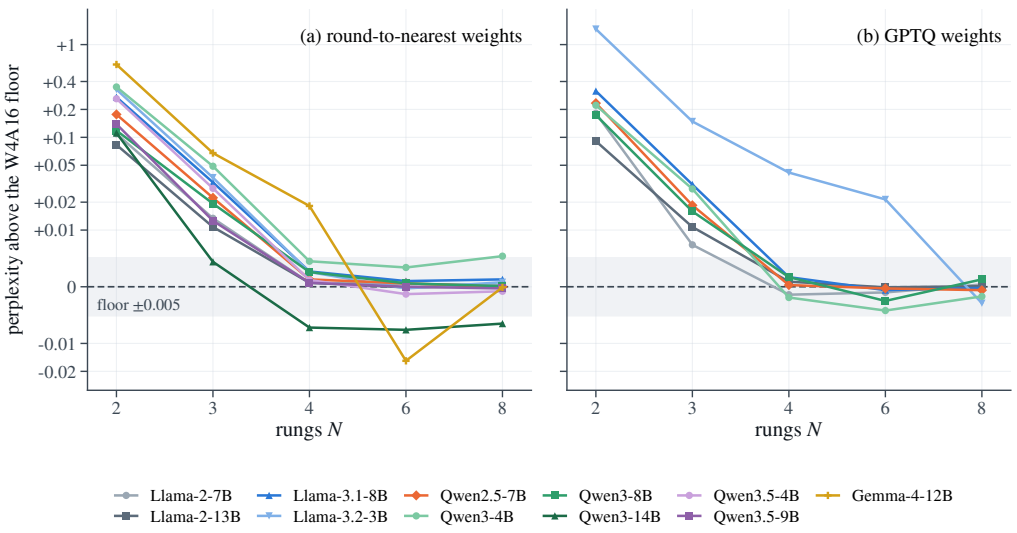


Figure 8: The ladder's gap to the weight-only floor as it deepens, on (a) round-to-nearest and (b) GPTQ weights (Tables 4 and 6; symmetric-log axis, the band marks ± 0.005). The knee is at $N{=}3$; beyond $N{=}4$ only Gemma 4 and, on the GPTQ track, Llama-3.2-3B still move by more than 0.005.

A.5 The sub-BF16 cell

Qwen3-4B's deployed cell (13.5468, Table 13) is below its BF16 perplexity (13.6590) on WikiText-2. We attribute the crossing to the checkpoint: its raw WikiText-2 perplexity improves under small perturbations, and RaZeR's own W4A4 result (13.82) likewise sits below their weight-only result (13.83). The fold is function-preserving at full precision (folded BF16 13.6623 vs. unfolded 13.6604 in the same run), and on C4 its gain transfers (-0.07) without crossing BF16. The relevant quantity is the -0.34 PPL gain over the unfolded ladder (13.8880, Table 4).

A.6 Smarter peeling versus one more rung

We compared greedy magnitude selection with smarter peeling: selection criteria for the two-operand split found by an offline search and tuned per model on Llama-2-7B, Llama-3.1-8B, Qwen2.5-7B, and Qwen3-8B. None is as effective as one more rung. On all four models the greedy SL3 ladder is more accurate than the searched two-operand split, and greedy SL4 is more accurate than a searched three-operand ladder. AWSLQ therefore uses greedy magnitude selection.

A.7 Adaptivity ablations

Depth could in principle vary per token (from each row's activation energy), per layer, or per expert, but each of these can only pay off where the uniform ladder leaves headroom to the floor, and at gap($N{=}4 \to$floor) ≤ 0.018 PPL on every model there is little. The per-expert axis fails for a second reason: a weight-statistics survey of all 48 layers $\times$ 128 experts found the experts quantization-homogeneous. Per-expert quantization difficulty (the block absmax-to-RMS ratio of the expert's weights, averaged over its gate, up, and down projections) has a within-layer coefficient of variation of 0.008 at the median and 0.021 at most, a max$-$min spread of 5% at the median, and a Spearman correlation of $+0.01$ with the norm of the expert's router row. Per-expert depth would therefore change little, and the ladder is uniform.

B The Fold against Calibrated Transforms

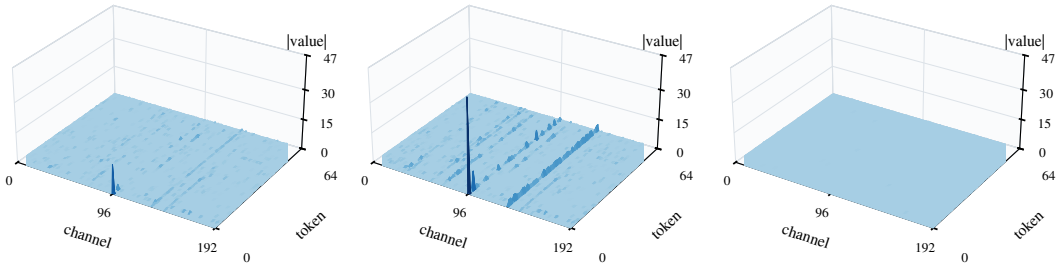
Which projections to fold. Dividing row j by s_j moves the relative quantization error onto the channels that used to set each block's scale, which is harmless where those channels carry little activation energy (the inputs of up, down, o and v on every model we examined) and harmful where

they carry the most (k on every model, gate and q on some); we fold the former and leave the latter unfolded, a rule fixed once for every model.

Calibrated and learned transforms. SVDQuant (Li et al., 2025), MosaicQuant (Hu et al., 2026) and ASER (Zhao et al., 2025b) migrate quantization difficulty into an absorber, AWQ (Lin et al., 2024) rescales weight channels by activation statistics for weight-only quantization, and OmniQuant (Shao et al., 2024) learns the transform from calibration data; rotation-based INT4 W4A4 (QuaRot (Ashkboos et al., 2024), SpinQuant (Liu et al., 2025c), FlatQuant (Sun et al., 2025)) uses per-channel and per-token scales, a different format from block-scaled NVFP4, and in ARCQuant's own NVFP4+QuaRot ablation the rotation does no better than round-to-nearest (Appendix C.5). A calibrated SmoothQuant dial swept over α does not subsume the fold (Table 7). We apply SmoothQuant's channel-max family, whose scale for input channel j is $\max|x_j|^{\alpha}/\max|W_j|^{1-\alpha}$, to the four folded projections of Qwen3-8B and Llama-3.2-3B; every $\alpha > 0$ needs calibration data for the activation maxima. On the dense arm, as SmoothQuant's guidance predicts, $\alpha{=}0$ is the worst of the family (11.64 vs. 8.79 at $\alpha{=}0.5$ on Llama-3.2-3B), and on Qwen3-8B no α beats unfolded dense. The ladder absorbs the activation-side damage: under the same $\alpha{=}0$ transform it lands at 10.19 on Llama-3.2-3B, within 0.06 of that transform's W4A16 floor (10.13), while uniform dense sits 1.51 above it. No calibrated α beats the zero-calibration fold within the tie rule on either model (Qwen3-8B 9.9041 vs. the family's best 9.8903 at $\alpha{=}0.75$, a tie; Llama-3.2-3B 8.2734 vs. 8.2794 at $\alpha{=}0.5$). The literal channel-max $\alpha{=}0$ setting is not the family's best under the ladder (10.19 against 8.28 at $\alpha{=}0.5$ on Llama-3.2-3B). The channel-norm rows are where the pairing of Section 3.3 shows: under the fold the ladder sits on the weight-only floor (9.9041 vs. 9.9061 on Qwen3-8B, 8.2734 vs. 8.2707 on Llama-3.2-3B) while uniform dense keeps a gap of 0.19 and 0.48 to it. The stronger counterfactual is a *learned* equivalent transform, OmniQuant's LET (Shao et al., 2024). We port its trainer to Qwen3, calibrate at its default 128 sequences of 2048 tokens from the WikiText-2 training split, and apply the learned scales to the checkpoint as an exact reparameterization. It regresses every arm under NVFP4 weights (dense 11.73, SL4 10.14, W4A16 10.13 against unfolded 10.12/9.95/9.95 and our fold's 10.10/9.90/9.91).

