
Small-State Coset Trellis Coding at IQ2_XXS Size: An Equal-Bytes Study in llama.cpp

Harshit Goyal¹

Abstract

Trellis-coded quantization of LLM weights is established (QTIP, EXL3), and `ik_llama.cpp`'s IQ2_KT already provides a trellis type in a llama.cpp fork. We ask how much a small-state code adds at a fixed file size, that of llama.cpp's IQ2_XXS, and which part of the pipeline supplies the gain. We store Ungerboeck's 8-state set-partitioned code over 4-weight symbols in a GGUF file, decoded by an unmerged CUDA patch to llama.cpp, behind a random Hadamard rotation and QuIP's LDLQ rounding, with input-column scales that minimise, under a diagonal Hessian model, QuIP's incoherence bound on LDLQ's error. On Qwen3-4B (post hoc) the file reaches WikiText-2 perplexity 16.64, against 28.61 for IQ2_XXS and 18.15 for the strongest IQ2_KT file of this size we built; it decodes at batch 1 at least as fast as IQ2_XXS but processes prompts at $0.74\times$. Pre-registered, it beats IQ2_KT on Llama-3.1-8B (8.44 against 9.36 with a BOS at every window, Section 4.3; IQ2_KT made from Q8_0) and loses on Qwen3-14B (9.57 against 9.40; tie on C4). At equal bytes, removing the trellis memory costs 0.076 nats per token: about 80% (post hoc) of the margin over a perplexity-tuned IQ2_KT, but 14% of the gain over IQ2_XXS.

1. Introduction

At about 2 bits per weight a 70B model needs about 19 GB instead of 141 GB in fp16 (calculated, Appendix L.4). Deployed runtimes such as llama.cpp (Gerganov & llama.cpp contributors, 2023) serve this rate with fixed-rate grid codes, the i-quants, which have fast kernels on many backends. The one we compare against, IQ2_XXS (Kawrakow, 2024b), packs 8-weight groups into indices of an E_8 -derived grid at

Appendices D, E (except E.4), H, I, J and L are in the supplementary material. ¹Genrius AI. Correspondence to: Harshit Goyal <harshit@genrius.in>.

2.0625 bits per weight on the quantized tensors; on Qwen3-4B it has $2.3\times$ the fp16 perplexity.

Codes that keep more accuracy at this rate do not run from such files. Academic vector and trellis codes (QuIP# (Tseng et al., 2024a), AQLM (Egiazarian et al., 2024), QTIP (Tseng et al., 2024b), LLVQ (van der Ouderaa et al., 2026)) lose far less (QTIP and LLVQ, as run by the LLVQ authors, reach $1.25\text{--}1.37\times$ fp16 perplexity on Qwen3-4B) but bring their own kernels. Trellis types for local runtimes live in a fork (the IQ*_KT types of ik_llama.cpp (Kawrakow & ik_llama.cpp contributors, 2024), whose 2-bit IQ2_KT is our closest baseline) or a separate library (EXL3 (turboderp-org, 2025)). We ask a systems question: how much 2-bit accuracy fits in a llama.cpp file of IQ2_XXS size, at a batch-1 decoding speed no lower than IQ2_XXS's, and what limits it.

Our system, Pingala (Figure 2), uses Ungerboeck's 8-state set-partitioned trellis code (Ungerboeck, 1982) as a vector quantizer on 4-weight symbols, stores it in a GGUF file and decodes it with a patch set on llama.cpp commit 7ac59a6, not a separate engine. Nothing in the coding structure is new (Appendix A), and 8 states are not an optimum: a 16-state code is better (Section 5.3). Our file, B4, gives each tensor one rate class and sets its input-column scales by minimising an incoherence bound (Section 3). The body compares B4 with IQ2_XXS and IQ2_KT; earlier builds and their controls are in Appendix D. Table 2 states each claim once, with its evidence and whether it was pre-registered.

Contributions.

1. **A same-harness, equal-bytes comparison with the closest deployed trellis type, including a pre-registered loss** (K1–K5; Qwen3-4B post hoc, Llama-3.1-8B and Qwen3-14B pre-registered). The file stores Ungerboeck's 8-state code as a 4-dimensional TCVC (Fischer et al., 1991) in GGUF; the baseline is ik_llama.cpp's IQ2_KT (Kawrakow & ik_llama.cpp contributors, 2024), six files of this size on Qwen3-4B. At no more bytes than IQ2_XXS it beats IQ2_XXS on three models and IQ2_KT on two, loses to IQ2_KT on Qwen3-14B, and, on Qwen3-4B, decodes at batch 1 at least as fast as IQ2_XXS.

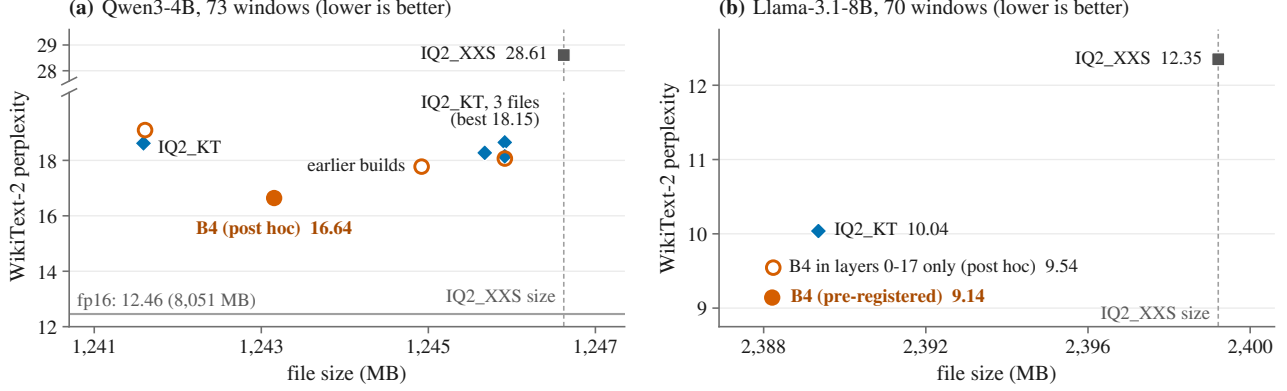


Figure 1. At IQ2_XXS size or less, B4 has lower WikiText-2 perplexity than IQ2_XXS and every IQ2_KT file we built, on both models shown; on Qwen3-14B IQ2_KT is lower (Table 1). Perplexity vs. file size (exact bytes, in MB): (a) Qwen3-4B, 73 windows (y axis broken, fp16 line at the bottom); (b) Llama-3.1-8B, 70 windows, each after the first without a BOS (B4 pre-registered; Section 4.3). Vermillion: ours (B4 filled; open markers: our earlier and partial builds, Appendix D); blue diamonds: IQ2_KT; grey squares: IQ2_XXS. In (a) the left IQ2_KT file is the smaller knapsack recipe; the plot’s “3 files” are 3 of the 5 IQ2_KT files at the larger size (knapsack, tuned and C4-calibrated, the last under an open circle).

2. **An equal-bytes decomposition of the gain (K6).** The trellis memory supplies most of the margin over a tuned IQ2_KT (a post hoc reading) but a small part of the gain over IQ2_XXS; the rest comes from the rest of the pipeline (QuIP’s LDLQ (Chee et al., 2023), the rotation, our column scale and the 256-point table; not separated). This extends QTIP’s source-level comparisons (Tseng et al., 2024b) to a fixed-size format; on a Gaussian source the code stays 1.74–1.91 dB from the bound at 4 to 32 states.
3. **A column-scale exponent derived for QuIP’s LDLQ bound (K7).** Under a diagonal Hessian model, Hölder’s inequality gives $c_j \propto s_j^{1/6} E_j^{-1/3}$ as the minimiser of QuIP’s incoherence bound, moving QuIP’s quarter power (Chee et al., 2023) to SmoothQuant’s form with $\alpha = 1/3$ (Xiao et al., 2023); the exponent was not swept.
4. **An equal-bytes negative result for per-tile allocation (K8).** Per-tile rate classes of one trellis code lower a weight-error proxy yet raise perplexity, extending evidence that layer-wise error misranks allocations (Sieberling et al., 2024; Malinovskii et al., 2025).

2. Background and Related Work

Trellis-coded quantization. Ungerboeck’s trellis-coded modulation (Ungerboeck, 1982) partitions a signal set into subsets and lets a convolutional code with 2^ν states choose the subset at each step. Trellis-coded quantization (TCQ) (Marcellin & Fischer, 1990) uses the same structure as a source code: the Viterbi algorithm finds the lowest-distortion path and the decoder follows it with no search. For a memoryless uniform source, 256-state TCQ comes within

0.21 dB of the distortion-rate bound. TCVQ (Fischer et al., 1991) replaces the scalar alphabet with vector codebooks. With $\nu = 3$ the state is the last three input bits, so the 8 states are $\{0, 1\}^3$ and the edges form the de Bruijn graph $B(2, 3)$. We name the 8 states after the ganas, the eight syllable triples of Pingala’s Chandahsutra (c. 3rd–2nd century BCE; Pingala, 1938), in the binary-counting order of its *prastara* (van Nooten, 1993; Plofker, 2009), and call the code the Pingala-8 trellis. Transitions and labels are Ungerboeck’s (labels taken from the ganas cost 7.7% in perplexity in an entropy-coded reference build; Appendix I), so the name is a credit, not a source of the gain: the system is named after, and inspired by, Pingala’s prosody (Appendix B).

LLM weight quantization. GPTQ (Frantar et al., 2023) introduced Hessian-aware column-by-column rounding; AWQ (Lin et al., 2024a) and SpQR (Dettmers et al., 2024) protect salient weights; QuIP (Chee et al., 2023) added incoherence processing and LDLQ rounding; QuIP# (Tseng et al., 2024a) the randomized Hadamard transform and E_8 codebooks; AQLM (Egiazarian et al., 2024) learned additive codebooks, and PV-Tuning (Malinovskii et al., 2024) fine-tunes the compressed representation. LLVQ (van der Ouderaa et al., 2026) uses Leech-lattice shape-gain codes and reports the strongest 2-bit Qwen3-4B results we know of.

Trellis quantizers for LLMs, and what is prior art. None of the coding machinery is ours (Appendix A.1 lists what each work states). Trellis-coded quantization and set partitioning are due to Ungerboeck (1982) and Marcellin & Fischer (1990), the vector form is TCVQ (Fischer et al., 1991), and small-state trellis quantization of network weights is standardised as dependent quantiza-

tion, an 8-state scalar trellis, in ISO/IEC 15938-17 (Haase et al., 2020; Kirchhoffer et al., 2022). For LLMs, QTIP (Tseng et al., 2024b) introduced TCQ with a bitshift trellis and incoherence processing; EXL3 (turboderp-org, 2025) ships a streamlined QTIP variant in a separate inference library. The IQ*_KT types of ik_llama.cpp (Kawrakow & ik_llama.cpp contributors, 2024; Kawrakow, 2024a; 2025c) are trellis types at this size in a llama.cpp fork, with CUDA batch-1 speed close to IQ2_XXS as reported by their author; IQ2_KT is the closest prior format in size, runtime family and speed, so we make no “first trellis type at this size” claim and give it same-harness rows. Proteus (Yang et al., 2026) (ICML 2026), the closest published precedent for a 2-bit trellis with a cheap decoder, generates code values by integer arithmetic with no table loads; CCQ (Zhou et al., 2025) decodes a convolutional code by bit shifts; both run in their own engines. XOR-Trellis (Que et al., 2026), CPTC (Anonymous, 2026), Tetra (Malandrino, 2026) and CurveTQ (Ding et al., 2026) are concurrent preprints that, by their abstracts, do not target a llama.cpp file; we did not run them. Rotation and LDLQ rounding come from QuIP and QuIP# (Chee et al., 2023; Tseng et al., 2024a). Our column scale has the activation-aware form of SmoothQuant and AWQ (Xiao et al., 2023; Lin et al., 2024a), and deriving its exponent from an error bound follows QuIP’s quarter power and TurboPress (Chee et al., 2023; Johnson, 2026). The batch-1 kernel follows LUT-GEMM, T-MAC and FLUTE (Park et al., 2022; Wei et al., 2024; Guo et al., 2024). What we add is the fixed-size file, its same-harness comparison with IQ2_KT, the exponent for QuIP’s LDLQ bound, and the equal-bytes analyses. QTIP’s own ablation (Tseng et al., 2024b, Table 10) found that an $L = 8$ trellis, the largest whose trellis and codebook can be stored (2^8 entries), loses accuracy to large- L trellises; our accuracy gap to QTIP is consistent with that.