B.1 What the fold does to the activations

Under SmoothQuant's channel-max scale at $\alpha{=}0$ the largest activation of a Qwen3-8B up-projection grows from 14.6 to 47.0 and whole channels rise with it; our channel-norm fold moves magnitude the same way. The ladder absorbs it: its first rung takes the largest-magnitude pairs, so the channels the fold enlarges are the ones it quantizes first, and the residual the remaining rungs see is flat (largest value at most 1.1). Figure 9 shows the input of one folded projection as it arrives, after the fold at SmoothQuant's own $\alpha{=}0$ setting, and after the ladder's first rung has taken its share.



(a) The activation as it arrives at a Qwen3-8B up-projection (layer 15, one WikiText-2 window; its first 64 tokens and the 192 channels around the hottest one): one channel spikes to 14.6 at the first token, and no other channel passes 1.5.

(b) After the fold at SmoothQuant's own α=0 setting (channel-max scale): the spike grows to 47.0 and whole channels are lifted, one of them above 1 at nearly every token. One dense NVFP4 block scale cannot represent both the spike and its neighbours.

(c) The residual after rung 1 has taken the two largest of four adjacent pairs per group of eight: flat by construction, largest value 0.6 here and 1.1 over all 2048 tokens.

Figure 9: The fold direction and the ladder's first rung. All three panels share the z axis; color follows height on a log scale, and everything below 0.5 takes the base color. Over all 2048 tokens the kurtosis of the per-channel maxima is $700 \rightarrow 1864 \rightarrow 7.5$. Our channel-norm fold moves magnitude the same way but toward the cold channels (largest value 9.2 on this layer) and is absorbed identically (residual 1.0).

B.2 The SmoothQuant α family and OmniQuant's LET

Table 7: The SmoothQuant α family (channel-max scales; every $\alpha > 0$ needs calibration data) on the four folded projections versus our zero-calibration fold, WikiText-2 perplexity over the full test stream. Every cell uses block-32 scales, so the dense column is the block-32 dense arm, not the block-16 dense baseline of Table 1. "dense NVFP4" = uniform W4A4; "SL4" = the ladder; "W4A16" = 4-bit weights with the activations left in 16-bit. Rows are transforms, columns the quantizer applied after them; our fold is shaded. Bold marks the lowest perplexity in each column, and any cell within the 0.02 tie rule of it. LET was trained on Qwen3-8B only.

	Qwen3-8B			Llama-3.2-3B		
Transform	dense NVFP4	SL4	W4A16	dense NVFP4	SL4	W4A16
none (unfolded)	10.1224	9.9488	9.9462	8.8388	8.3219	8.3193
SmoothQuant α=0	10.4808	10.1902	10.1430	11.6402	10.1852	10.1331
SmoothQuant α=0.25	10.1830	9.9269	9.9275	8.9341	8.3656	8.3633
SmoothQuant α=0.5	10.3260	10.0495	10.0473	8.7852	**8.2794**	**8.2754**
SmoothQuant α=0.75	10.2202	**9.8903**	**9.8881**	8.9339	8.3485	8.3459
SmoothQuant α=1.0	10.4783	10.0307	10.0219	9.4898	8.6324	8.6215
OmniQuant LET (learned)	11.7315	10.1365	10.1327	—	—	—
our fold (zero-calibration)	**10.1000**	**9.9041**	**9.9061**	**8.7518**	**8.2734**	**8.2707**

C Evaluation Details

C.1 Zero-shot suite

Saturation holds on accuracy (Table 8; eight tasks, uniform arms since log-likelihood scoring has no decode phase): on all ten models neither ladder arm falls more than half a point below the W4A16 reference, both recover $+0.3$ to $+1.9$ points over uniform dense, and the deployed configuration sits at or above the reference on average (mean $+0.12$) and on all but two of the models where the probe selects the fold. Under MMLU 5-shot (Table 9) the ladder sits on the floor (mean -0.03 at SL4, -0.07 at SL3).

Table 8: Zero-shot average over eight tasks (PIQA, ARC-e, ARC-c, HellaSwag, WinoGrande, LAMBADA, BoolQ, MMLU 0-shot; `acc_norm` where defined), pinned `lm-eval`. The W4A16 reference is 4-bit weights with the activations left in 16-bit. The two ladder rows are the fold (probe-chosen τ) plus the uniform ladder; on Llama-2-7B, Llama-2-13B and Qwen2.5-32B, where the probe declines the fold, they are the plain ladder. Bold marks the best quantized cell in each column; BF16 above the dashed rule is a reference.

	Llama-2		Llama-3		Qwen2.5		Qwen3			
Arm	7B	13B	3.1-8B	3.2-3B	7B	32B	4B	8B	14B	32B
BF16	67.0	71.2	74.2	67.2	73.5	77.5	69.3	72.7	76.1	77.0
dense NVFP4	65.3	70.3	71.5	64.5	72.1	77.2	65.3	71.1	74.3	76.3
W4A16 reference	66.1	70.7	**73.0**	65.5	73.2	77.9	**67.6**	71.6	75.0	**77.0**
fold + SL3	66.1	70.6	72.8	66.0	73.1	**78.0**	67.2	**71.9**	**75.5**	76.8
fold + SL4	**66.2**	**70.8**	**73.0**	**66.2**	**73.4**	77.9	67.2	**71.9**	75.4	76.8

Table 9: MMLU 5-shot (%), pinned `lm-eval`, `num_fewshot=5`. SL4 sits on the W4A16 reference (mean SL4−reference -0.03). Uniform dense trails the reference by 0.4–2.0 on every model. Bold marks the best quantized cell in each column; BF16 above the dashed rule is a reference.