Bit allocation under a storage budget. Allocating bits unevenly under a size budget is established: Q-Palette (Lee & Song, 2025), ScaleBITS (Li et al., 2026), AlignQuant (Zhang et al., 2026) (two-dimensional tiles as the unit of precision) and SAGE (Yan et al., 2026); the llama.cpp recipes already mix types across tensors. Our per-tile classes are a small member of this family and we do not compare against these allocators; we report that in our format the weight-error proxy that drives the allocation ranks allocations in the wrong order (Section 5.4).

3. Method

Front end. Each linear weight matrix $W \in \mathbb{R}^{m \times n}$ is prescaled per input column by c_j (below), rotated with a random-sign fast Walsh–Hadamard transform over 256 columns, and given one fp16 scale per 256 weights per row. The layer Hessian $H = \frac{1}{N} X^\top X$ is collected by propagat-

ing calibration text (256 sequences of 2048 tokens for B4, 512 for our earlier builds; English C4 (Raffel et al., 2020), checked to share no word 10-gram with the importance-matrix text or the WikiText-2 test set) through the fp16 model layer by layer, as in GPTQ and QuIP. All side information (column scales, rate classes, clamp deltas) is stored in the file, so the reported size counts every bit the runtime reads.

Column scale. Model the Hessian as diagonal ($H_{jj} \propto s_j$, with s_j the importance-matrix entry of input column j) and let $E_j = \sum_r W_{rj}^2$ be the column energy. After the rotation the scaled weights of a block b of B columns are close to i.i.d., and QuIP’s incoherence upper bound on LDLQ’s error (Chee et al., 2023) gives, for the block, the proxy loss

$$\begin{aligned} L_b(c) &= \left(\frac{1}{B} \sum_{j \in b} c_j^2 E_j \right) \left(\frac{1}{B} \sum_{j \in b} \frac{\sqrt{s_j}}{c_j} \right)^2 \\ &\geq \left(\frac{1}{B} \sum_{j \in b} (s_j E_j)^{1/3} \right)^3 \end{aligned} \quad (1)$$

by Hölder’s inequality with exponents $(3, \frac{3}{2})$, with equality iff $c_j \propto s_j^{1/6} E_j^{-1/3}$ (derivation: Appendix E.4). This is the activation-aware channel scale of SmoothQuant and AWQ (Xiao et al., 2023; Lin et al., 2024a) in SmoothQuant’s form with $\alpha = 1/3$ and RMS in place of maximum magnitudes; the derivation fixes α for this upper bound only. QuIP’s incoherence processing already rescales columns by $(H_{jj}/E_j)^{1/4}$, the same form with $\alpha = 1/2$, stated in its supplement as the minimiser of $\text{tr}(D^{-1} H D^{-1}) \|W D\|_F^2$, its proxy $\text{tr}(H) \|W\|_F^2$ for the output error (Chee et al., 2023, Alg. 1 and Supp. B.1); Narita & Sato (2026) obtain the same L_2 form by Cauchy–Schwarz from a bound on the activation-quantization residual of W4A4 quantization; Eq. 1 instead minimises QuIP’s incoherence bound for LDLQ, which moves the exponent to $1/3$. The bound is not LDLQ’s error: with error feedback the scale acts through the Cholesky pivots of the scaled Hessian (Ben David et al., 2026; Lifar et al., 2026; Ordentlich & Polyanskiy, 2026), and the exponent that minimises the bound is not shown to minimise the error. B4 normalises the scale to geometric mean 1, clips it to $[1/64, 64]$ and stores it as one fp16 value per column. Our previous build used $c_j = \sqrt{\max(s_j/\bar{s}, 10^{-2})}$, which ignores E_j .

Coset trellis code (Pingala-8). Weights are grouped into 4-dimensional symbols x_t along a row. A table A holds 256 int8 4-vectors and its coset is $B = A + c$ for one fixed int8 vector c . The encoder state $s_t \in \mathcal{G} = \{0, 1\}^3$ is the last three trellis input bits. Following Ungerboeck’s feedforward labelling, the stored index MSB is $z_1 = u_t \oplus s^{(0)} \oplus s^{(2)}$ and the subset bit is $z_0 = s^{(1)}$, which selects A or B . For a

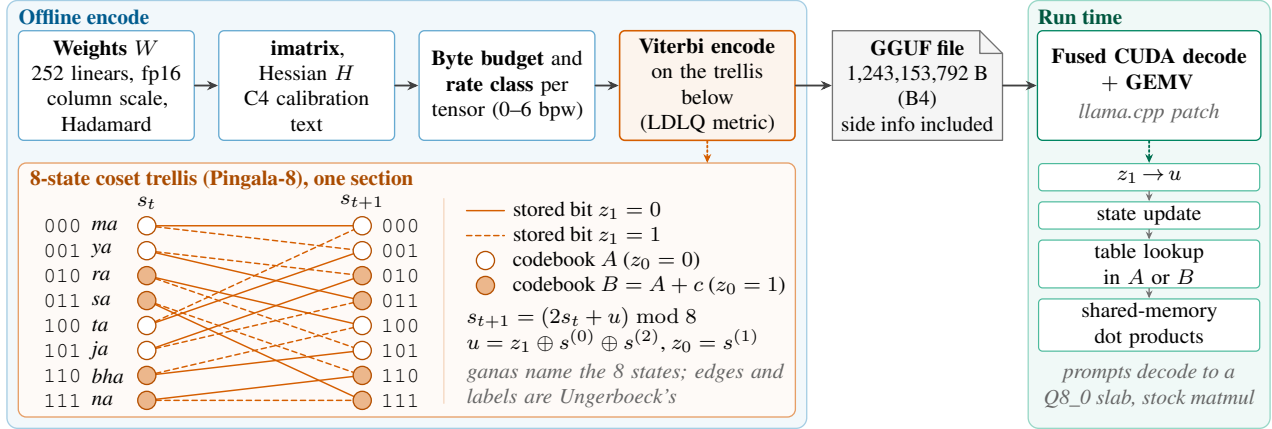


Figure 2. Pingala encodes offline and decodes in one fused CUDA kernel (B4 shown): a Viterbi search over the 8-state coset trellis (the Pingala-8 trellis) writes the GGUF file, and the kernel decodes it at run time. States are named by the gana convention (*ma* is 000, *na* is 111); edges and labels are Ungerboeck's.

row w the encoder solves

$$\min_{u_{1:T}} (w - \hat{w}(u))^T H (w - \hat{w}(u)), \quad \hat{w}_t = \sigma \mathcal{C}_{z_0(s_{t-1})}[i_t], \quad (2)$$

with a per-survivor Viterbi search and LDLQ error feedback (Chee et al., 2023) (512 distance evaluations per weight; Appendix C). The decoder recovers u from z_1 word-parallel over 32-symbol slices (state reset per slice), so the trellis adds no bits and $O(1)$ work per weight.

Rate classes and allocation. A unit (a whole tensor in B4, a tile of $32 \text{ rows} \times 256 \text{ columns}$ in our first build) is coded in one of 8 rate classes (0, 1, 1.5, 2, 2.5, 3, 4 and 6 bits per weight); stage 1 is a 4-, 6- or 8-bit index into A and the higher classes add residual stages. Every tile of a class has the same byte count and the class is uniform across a warp, so the decoder does not diverge. With distortion $D_j(k)$ of unit j at class k and a byte budget B , allocation is the multiple-choice knapsack

$$\min_{k_j} \sum_j \lambda_j D_j(k_j) \quad \text{s.t.} \quad \sum_j |j| b_{k_j} \leq 8B, \quad (3)$$

where λ_j is the end-to-end sensitivity of unit j ; our allocator uses $\lambda_j = 1$ on the importance-weighted error, the *weight-error proxy*. B4 gives each of the 252 linear tensors of Qwen3-4B one class at the nearest rate, with byte carry and salient protection off, and stores each as one int8 GGUF tensor (code tables, a header word with the class, the fp16 scales and the trellis payload; no tile words or row permutations) plus a sparse clamp delta. The file has 1,243,153,792 bytes, 3,467,520 fewer than IQ2_XXS. The per-tile build, the size-matched flat builds and the layout change that led to B4 are in Appendix D.

Runtime in llama.cpp. The runtime is two unmerged patches on llama.cpp commit 7ac59a6, a CUDA path and

an x86 AVX2 CPU backend. Batch-1 generation uses an activation-table GEMV, as in LUT kernels (Park et al., 2022; Wei et al., 2024; Guo et al., 2024) (instruction counts: Appendix C). Any batch above one, prompts included, decodes each matrix to an 8-bit Q8_0 slab for the stock Q8_0 multiply; perplexity and KL divergence use this path, generation speed the GEMV. For B4 the GEMV path itself gives perplexity 16.68 against 16.64 on the slab (Appendix E.4). The native file decodes to the evaluated 8-bit carrier (99.975–99.981% of slab blocks bit-equal), and a corrupted trellis header corrupts its whole tensor (Appendix C).

4. Experiments

Setup. Models: Qwen3-4B (Yang et al., 2025), our development model, and Llama-3.1-8B (base) and Qwen3-14B as pre-registered models. Baselines: llama.cpp IQ2_XXS and the other stock quants made from the same fp16 file and importance matrix, and ik_llama.cpp IQ2_KT built from the fork (commit 77bef17) with the same inputs. On Qwen3-4B we built six IQ2_KT files of this size (Table 1): two knapsack recipes, three recipes tuned for perplexity and registered with predictions before they were quantized (the best is *tuned*), and the *C4-calibrated* file, which tests whether our held-out margin is a calibration-domain effect; the fork's default recipe is 4.6% larger than B4. The two binaries agree on the same IQ2_XXS file (paired $\Delta\text{NLL} = -0.0002 \pm 0.0006$). Hardware: one RTX 5070 Ti Laptop GPU (12 GB). Sizes are exact file bytes.