	Llama-2		Llama-3		Qwen2.5	Qwen3			
Arm	7B	13B	3.1-8B	3.2-3B	7B	4B	8B	14B	mean
BF16	45.8	55.1	65.4	56.6	74.3	70.1	74.9	78.8	65.1
dense NVFP4	42.7	53.9	62.6	53.2	72.4	66.8	72.3	77.8	62.7
W4A16 reference	**43.9**	**54.3**	**63.8**	53.6	73.1	**68.8**	**73.9**	**78.2**	**63.7**
fold + SL3	43.7	**54.3**	63.7	54.5	**73.2**	68.2	73.7	77.7	63.6
fold + SL4	43.8	**54.3**	**63.8**	**54.9**	73.1	68.1	73.6	77.8	**63.7**

C.2 GSM8K breadth and Instruct checkpoints

Uniform dense W4A4 loses 4.5 points of GSM8K accuracy on Qwen3-8B; the ladder at decode recovers as much of that as the retreat does, and on all three models with a retreat arm it trails by at most a tenth of a point. Across a seven-model $n{=}250$ grid the SL4 ladder scores above uniform dense on five models, and on Qwen3-14B the probe-chosen fold turns a dense lead into a ladder lead (Table 10). LongBench-V2 gives low absolute accuracy and does not favor the ladder: every arm sits near chance (random $= 25\%$), with BF16 at 31.5 ± 0.5, uniform dense at 33.9 ± 0.3 and the ladder at 31.6 ± 0.2; it is the one benchmark here whose ladder arm was run behind W4A16 prefill.

In the deployed configuration (probe-chosen fold plus SL4 decode), full-test-set accuracy ($n{=}1319$) through pinned lm-eval is Qwen3-4B 81.88, Qwen2.5-7B 80.36, Qwen3-14B 86.66, and Llama-3.2-3B 22.97 (a base model that is weak at GSM8K in BF16). The exceptions to the $n{=}250$ ordering are Llama-3.1-8B, where the two ladder arms straddle uniform dense and both gaps sit inside the ± 3.2-point standard error at $n{=}250$, and Qwen3-14B, where the probe-chosen fold restores the ordering (folded SL4 90.4, folded dense flat at 89.2).

Table 10: GSM8K 8-shot CoT accuracy (%) on the n=250 breadth grid (the first 250 test questions, scored by our own evaluation script; these questions are easier than the full set, so cells are comparable only within this table). Prefill is dense NVFP4 in every W4A4 arm and the decode arm varies, as in Table 2, while the W4A16 reference leaves the activations in 16-bit in both phases; bold marks the best quantized cell in each column, BF16 above the dashed rule is a reference, and our arms are shaded. [f] Qwen3-14B with its probe-chosen fold (τ=1.0) applied to every one of its quantized arms. Llama-3.2-3B and the two Llama-2 models are weak at GSM8K in BF16, and the standard error at n=250 runs from about 1.9 to 3.2 points. Against uniform dense, SL3 gains +1.2 to +3.2 points on six of the seven models and SL4 +1.6 to +4.4 on five. The SL4–SL3 differences are within noise: on the full test set the gap on Qwen3-8B is −0.46 points (87.41 vs. 87.87, Table 2) against a 1.28-point standard error of the difference.

	Llama-2		Llama-3		Qwen2.5	Qwen3		
Arm	7B	13B	3.1-8B	3.2-3B	7B	4B	14B	14B[f]
BF16	17.2	30.0	59.6	28.4	87.6	88.4	91.6	91.6
dense NVFP4	12.8	24.8	46.8	20.0	81.2	83.2	**89.6**	89.2
W4A16 reference	14.4	**26.4**	**48.4**	**24.4**	84.0	84.8	87.6	88.8
SL3 decode	14.0	**26.4**	**48.4**	22.4	84.4	84.4	87.6	88.4
SL4 decode	**15.6**	**26.4**	44.8	23.6	**85.2**	**87.6**	86.4	**90.4**

Instruct checkpoints run under the chat-template task `gsm8k_cot_llama`, BF16 against the deployed configuration: Llama-3.1-8B-Instruct 85.52 / 81.05 and Llama-3.2-3B-Instruct 77.41 / 74.00; RaZeR reports 82.64 / 75.74 for its W4A4 through the same lm-eval release with its own task configuration (its FP16 anchors are 85.52 / 78.09).

C.3 MATH-500 under strict extraction

Table 2 scores MATH-500 with `math_verify`, which accepts any answer mathematically equivalent to the reference. The task also logs `exact_match`, a strict extraction rule, and the two disagree by 6.8 points on BF16 alone: the strict rule reads answer formatting as well as correctness before quantization enters (Table 11). The gap is 6.2–8.2 points on every arm except Qwen3-8B's two folded arms, where it is 11.8 and 12.0. The W4A16 reference on that checkpoint sits at 6.2, so the larger gap goes with the fold on this model, which suggests this configuration changes how Qwen3-8B formats its final answer; the equivalence rule, which ignores format, places those arms above uniform dense.

Table 11: MATH-500, greedy, both scorers. "gap" = equivalence − strict; bold marks the best quantized cell in each score column; BF16 above the dashed rule is a reference.

	Qwen3-8B			Qwen3-14B		
Arm	strict	equiv.	gap	strict	equiv.	gap
BF16	46.2	53.0	6.8	45.8	52.8	7.0
dense NVFP4	41.4	49.6	8.2	43.0	50.8	7.8
W4A16 reference	**44.2**	50.4	6.2	44.6	51.2	6.6
fold + SL3	40.2	**52.0**	11.8	47.6	54.2	6.6
fold + SL4	38.6	50.6	12.0	**48.2**	**55.4**	7.2

C.4 Protocols

Table 12: The per-model fold, chosen by the preparation-time self-probe (Figure 5).

Model	Configuration
Qwen3-4B, 8B, 14B, 32B; Qwen3.5-9B	fold, τ=1.0
Qwen2.5-7B, Llama-3.1-8B, Llama-3.2-3B, Qwen3.5-4B, Gemma 4 12B	fold, τ=0.5
Llama-2-7B, Llama-2-13B, Qwen2.5-32B	plain ladder (probe declines)

Self-probe. 16 windows $\times$ 2048 tokens generated by the BF16 checkpoint at temperature 1.25, top-p 1.0, seed 1; candidates $\{\text{plain}, \text{fold } \tau{=}0.5, \text{fold } \tau{=}1.0\}$ under SL4; per-window NLL; the model is folded if either fold arm's mean NLL is below the plain arm's, at the arm with the lower mean. No significance gate is applied: on the held-out models no chosen fold is worse than the plain ladder by more than 0.001 PPL, while a declined fold can forgo up to 1.67. The eight held-out models were probed before their ground truth was measured. Measured against an oracle τ (the better of the two fold strengths under full evaluation), the probe captures about 75% of the available fold gain; the residual is $+0.05$ PPL on average and $+0.565$ at worst (Qwen3-1.7B, a held-out model outside the main tables, whose fold gain is -1.67, of which the probe captures -1.11). Finer τ grids, more windows and a higher sampling temperature all leave the residual in place, which points to a mismatch between the generated-text and WikiText-2 selection objectives rather than to statistical power: on that model the τ that is best on its own generated text is not the one that is best on WikiText-2.