Metrics. Perplexity: WikiText-2 test (Merity et al., 2016), all tokens, windows of 4096 tokens in llama.cpp's strided mode (the protocol of van der Ouderaa et al. (2026)): 73 windows on Qwen3-4B (fp16: 12.4567) and 70 on Llama-3.1-8B. Paired ΔNLL is the mean over windows of the per-window difference in negative log-likelihood (nats per

Table 1. B4 has the lowest perplexity on both texts and the lowest KL divergence of every llama.cpp-format file at IQ2_XXS size that we measured on Qwen3-4B, and the lowest perplexity on Llama-3.1-8B (perplexity only); on Qwen3-14B IQ2_KT is lower (pre-registered loss). WikiText-2 test, context 4096 (73 windows on Qwen3-4B and Qwen3-14B, 70 on Llama-3.1-8B); C4: held-out text, 73 windows. Δ NLL: B4 minus the row, paired per window (nats per token, 95% interval; windows on which B4 is better); negative means B4 is better. ^cInterval corrected for lag-1 autocorrelation. KL div.: 72 chunks to Q8_0 (mean $\pm$ s.e.). IQ2_KT (ik_llama.cpp fork): knapsack recipes with a calculated sensitivity proxy, at two sizes; tuned: the best of three recipes tuned for perplexity; C4-calibrated: the larger knapsack recipe with an importance matrix from our C4 calibration text. $-$: not measured or not paired. Our earlier builds: Table 7 (Appendix D).

Format	Runtime	File bytes	PPL	Ratio	Δ NLL, B4 $-$ row	C4 PPL	Δ NLL C4, B4 $-$ row	KL div., 72 ch.
<i>Qwen3-4B</i>								
fp16	llama.cpp	8.05 GB	12.4567	1.000	$-$	$-$	$-$	$-$
IQ2_XXS	llama.cpp	1,246,621,312	28.6083	2.297	-0.542 ± 0.020 (73/73)	37.28	-0.425 ± 0.013^c (73/73)	1.397 ± 0.012
IQ2_KT, knapsack, smaller	ik fork	1,241,589,376	18.6165	1.494	$-$	27.74	$-$	0.863 ± 0.010
IQ2_KT, knapsack	ik fork	1,245,914,752	18.6481	1.497	$-$	28.00	$-$	0.876 ± 0.010
IQ2_KT, tuned	ik fork	1,245,675,136	18.2721	1.467	-0.094 ± 0.010^c (71/73)	27.33	-0.115 ± 0.008^c (73/73)	0.852 ± 0.010
IQ2_KT, C4-calibrated	ik fork	1,245,914,752	18.1480	1.457	-0.087 ± 0.011^c (71/73)	27.15	-0.108 ± 0.008^c (73/73)	0.893 ± 0.010
Ours (B4)	ours	1,243,153,792	16.6413	1.336	0	24.36	0	0.585 $\pm$ 0.008
IQ2_KT, default recipe	ik fork	1,300,786,816	17.2750	1.387	$-$	$-$	$-$	$-$
IQ2_XS	llama.cpp	1,354,100,352	19.0193	1.527	$-$	$-$	$-$	$-$
Q2_K	llama.cpp	1,669,500,032	15.0133	1.205	$-$	$-$	$-$	$-$
<i>Llama-3.1-8B (base); IQ2_KT quantized from Q8_0; windows after the first start without a BOS</i>								
IQ2_XXS	llama.cpp	2,399,212,576	12.35	$-$	-0.301 ± 0.010^c (70/70)	$-$	$-$	$-$
IQ2_KT	ik fork	2,389,350,432	10.04	$-$	-0.094 ± 0.005^c (70/70)	$-$	$-$	$-$
Ours (B4, pre-registered)	ours	2,388,210,816	9.14	$-$	0	$-$	$-$	$-$
<i>Qwen3-14B (pre-registered)</i>								
IQ2_XXS	llama.cpp	4,299,192,000	10.5413	$-$	-0.097 ± 0.007^c (73/73)	16.6968	-0.091 ± 0.006^c (73/73)	$-$
IQ2_KT	ik fork	4,288,068,800	9.3985	$-$	$+0.018 \pm 0.008^c$ (12/73)	15.2349	$+0.001 \pm 0.006^c$ (33/73)	$-$
Ours (B4, pre-registered)	ours	4,288,702,112	9.5684	$-$	0	15.2490	0	$-$

token) between two files (B4 minus the other file, negative favours B4; other pairs first minus second), with ± 1.96 standard errors and the windows won. Adjacent windows are correlated (lag-1 autocorrelation $+0.40$), so margins marked corrected use an autocorrelation-corrected interval, and small margins also get a moving-block bootstrap (blocks of 5 windows). KL divergence is against the next-token distribution of the 8-bit Q8_0 file, which stands in for fp16 (0.33% above it in perplexity), on 72 chunks of WikiText-2 test (development text, not held out). Held-out text: C4 validation, 73 windows; it is held out from selection but in-domain for every file and for our C4 Hessians, a bias the C4-calibrated IQ2_KT tests. All files share one C4-train importance matrix, except the C4-calibrated IQ2_KT, whose matrix comes from our C4 calibration text. Every build was evaluated on WikiText-2 during development, so only C4 is held out. Speed: llama-bench tg128 (batch-1 decoding) and pp512, as same-round ratios to IQ2_XXS (Appendix K). Pre-registrations are timestamped text files, not public registry entries; Appendix K.1 lists their SHA-256 prefixes.

4.1. Main result

Table 1 gives every file of this size and Table 2 the paired margins (K1). B4 is also below the 4.6% larger default-recipe IQ2_KT (17.28). Methods that need their own runtime can be lower (Section 4.2).

What was pre-registered. On Qwen3-4B, B4 is post hoc: it is the last of a chain of builds, each defined after the one

before and all evaluated on WikiText-2 during development (Appendix D). Its one pre-registered rule there, written before the build, was a paired win over our previous build with a 95% interval excluding zero; it holds (-0.066 ± 0.009 (70/73 windows) on WikiText-2; C4: K7) at 1.8 MB fewer bytes and equal speed, while its point prediction (17.45 [17.00, 17.85]) missed: the gain was larger. The $c_j = 1$ control of K7 was pre-registered (Appendix E.4). Its margins over IQ2_KT are post hoc, although the four challenger IQ2_KT recipes were registered with predictions before they were quantized. On Llama-3.1-8B and Qwen3-14B, B4 was pre-registered against IQ2_KT; it wins on the first and loses on the second (Section 4.3).