lm-eval. Version 0.4.9.2 with three compatibility patches for transformers and datasets 5.x, none of which changes task logic: two guards for the removed `AutoModelForVision2Seq` symbol on the unused vision-language path, and namespaced dataset ids for WinoGrande and BoolQ (`allenai/winogrande`, `aps/super_glue`); GSM8K reads its task file from a copy pointing at the `openai/gsm8k` mirror. The generative benchmarks add math-verify, antlr4 4.11, wonderwords and nltk; GSM8K uses `gsm8k_cot` (chat template off) for base models and `gsm8k_cot_llama` (template on) for Instruct models; the zero-shot suite uses `acc_norm` where defined and `acc` otherwise. On the five arms scored both ways, our own evaluation script agrees with lm-eval's flexible extraction within 0.2–2.1 points. We report flexible extraction throughout; lm-eval's strict rule also requires the exact answer phrase "The answer is".

Long context. Needle retrieval saturates and aggregation does not (Table 3): passkey retrieval is 1.00 in 88 of 90 cells, while the full thirteen-task RULER suite orders the arms, with both ladder arms above uniform dense and within one standard error of the W4A16 reference, and the separation falling in aggregation and question answering rather than in retrieval. The reference row uses block-16 weights, the scale a weight-only deployment would use. Passkey retrieval follows the PoSE evaluation (Zhu et al., 2024) that KVQuant (Hooper et al., 2024) cites (50 trials per length, random depth, integer passkey in 1–50000, greedy 10 tokens, first-integer match). All three passkey misses come from Qwen3-14B's decode-ladder arm (one trial at 8K, two at 32K), and each returns a correct prefix of the passkey and stops (60 for 600, 486 for 48686, 213 for 21302); a five-fold larger generation budget gives the same answers. All thirteen RULER tasks use lm-eval's generators at 100 samples per length; The suite stops at 32K, since longer contexts need RoPE scaling, which changes the model.

C.5 Every published W4A4 perplexity at the matched protocol

Table 1 keeps four published methods. This ledger keeps every W4A4 perplexity we found that is reported on a base checkpoint of Table 1 as token perplexity at sequence length 2048, over the WikiText-2 test stream or under the same C4 protocol (Tables 13 and 14). Three papers meet that condition. RaZeR (Chen et al., 2025b) states the protocol and re-implements the other formats itself at block 16, so those rows are RaZeR's numbers for those methods, not numbers those papers published. ARCQuant (Meng et al., 2026a) and SharQ (Meng et al., 2026b) do not state a sequence length or split, but their FP16 anchors (6.24/6.85/5.02 and 6.25/6.85) agree with the BF16 row of Table 1 within about 0.01, as do all sixteen of RaZeR's, and their own dense NVFP4 baselines (6.95/7.29 and 6.94/7.29 on Llama-3.1-8B/Qwen2.5-7B) agree with RaZeR's 6.95 and our 6.9467/7.2924 to the same tolerance. Comparators are quoted as published, at two decimals.

Table 13: WikiText-2 ledger: every published W4A4 token perplexity on the checkpoints of Table 1, sequence length 2048, KV cache in 16-bit. "Calib." = whether the row uses calibration data (n/s = not stated by RaZeR). The first block is RaZeR's re-implementation of the named formats, from its Table 3. ‡Not NVFP4: FlatQuant and Atom run in their native formats, and the W4A8 row is MXFP4 weights with MXFP8 activations, all as run by ARCQuant. †Below the BF16 perplexity (Appendix A.5). Our rows are shaded; bold marks the lowest value in a column.

			Llama-2		Llama-3		Qwen2.5		Qwen3			
Method	Calib.	Source	7B	13B	3.1-8B	3.2-3B	7B	32B	4B	8B	14B	32B
Re-implemented by RaZeR (Chen et al., 2025b), block 16												
MXFP4, round-to-nearest	no	RaZeR, Tab. 3	7.01	5.83	8.29	10.18	—	—	17.46	11.84	10.20	8.49
NVFP4, round-to-nearest	no	RaZeR, Tab. 3	5.77	5.07	6.95	8.60	—	—	13.88	10.05	8.92	7.99
NF4	no	RaZeR, Tab. 3	5.83	5.14	7.14	8.87	—	—	14.99	10.20	9.16	8.23
BlockDialect	n/s	RaZeR, Tab. 3	5.76	5.08	6.89	8.60	—	—	14.48	10.34	9.13	7.92
MR-GPTQ (Egiazarian et al., 2026)	yes	RaZeR, Tab. 3	5.74	5.07	6.89	8.64	—	—	14.11	10.26	8.95	7.87
Four-over-Six (Cook et al., 2025)	no	RaZeR, Tab. 3	5.72	5.06	6.88	8.53	—	—	13.88	10.00	**8.83**	7.91
Published methods, own numbers												
RaZeR (Chen et al., 2025b)	yes	RaZeR, Tab. 3	5.66	5.01	6.74	8.37	—	—	13.82	**9.83**	8.86	7.84
SharQ (Meng et al., 2026b)	no	SharQ, Tab. 1	—	—	6.73	—	7.15	—	—	—	—	—
ARCQuant (Meng et al., 2026a)	yes	ARCQuant, Tab. 1	—	—	6.87	—	7.28	5.38	—	—	—	—
Ablations reported by SharQ												
sparse + sparse (two sparse operands)	no	SharQ, Tab. 5	—	—	6.95	—	—	—	—	—	—	—
dense + dense (two dense operands)	no	SharQ, Tab. 5	—	—	**6.63**	—	—	—	—	—	—	—
Baselines run by ARCQuant												
NVFP4 + SmoothQuant (Xiao et al., 2023)	yes	ARCQuant, Tab. 2	—	—	6.92	—	7.29	—	—	—	—	—
NVFP4 + QuaRot (Ashkboos et al., 2024)	yes	ARCQuant, Tab. 2	—	—	6.99	—	7.30	—	—	—	—	—
FlatQuant (Sun et al., 2025)‡	yes	ARCQuant, Tab. 1	—	—	6.95	—	7.88	5.74	—	—	—	—
Atom (Zhao et al., 2024)‡	yes	ARCQuant, Tab. 1	—	—	7.52	—	8.96	5.83	—	—	—	—
W4A8, MXFP4 weights + MXFP8 activations‡	no	ARCQuant, Tab. 1	—	—	7.07	—	7.44	5.41	—	—	—	—
This paper												
dense NVFP4, round-to-nearest, block 16	no	Table 1	5.7683	5.0660	6.9467	8.6120	7.2924	5.4059	13.9375	10.0537	8.9367	8.0141
AWSLQ (SL4 with the probe-chosen fold), block 32	no	Table 1	**5.6567**	**4.9913**	6.6757	**8.2734**	**7.1193**	**5.2498**	**13.5468**†	9.9041	8.8585	**7.8071**

Table 14: C4 ledger, under the C4 protocol of Table 1 (256 random 2048-token crops of C4 validation). Only RaZeR reports C4; SharQ and ARCQuant report WikiText-2 alone, and no published C4 number exists for the Qwen2.5 checkpoints. Columns and marks as in Table 13.