KL divergence and zero-shot accuracy. B4’s 72-chunk KL divergence is below that of all six IQ2_KT files of this size (0.852–0.893, the C4-calibrated one highest at 0.893 ± 0.010 , 0.307 ± 0.013 above B4; top-1 agreement 63.7% against 71.3%) and of IQ2_XXS (Table 1). Zero-shot (Table 3), B4 averages 50.83 (80% prediction 49.9 [48.7, 51.2]; ARC-Easy above its interval), 4.31 ± 1.15 points above the C4-calibrated IQ2_KT (46.52; predicted 46.4 [45.3, 47.5]), higher on all five tasks, by about two standard errors or more on four (unpaired; ARC-Challenge $z = 0.68$; WinoGrande and ARC-Easy survive a Bonferroni correction); IQ2_XXS averages 41.9.

4.2. Speed and where we lose

Speed (C4, C5). Absolute rates vary by up to 46% across sessions, so we compare only within one. In the exclu-

Small-State Coset Trellis Coding at IQ2_XXS Size

Table 2. The claims of this paper (K1–K8), each with its evidence and status. Δ NLL: paired, nats per token, 95% interval (windows won); B4 minus the other file, so negative favours B4, except K6 (memoryless file minus B4). Speed: same-round ratio to IQ2_XXS. Rule: a decision rule written before the run. Evidence and caveats: the section in the last column.

Claim	Evidence	Status
K1 Qwen3-4B: lowest perplexity (both texts) and KL of the llama.cpp-format files at IQ2_XXS size	vs. C4-calibrated IQ2_KT -0.087 ± 0.011 (71/73, corrected) (WikiText-2), -0.108 ± 0.008 (73/73, corrected) (C4); vs. tuned IQ2_KT -0.094 ± 0.010 (71/73, corrected), -0.115 ± 0.008 (73/73, corrected); vs. IQ2_XXS -0.542 ± 0.020 (73/73 windows), -0.425 ± 0.013 (73/73, corrected)	post hoc; the one rule (vs. our previous build) holds; §4.1
K2 Llama-3.1-8B: beats IQ2_KT	-0.103 ± 0.007 (BOS at every window, 70/70); -0.094 ± 0.005 (strided, 70/70)	pre-registered rule holds; IQ2_KT from Q8_0; §4.3
K3 Qwen3-14B: loses to IQ2_KT	$+0.018 \pm 0.008$ (12/73) on WikiText-2; tie on C4 ($+0.001 \pm 0.006$)	pre-registered rule: loss; §4.3
K4 Qwen3-4B: batch-1 decoding at least as fast as IQ2_XXS	tg128 1.096 [1.034, 1.158]×	edge not established; one GPU; §4.2
K5 Prompts and batched decoding are slower	pp512 0.741 [0.710, 0.772]×; 2–8 sequences 0.18–0.34×	measured loss; §4.2
K6 The trellis memory: about 80% of the margin over tuned IQ2_KT, 14% of the gain over IQ2_XXS	removal costs $+0.076$ [$+0.062$, $+0.090$] (70/73) at identical blob bytes	rule: load-bearing, not the main source; 80% post hoc; §5.1
K7 Bound-derived column scale	vs. our previous build -0.068 ± 0.007 (73/73, corrected) (C4; several settings changed together); $c_j = 1$ control ties on WikiText-2, $c_j = 1$ minus B4 $+0.048 \pm 0.005$ on C4	pre-registered rule and control; exponent not swept; §4.1
K8 A weight-error proxy misranks allocations	flat arm’s proxy error $1.27\times$ the per-tile arm’s, yet PPL 17.79 (flat) against 18.23; flat minus per-tile -0.0245 ± 0.0081 (51/73)	rule fixed before the run; one build per arm; §5.4

Table 3. B4 has the highest zero-shot accuracy of the 2-bit files shown on every task. Zero-shot accuracy (%) of the native files on Qwen3-4B (HellaSwag (Zellers et al., 2019) 1000 items, WinoGrande (Sakaguchi et al., 2020) 1267, ARC-Easy and ARC-Challenge (Clark et al., 2018) validation splits, 570 and 299, full MMLU (Hendrycks et al., 2021) 14,042); 0-shot continuation scores, far below 5-shot MMLU numbers (Appendix E.2). Q8_0 stands in for fp16. Last row: standard errors of a typical 2-bit file (our per-tile build), HellaSwag $\pm$ half its 95% interval, rounded. All rows, earlier builds included: Table 8. ^eRerun with `--no-escape` (ik_llama.cpp had read escape sequences in the binary task file).

	HS	WG	ARC-E	ARC-C	MMLU
Q8_0	65.8	65.51	66.14	45.15	38.74
Ours (B4)	57.7	63.77	60.88	37.79	34.01
IQ2_KT, knapsack	53.5	58.80	50.53	34.45	32.69 ^e
IQ2_KT, C4-calibrated	53.3	57.93	53.33	35.12	32.91 ^e
IQ2_XXS	47.8	57.46	43.86	29.10	31.07
IQ2_XS (larger file)	54.1	59.35	57.72	34.11	33.44
s.e. (typical)	± 3.1	± 1.37	± 2.06	± 2.79	± 0.40

sive session with all files (Table 4; all 15 pre-registered ratios inside their intervals) B4 also decodes at 1.102 [1.050 , 1.154]× the C4-calibrated IQ2_KT (tg128, 213 tokens/s); in a B4-only session of 10 exclusive rounds it decodes at 1.109 [1.099 , 1.119]× IQ2_XXS and 1.106 [1.094 , 1.117]× the knapsack IQ2_KT (Appendix E.4). We do not treat this edge as established. At one sequence, llama-batched-bench gives only 1.020 [0.981 , 1.060], and in the same session two of our earlier builds, which differ in file layout and

Table 4. B4 decodes at batch 1 (tg128) at least as fast as IQ2_XXS and IQ2_KT; it is slower in prompt processing (pp512), and batched decoding of 2 to 8 sequences is competitive only with the post hoc multi-column kernel. One exclusive bench session with all files, one RTX 5070 Ti Laptop GPU: 10 rotated rounds, 229 runs, none failed, no other GPU process; mean same-round ratio to mainline IQ2_XXS, 95% t -interval (depth 0); batched: decoding rate of llama-batched-bench (128-token prompt, 128 tokens per sequence), mean ratio. IQ2_XXS medians: 197.1 and 7561 tokens/s. IQ2_KT: ik_llama.cpp binary. Earlier builds in the same session: Table 9.

Format	tg128	pp512	2 / 4 / 8 seq.
B4 (batch-1 GEMV)	1.096 [1.034, 1.158]	0.741 [0.710, 0.772]	0.18 / 0.26 / 0.34
B4, multi-column (post hoc)	1.094 [1.053, 1.134]	0.750 [0.732, 0.769]	0.92 / 1.06 / 1.09
IQ2_KT, C4-calibrated	0.995 [0.952, 1.038]	0.949 [0.906, 0.992]	0.66 / 0.73 / 0.73
IQ2_KT, knapsack	0.983 [0.946, 1.020]	0.923 [0.877, 0.969]	–
Q4_K_M (larger file)	0.788 [0.738, 0.838]	0.890 [0.833, 0.946]	–

two tensors but not in CUDA code, decode at 0.977 and $1.069\times$ IQ2_XXS: a step of about 9% with no known cause, as large as B4’s edge (Appendix D). The supported claim is batch-1 decoding at least as fast as IQ2_XXS on this GPU. Prompts are slower because the stock Q8_0 multiply on the decoded slab is slower than the IQ2_XXS multiply; IQ2_KT prefills about $1.3\times$ faster than B4. Batched decoding is a clear weakness of the shipped runtime: with 2 to 8 sequences every micro-batch decodes each matrix to the slab (micro-batch sweep: Table 13). The post hoc multi-column kernel decodes each weight once for all sequences (Table 4; 1.39 – $1.50\times$ IQ2_KT).

Other 2-bit methods. B4 ($1.336 \times$ fp16, about 2.14 bits per weight on the linears) beats a third-party LLVQ checkpoint (2.16 bits; $1.380 \times$ in llama.cpp) and QTIP 3INST as run by the LLVQ authors (2.0 bits, their harness, $1.373 \times$), but loses to the LLVQ paper’s 2-gain-bit row ($1.252 \times$) and to Q2_K (15.01, 34% more bytes); AutoRound (Cheng et al., 2024) w2g128 (30.26) is worse than all of our builds (Appendix E.3). WaterSIC (Lifar et al., 2026) entropy-codes integer weights and needs its own decoder (no llama.cpp kernel); it reports 11.37 on Qwen3-8B at 2.125 bits ($1.169 \times$ its bf16 9.73; calibrated on WikiText-2 train, 2048-token context; their Table 2) but no row for our models, so we cite it and do not run it.

4.3. Two more models (pre-registered)

Llama-3.1-8B: a pre-registered win (K2). Our first build there, an earlier flat recipe, failed (Appendix F). We then pre-registered B4, the Qwen recipe with the same build engine and evaluation protocol and one calibration change (every second calibration sequence starts mid-text without a BOS, as the evaluation windows after the first do), with a prediction (9.85 [9.50, 10.25]) and the rule that B4 wins only if its paired Δ NLL against IQ2_KT at no more bytes is negative with a 95% interval excluding zero. The rule was written after B4’s Qwen3-4B result and after an intermediate Llama build with the column scale in layers 0–1 only had been evaluated on the same windows (10.38; Appendix F). It holds (Table 1): B4 is better than IQ2_KT and IQ2_XXS on 70 of 70 windows each (intervals corrected for lag-1 autocorrelation), at 1,139,616 fewer bytes than IQ2_KT; the prediction was too pessimistic. Three caveats bound the result. (i) IQ2_KT was quantized from the Q8_0, which on Qwen3-4B we estimate costs it about 0.026 nats against a bf16 source, well below the margin. (ii) In llama.cpp’s strided mode every window after the first starts with an ordinary token instead of a BOS, the regime in which our earlier flat build failed. A second pre-registered test with a BOS at every window (same rule, lag-1 corrected interval) gives 8.44 against 9.36 on WikiText-2 (K2), and 13.42 against 15.56, -0.148 ± 0.016 on C4, better on 70 of 70 and 70 of 71 windows (predicted $-0.07 [-0.10, -0.03]$ and $-0.06 [-0.10, -0.02]$; both missed low). (iii) No KL divergence, zero-shot or speed was measured.

Qwen3-14B: a pre-registered loss (K3). With B4’s recipe unchanged, all three files made from Q8_0 and the rule that B4 wins (loses) iff its paired WikiText-2 Δ NLL against IQ2_KT at no more bytes is negative (positive) with a 95% interval excluding zero, B4 loses to IQ2_KT (Table 1). On held-out C4 the two tie, and B4 beats IQ2_XXS on both texts ($-0.097, -0.091$). The comparison is at equal file bytes, not equal linear bytes (Appendix G).

Table 5. The coset trellis is 1.2–1.3 dB above IQ2_XXS, and about 1.8 dB from the bound remains at every state count. Distance to $D(R) = 2^{-2R}$ at 2.0625 bits per weight (iid $N(0, 1)$, 4 seeds of 2^{20} symbols, all bits counted: eight index bits per four weights plus one fp16 scale per 256; bound 12.42 dB). SNR and gap in dB. Enc.: distance evaluations per weight; Dec.: integer operations per weight for the trellis label; tuned: best generator and retrained table. Published rows: Table 1 of Tseng et al. (2024b), iid Gaussian at 2.0 bits per weight, gap to the 2.0-bit bound (12.04 dB); not rerun.