			Llama-2		Llama-3		Qwen3			
Method	Calib.	Source	7B	13B	3.1-8B	3.2-3B	4B	8B	14B	32B
Re-implemented by RaZeR (Chen et al., 2025b), block 16										
MXFP4, round-to-nearest	no	RaZeR, Tab. 3	8.67	7.49	11.65	13.49	20.24	15.77	13.62	11.87
NVFP4, round-to-nearest	no	RaZeR, Tab. 3	7.29	6.66	9.94	11.65	17.21	13.75	12.38	11.14
NF4	no	RaZeR, Tab. 3	7.37	6.70	10.21	11.93	18.07	14.06	12.63	11.62
BlockDialect	n/s	RaZeR, Tab. 3	7.27	6.65	9.87	11.57	17.52	13.99	12.46	11.12
MR-GPTQ (Egiazarian et al., 2026)	yes	RaZeR, Tab. 3	7.25	6.65	9.92	12.18	17.27	13.85	12.38	11.09
Four-over-Six (Cook et al., 2025)	no	RaZeR, Tab. 3	7.25	6.63	9.83	11.53	17.21	13.73	12.33	11.14
Published method, own numbers										
RaZeR (Chen et al., 2025b)	yes	RaZeR, Tab. 3	7.19	6.59	9.63	11.27	17.11	**13.60**	12.26	11.02
This paper										
dense NVFP4, round-to-nearest, block 16	no	Table 1	7.2859	6.6585	9.9410	11.6476	17.2620	13.7831	12.4066	11.1560
AWSLQ (SL4 with the probe-chosen fold), block 32	no	Table 1	**7.1788**	**6.5826**	**9.5356**	**11.1733**	**16.9122**	13.6179	**12.2423**	**10.9594**

RaZeR calibrates on the Pile and ARCQuant on 128 WikiText-2 sequences; ARCQuant's 6.87 is its WikiText-2-calibrated value, and it reports 6.90 when calibrated on C4 instead (its Table 6). SharQ, like AWSLQ, uses no calibration data. Every comparator quantizes at block 16, SharQ's sparse operand excepted; the AWSLQ rows use block-32 scales on the ladder. The dense NVFP4 rows of Table 1 predate a rounding fix in our quantization simulator; on Llama-3.1-8B and Qwen3-8B the fix raises them by 0.001 and 0.013 PPL (6.9478 and 10.0665) and leaves the ladder rows unchanged, and the dense-term rows of Table 4 use the fixed simulator.

What the ledger leaves out. A few papers report W4A4 perplexity on these checkpoints, but not on a scale we can put next to ours. Four-over-Six's own paper (Cook et al., 2025) uses word-level perplexity, under which the same Llama-3.1-8B dense NVFP4 cell reads 8.43 instead of 6.95; MixFP4 does the same, and IF4 (arXiv:2603.28765), its successor, evaluates only Nemotron-3 and Qwen3.5 checkpoints and does not state a sequence length or split. MR-GPTQ (Egiazarian et al., 2026) publishes no perplexity at all, so its rows in the ledger are RaZeR's re-implementation. SOAR (arXiv:2605.12245) evaluates Instruct checkpoints only. TORQ (arXiv:2605.19561), an MXFP4 method, does not state its protocol, and its Qwen3-8B BF16 baseline of 9.00 is far from the 9.72–

9.73 measured by RaZeR, by an independent evaluation on Ascend hardware (arXiv:2602.12635), and by us, so we could not place its quantized numbers. ConQuR (arXiv:2605.10793) quantizes to INT4 and averages its perplexity over WikiText-2, PTB, and C4. RaZeR's Table 13 is the one place the rotation methods appear at a matched protocol, but it quantizes the KV cache to 4 bits as well, on Llama-2 only. For reference, that table reports SpinQuant (Liu et al., 2025c) at 5.96/5.24, OSTQuant at 5.91/5.25, DuQuant at 6.08/5.33, and Atom (Zhao et al., 2024) at 6.04/5.27 on Llama-2-7B/13B, against RaZeR's own 4-bit-KV row of 5.72/5.06.

D Kernel Details

D.1 Kernel tiling and pipeline

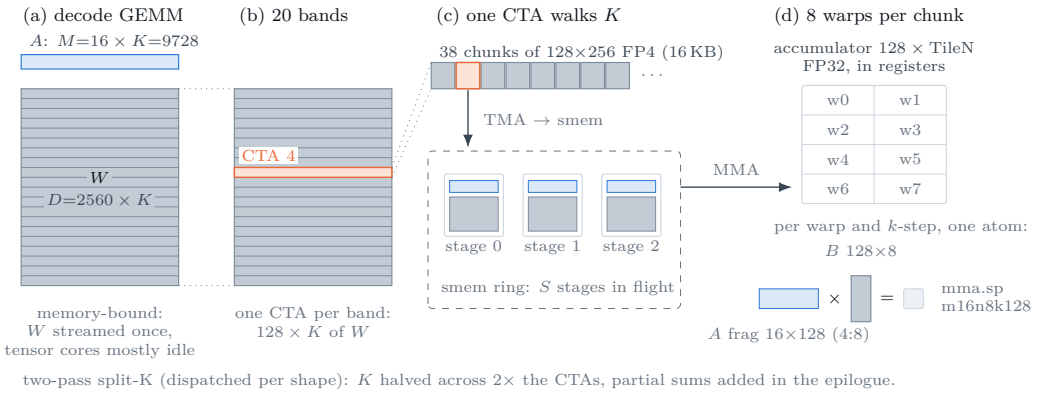


Figure 10: The decode GEMM on sm_120, from the whole product down to one instruction. (a) The product: the activation tile A ($M\times K$) against the weight W, which is streamed once while the tensor cores sit mostly idle. (b) W is cut into row bands of 128 (or 64, per shape), one thread block (CTA) each. (c) A CTA walks K in 256-wide chunks through a shared-memory pipeline fed by TMA. (d) Eight warps cover the accumulator with sparse MMA atoms from register fragments. The two-pass split-K, dispatched per shape (Section 4), halves K across twice the CTAs and adds the partial sums in the epilogue.

The fused kernel issues the N sparse MMAs (`mma.sp` with 4:8 pair metadata and per-operand E4M3 scales) into a single FP32 accumulator and is bit-exact against a host reference with zero spills through $N{=}4$. Its cost is the shared-memory fill: each operand's tile must be filled in every thread block before its MMA can issue, and that fill grows with K, which is what the M-stacked kernel of Section 4 removes.

D.2 Per-GEMM decode latency

Table 15: Decode GEMM latency (μs, cold L2, median of three process replicas, RTX 5090; D = output features, K = input features; 2560:9728 is the Qwen3-4B down projection, the other shapes are Llama-3.1-8B layers). The bracketed number is that column's latency divided by the cuBLASLt-FP4 latency in the same row (below 1, the ladder is faster; our columns are shaded, and a bold cell of ours is faster than every baseline in its row). The stacked kernel runs at every M (one tile per 32 tokens beyond the first); the last column is the pre-stacking fused kernel.

Shape D:K	M	SL3	SL4	cuBLASLt-FP4	CUTLASS dense	fused SL3
2560:9728	16	**16.64** (0.681)	**16.38** (0.670)	24.45	34.82	34.82 (1.424)
	32	**16.51** (0.719)	**16.38** (0.713)	22.98	34.82	35.36 (1.539)
	64	**18.43** (0.802)	**18.43** (0.802)	22.98	34.82	33.34 (1.451)
	128	22.98 (1.000)	**22.72** (0.989)	22.98	34.82	34.21 (1.489)
4096:4096	16	**11.84** (0.662)	**11.78** (0.658)	17.89	18.43	22.53 (1.259)
	32	**11.87** (0.701)	**11.58** (0.684)	16.93	18.43	21.06 (1.244)
	64	**12.90** (0.787)	**12.93** (0.789)	16.38	18.43	19.94 (1.217)
	128	**15.84** (0.967)	**15.81** (0.965)	16.38	18.43	18.98 (1.158)
28672:4096	16	**44.38** (0.722)	**44.45** (0.723)	61.44	68.13	77.25 (1.257)
	32	**44.54** (0.777)	**44.51** (0.776)	57.34	62.94	73.18 (1.276)
	64	**45.57** (0.927)	**44.61** (0.908)	49.15	53.22	63.49 (1.292)
	128	73.98 (1.505)	73.22 (1.490)	49.15	49.15	57.34 (1.167)
6144:4096	16	**14.34** (0.778)	**14.30** (0.776)	18.43	22.53	26.62 (1.444)
	32	**14.34** (0.778)	**14.24** (0.773)	18.43	19.94	26.62 (1.444)
	64	**14.94** (0.811)	**15.10** (0.819)	18.43	18.43	24.77 (1.344)
	128	26.05 (1.413)	26.11 (1.417)	18.43	18.43	20.48 (1.111)
4096:14336	16	**26.18** (0.761)	**26.62** (0.774)	34.40	49.15	63.65 (1.850)
	32	**26.62** (0.775)	**26.18** (0.762)	34.37	49.15	59.42 (1.729)
	64	**32.13** (0.935)	**32.06** (0.933)	34.37	49.15	49.70 (1.446)
	128	39.36 (1.145)	39.39 (1.146)	34.37	49.15	49.73 (1.447)

D.3 Where the decode block's time goes

Table 16 and Figure 11 break down the block totals of Section 6: SL3 is faster than the vendor on all four operations, its preparation adds 18.0 μs to the bare GEMMs against 12.6 μs for the vendor's quantizer, and the fourth rung costs 1.6 μs of preparation rather than GEMM time.

Table 16: Where the decode block's time goes, M=16 on Llama-3.1-8B (μs, RTX 5090, cold L2). The four operation columns are prefix-differenced splits of the chained block, each including its own activation quantization; Σ GEMMs sums the same four GEMMs timed in isolation, and the chain minus that sum is the quantization and preparation the chain adds. A prefix-differenced split has about 2 μs of noise (Appendix D.6). Our block totals are bold.

	QKV 6144×4096	o 4096×4096	gate-up 28672×4096	down 4096×14336	Σ GEMMs	quant. + prep.	chained block
cuBLASLt-FP4	22.5	14.4	67.6	40.4	132.2	12.6	144.7
SL3 (stacked)	18.4	14.3	47.1	34.8	96.7	18.0	**114.7**
SL4 (stacked)	18.4	14.3	49.2	34.8	97.2	19.6	**116.7**

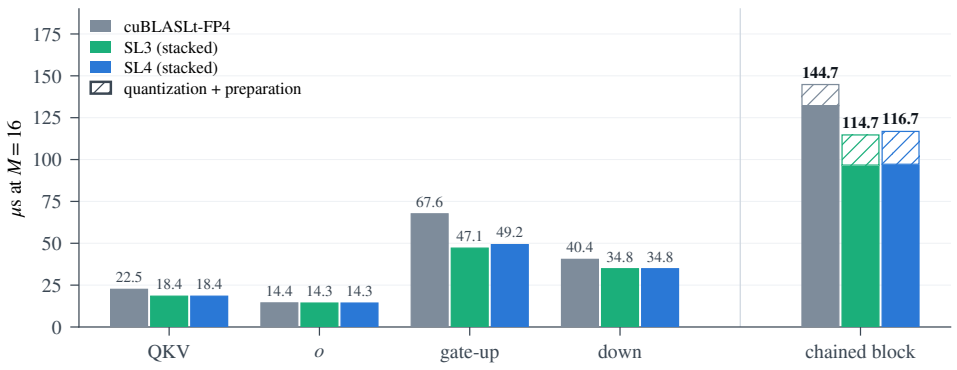


Figure 11: Per-operation latency (Table 16) of the chained Llama-3.1-8B decode block at $M{=}16$ on the RTX 5090 (each bar includes that operation's own activation quantization), and the chained block split into the sum of the isolated GEMMs (solid) and the quantization and preparation the chain adds (hatched).

D.4 The B200 port

On B200 a graph node costs 2–4 μs, so the launches together are about a third of the block and the block is decided by them rather than by GEMM streaming, and three changes outside the GEMM mainloop reduce that overhead. The prepare kernel runs once per linear layer with one CTA per activation row, so its RMSNorm statistics never leave the CTA and no grid-wide barrier remains. Every kernel is issued as a programmatic dependent launch, so its prologue overlaps its predecessor's tail. The fused epilogue keeps its index arithmetic out of the reduce loop and reads the accumulator in 32-column subtiles. Table 17 and Figure 12 split the result: our four GEMMs total 47.6 μs against the vendor's 59.3, with gate-up and down faster outright (the QKV and o splits are unreliable because their launches overlap), and the remaining deficit is preparation, 26.1 μs against 20.5, concentrated at the two sites without a norm (o and down), where the three-level peel is about 1,300 dependent instructions per thread against the vendor quantizer's 100. In isolation each stacked GEMM trails cuBLASLt-FP4 by 4% (geometric mean over the five shapes), and the cause is the fused epilogue's fixed 2 μs, which is exposed when nothing follows the kernel and hidden inside the chain; the single-operand mainloop is faster than the vendor's at three of the five shapes and level with it at a fourth. The third rung is free here as on sm_120, at 0.06 μs per GEMM; the fourth costs 6.2 μs on the block (3.6 of preparation, 2.6 of GEMM) and brings the ladder level with the vendor. The per-tensor activation scale in the B200 block is the previous decode step's, with the block scales computed on the current token; taking the current step's scale needs a grid-wide reduction before any block can be scaled, and that variant trails the vendor by 8%; a barrier-free variant trails by 5%. Simulated on Llama-3.1-8B WikiText-2, the previous-step scale costs $+0.020$ perplexity for SL3 and $+0.007$ for SL4 with one scale shared across the 16-row batch, and $+0.098$ and $+0.063$ per token at $M{=}1$, where it is therefore not used; every accuracy table and every RTX 5090 latency in this paper uses the exact per-token scale. Reusing a per-tensor activation scale is the deployed convention: NVFP4 checkpoints exported by ModelOpt carry a calibrated per-tensor activation scale that vLLM applies unchanged at every step, computing only the per-16 block scales on the current input (vLLM Project, 2026a;b); the previous-step scale refreshes that scale every token instead of never, and the vendor block of Table 17 uses a static calibrated scale as well (Appendix D.6).

Table 17: Where the B200 decode block's time goes, M=16 on Llama-3.1-8B (μs, sm_100a, cold L2, one CUDA graph): the graph-prefix split of the best run into GEMM and quantization or preparation per operation. Programmatic dependent launch overlaps each node's prologue with its predecessor, so the per-node split is indicative and its cells are rounded to a tenth; the block totals (79.87 and 73.73) are medians over 36 and 27 runs. The vendor block launches its quantizer as an ordinary kernel; Appendix D.6 gives the variant with it launched programmatically. Our block total is bold.

		QKV	o	gate-up	down	Σ	chained block
cuBLASLt-FP4	GEMM	6.1	8.2	28.7	16.4	59.3	79.9
	quantization	12.3	0.0	6.1	2.1	20.5	
SL3 (stacked)	GEMM	8.2	6.1	21.0	12.3	47.6	**73.7**
	preparation	10.2	4.1	5.6	6.1	26.1	

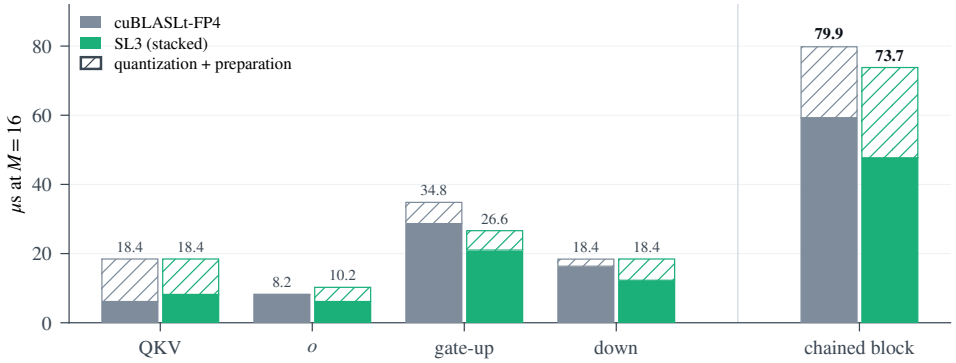


Figure 12: GEMM (solid) and quantization or preparation (hatched) per operation of the chained Llama-3.1-8B decode block at M=16 on the B200, and the chained block total (Table 17).

D.5 Marlin on sm_120: port and W4A16 baseline

Stock Marlin crashes on consumer Blackwell with `cudaErrorIllegalInstruction`: its `createpolicy.fractional.L2::evict_first` and `cp.async …L2::cache_hint` instructions are illegal on sm_120. A one-instruction port (plain `cp.async.cg` under `__CUDA_ARCH__` $\geq$ 1200) fixes it. Separately, the grouped 64×256 configuration carries a shared-memory race that is latent in the upstream code. At group size 128 with a K-tile of 64 (thread$_k$=64), one scale slot serves two pipeline stages, and the `cp.async` refill overwrites it while lagging warps are still reading; no architecture places a barrier there. sm_80 and sm_86 win the race by timing. sm_120 with CUDA 13 loses it: results are intermittently wrong at M=32 and always wrong at $M \geq 64$, with 1–4 $\times$ 10^{-2} relative error. One architecture-guarded `__syncthreads` before the refill fixes it. With both fixes, every timed Marlin output has mean relative error $\leq 2 \times 10^{-3}$ and maximum $\leq 10^{-2}$ against an FP32 matmul of the dequantized weights. cuBLASLt-FP4 raises an illegal memory access at M=1, D=28672, K=4096 (reproducible on cuBLASLt 13.2), so that cell is missing. At $M \leq 8$ weight-only matrix-vector kernels beat cuBLASLt's FP4 GEMM on sm_120 as well (SceneWorks, 2026; Lee et al., 2026); the claim of Section 6 is the tensor-core regime from M=16.

Table 18: W4A16 (Marlin, sm_120 port) versus FP4 paths at D=2560, K=9728 (μs, same session, cold L2, medians of three process replicas; our columns are shaded, and a bold cell of ours is faster than every baseline in its row). The last column is Marlin / cuBLASLt-FP4 as a geometric mean over five layer shapes (*four at M=1, where cuBLASLt fails at D=28672, K=4096; see the text above). The stacked columns run at every M (one tile per 32 tokens beyond the first). The CUTLASS dense column is the dense NVFP4 kernel the dispatch uses at prefill. At M=1 Marlin runs at 18.4 μs and our stacked kernels at 16.5 and 16.4: moving the same weight bytes, W4A16 (4-bit weights with the activations left in 16-bit) and the FP4 ladder are one latency class. The stacked kernels pad M=1 to 32 rows, so their M=1 and M=32 cells coincide. Every column is from the same sitting as Table 15.

M	Marlin W4A16	cuBLASLt-FP4	SL3	SL4	CUTLASS dense	Marlin / cuBLASLt-FP4 (5-shape geomean)
1	18.4	22.5	**16.5**	**16.4**	34.8	0.759*
16	18.9	24.4	**16.6**	**16.4**	34.8	0.735
32	30.7	23.0	**16.5**	**16.4**	34.8	1.079
64	43.5	23.0	**18.4**	**18.4**	34.8	1.685
128	47.1	23.0	23.0	**22.7**	34.8	2.280
512	118.8	30.6	61.9	61.9	34.8	4.537

D.6 Block size, timing, and builds

Why the activation block is 32. The block-scaled sparse MMA on sm_120 takes one 512-byte fragment of A holding 1024 stored FP4 values (16 rows by 64 stored values, one 128-element slice of K) and a scale operand with four E4M3 entries per row. For a dense operand those 64 values are 64 logical elements, one scale per 16, which is the NVFP4 block. For a 2:4-sparse operand the same 64 stored values stand for 128 logical elements, and the instruction still carries only four scales (there is no eight-scale variant on sm_120; CUTLASS documents 32 or 64 elements per scale for sparse block-scaled operands (NVIDIA, 2026a)), so each scale spans 32 logical elements. The weight operand of the same instruction must use the same scale width, which is why our weights are block-32 as well. Per rung, the activation payload is 2.75 bits per logical element: four stored E2M1 values per eight elements (2 bits), a 2-bit index for each of the two kept pairs (0.5 bit), and one E4M3 scale per 32 elements (0.25 bit); three rungs carry 8.25 bits per element and four carry 11, against 4.5 for dense NVFP4. On the weight side the wider block is a saving: 4 bits plus one E4M3 scale per 32 elements is 4.25 bits per weight against 4.5 at block 16, so our kernel streams 5.6% fewer weight bytes than cuBLASLt's block-16 format. The utilization figures of Section 6 use the block-32 byte count for both kernels.

Latency protocol. All latencies are cold-L2 (a weight ring larger than the 96 MB L2), median of 200 CUDA-event iterations after 20 warm-ups, three process replicas per point (worst spread 7.8%, one timer tick at the shortest shapes; 2.4% on the chained block), on an idle RTX 5090, with cuBLASLt 13.2 timed on the fastest algorithm its heuristic returns. Chained-block numbers come from a CUDA-graphed chain of the four linear layers of one Llama-3.1-8B decode block with each side's own quantizer, both sides measured in the same session. cuBLASLt ships no activation quantizer, so the vendor block uses one we wrote for the benchmark: one pass with hardware E2M1 conversion, RMSNorm fused at the two norm sites, per-16 E4M3 block scales, and a static calibrated per-tensor scale with no per-token maximum pass, which is both the standard deployed configuration and the cheaper of the two options. Its cost is included in the vendor's block total. The ladder's RTX 5090 numbers compute the exact per-token tensor scale.

Kernel timing and builds. The timer quantizes at 1.024 μs, and small differences are reported in timer ticks. Per-operation splits come from prefix differencing of the graphed chain: the chain is graphed up to each successive operation and the medians are subtracted. In-stream events would inflate the totals by 6–10%, and splits below ≈ 2 μs sit at the noise floor. At these shapes cuBLASLt dispatches its JIT-generated `nvjet` kernels, 32×32-tile with split-K. All configurations compile with 168 registers and zero spills through N=4. The register-local epilogue reserves 16 B of epilogue shared memory against the stock 14,336 B; the earlier staged variant used 8,448 B. The prepare kernel selects the pairs, writes the 4:8 metadata and E4M3 scales, and leaves the next residual behind; the two-pass split-K halves K across twice the thread blocks and sums in the epilogue, as cuBLASLt's own split-K decode kernels do. The two-pass split-K is used only where $(K/2) \bmod 256 = 0$ (each half of K a whole number of 256-wide chunks); unlike stream-K on sm_120 it never deadlocks, and

its outputs match the reference to BF16 rounding in all ten correctness checks. The B200 results reproduce on four hosts with two driver versions (580.167 and 595.91): isolated latencies to within a timer tick, the chained-block medians exactly. The B200 block is measured with the protocol above and a 169 MiB weight ring against the 126.5 MiB L2. Programmatic dependent launch is applied to every kernel we write. cuBLASLt exposes no such attribute, so the vendor's GEMMs are ordinary launches, and the 79.9 μs vendor block of Table 17 launches its quantizer as an ordinary kernel as well. Issuing that quantizer as a programmatic dependent launch brings the vendor block to 75–78 μs (medians on two hosts), and the ladder remains 2–6% ahead. The fused epilogue attaches through CUTLASS's stock warp-specialized epilogue adapter.

D.7 The sparse mainloop's headroom

The sparse instruction prices a rung at half a dense term, and the RTX 5090 delivers that price: in a register-resident microbenchmark with no memory traffic (secYOUre, 2026), built for sm_120a in our scale format (UE4M3 scales, four per instruction), the sparse block-scaled FP4 MMA sustains 4086 TFLOPS against 2043 for the dense one, a ratio of 2.00. The kernels do not (Table 19): at compute-bound shapes CUTLASS's dense mainloop reaches 69–71% of its instruction peak and its sparse mainloop 40–44%, so one sparse operand runs only 1.16–1.25× faster than the dense GEMM. Under Nsight Compute at 4096^3 the dense mainloop keeps the tensor pipe active for 76% of the elapsed cycles and the sparse mainloop for 50%. The sparse loop's extra stalls sit on the shared-memory load queue (1.32 against 0.06 warps per issue) and on load completion (1.86 against 1.30), with a busier tensor-pipe queue behind them (2.03 against 1.33): bursts of fragment loads (the operand, its 4:8 metadata from global memory, its scales) that are not interleaved with the MMAs, followed by bursts of MMAs on which the other warps wait. The two mainloops differ at the source: the dense collective preloads the next k-block's fragments before computing the current one, and the sparse collective loads and then computes. Metadata handling, register pressure and synchronization differ as well, and prefetching lengthens register lifetimes, so how much of the gap a prefetching sparse mainloop closes is an open question.

The fused ladder inherits the sparse loop. Measured the same way at M=4096–8192 on two shapes, three fused rungs cost 1.71–1.74× one dense term (0.57–0.58 per rung against 0.50 at the instruction) and 0.87–0.90 of a two-term dense pair timed as a dense GEMM over $2M$ rows, the M-stacked pair without its epilogue reduce; four rungs cost 2.71–2.89× one term and 1.40–1.50 of the pair. These are the kernel's prices for the budgets of Figure 3. With R the sparse-to-dense throughput a rung achieves through its mainloop, three rungs cost $3/(2R)$ of the pair. One operand in the stock kernel reaches R=1.16–1.25 (Table 19); the fused ladder, whose rungs share one weight fill, reaches $R \approx 1.7$, the 0.87–0.90 above; parity is R=1.5, and the instruction's R=2 gives 0.75. The fused ladder's prefill cost in Section 4 (1.62–1.92× the dense GEMM) is an upper bound on its compute-bound cost for the same reason.

Table 19: One sparse operand against the dense GEMM at compute-bound shapes (μs, RTX 5090, cold L2, median of 200 iterations; the stock CUTLASS sm_120 sparse mainloop against the CUTLASS dense NVFP4 128×128×128 kernel). Throughput counts logical multiply-accumulates for the sparse operand; the percentage is of the instruction peak, 2043 TFLOPS dense and 4086 sparse.

Shape D:K	M	dense	sparse	speedup	dense TFLOPS	sparse TFLOPS
2560:9728	4096	141.9	118.8	1.19×	1438 (70%)	1718 (42%)
2560:9728	8192	281.2	225.3	1.25×	1451 (71%)	1811 (44%)
4096:4096	4096	97.3	84.0	1.16×	1413 (69%)	1637 (40%)
4096:4096	8192	188.4	153.6	1.23×	1459 (71%)	1790 (44%)