Code	SNR	Gap	Gap, tuned	Enc.	Dec.
IQ2_XXS (llama.cpp)	9.28	3.14	–	–	–
Scalar Lloyd–Max	9.33	3.09	–	–	–
4-d table, no trellis	10.10	2.32	–	64	0
Coset trellis, $S = 4$	10.51	1.91	1.84	256	2.5
Coset trellis, $S = 8$ (shipped)	10.54	1.88	1.79	512	2.5
Coset trellis, $S = 16$	10.55	1.87	1.76	1024	2.5
Coset trellis, $S = 32$	10.52	1.90	1.74	2048	2.5
QTIP, published, 2.0 bpw	11.61	0.43	–	–	–
QuIP# E8P, published, 2.0 bpw	10.51	1.53	–	–	–

5. Analysis

5.1. Where B4’s margin comes from

To measure what the Pingala-8 trellis contributes inside B4, we rebuilt B4 with only the trellis memory removed (one-state greedy LDLQ path, zero coset vector, our memoryless 256-point four-dimensional table) at identical blob bytes, with a prediction and a rule written before the build (Appendix E.4). Removal costs $+0.076 [+0.062, +0.090]$ nats (K6, lag-1 corrected), more than we predicted ($+0.045 [+0.025, +0.075]$, 90%). Against IQ2_KT the trellis is the main term: the memoryless file keeps only -0.018 ± 0.017 (45 of 73 windows) of B4’s margin over the tuned IQ2_KT, so the trellis supplies about 80% of that margin, at no extra bits; this share is a post hoc reading. By the rule written before the build (main source only above 15% of the gain over IQ2_XXS), the trellis is load-bearing but not the main source of B4’s gain. Against IQ2_XXS it is a minor term, 14% of B4’s gain (the memoryless file is still -0.466 below IQ2_XXS); the rest comes from the rest of the pipeline (rotation, LDLQ, the column scale and the 256-point table; not separated). A 16-state per-tile carrier build agrees: removing the trellis at the same container bytes raises the perplexity ratio from 1.508 to 1.607, and the trellis supplies 0.063 of a 0.421-nat gain (15%; the front end without it 0.357, 85%). So the front end brings a memoryless table to about the level of the tuned IQ2_KT, and the trellis buys the margin over it. Against the C4-calibrated IQ2_KT the memoryless file keeps -0.011 ± 0.016 (41 of 73 windows), so the trellis supplies about 88% of that margin.

5.2. Distance to the rate-distortion bound

On a Gaussian source (Table 5) the trellis adds 0.41 to 0.45 dB to the memoryless four-dimensional table (about 25% lower mean squared error than IQ2_XXS at identical bits). With tuned tables the gap falls only slightly from

4 to 32 states, so more trellis memory does not close it (longer blocks were not tested). On three Hadamard-rotated Qwen3-4B tensors of layer 10 the ordering is the same, with a near-tie on `attn_q` (margins over IQ2_XXS of 1.29, 1.0 and 0.08 dB on `ffn_up`, `ffn_down` and `attn_q`). The side information of our first, per-tile build (0.0146 bits per weight, Table 21; less in B4’s layout, which drops tile words and permutations) would add about 0.09 dB to each gap. Charged at 6.02 dB per bit for its extra 0.0625 bits, our code is about 0.35 dB below QuIP#’s E8P codebook and 1.45 dB below QTIP’s long trellis; the small state count is the price of a decoder that runs at llama.cpp generation speed.

5.3. State count: why 8 states ship

On the full model the 16-state trellis is 1.7% better in perplexity than the 8-state one (18.78 against 19.10 on 8-bit carriers of our first, per-tile build), more than the 0.013–0.033 dB source-level difference predicts; 4 states give 20.14 (a later build script; Appendix I). Neither the shipped tables (retrained 8-state tables: within 0.05% in weight error) nor the kernel favours 8 states (only the label function changes; 4-, 8- and 16-state kernels ran within 1.7%, inside round noise). We ship 8 states because only the 8-state code has a native build: every native file in this paper, B4 included, and with it every measured size, speed and bit-equal decode, exists only for 8 states, while the 16-state result is a carrier-based control. A native 16-state B4 is the next build to make, not a result we have.

5.4. A weight-error proxy misranks allocations

At equal bytes, per-tile rate classes lower the held-out weight error (one class per tensor has $1.27\times$ the error of the per-tile arm) yet raise perplexity (K8; flat better on 51 of 73 windows; decision rule fixed before the run; one build per arm, one model and text; Appendix D.5). So λ_j in Eq. 3 should be measured end to end, not read off a weight-error proxy (as in EvoPress; Sieberling et al., 2024); layer-wise error is a known imperfect proxy for the end loss (HIGGS, GuidedQuant and YAQA; Malinovskii et al., 2025; Kim et al., 2025; Tseng et al., 2026). This is why B4 has one class per tensor.

6. Limitations

Post hoc headline. On Qwen3-4B, B4 is the last of a chain of builds developed on WikiText-2; only C4 is held out, and the column-scale exponent was not swept. B4’s margin over the strongest IQ2_KT of this size we built (K1) may shrink with a better IQ2_KT recipe or calibration. **Speed.** Batch-1 decoding at least as fast as IQ2_XXS is the supported claim; the measured edge is the size of an unexplained step between two earlier builds (Section 4.2) and was measured on one laptop GPU. Prompt processing and batched decoding

of 2 to 8 sequences on the shipped path are slower than IQ2_XXS (K5); batch-1 decoding is the supported regime; other GPUs were not measured, and the CPU backend only in Appendix H. On the batch-1 path B4’s perplexity is 0.26% above the batched path (Appendix E.4; earlier builds: Appendix L.3). The multi-column GEMV failed its perplexity gate and is post hoc (Appendix E.4). **Scope.** Three models: Qwen3-4B; Llama-3.1-8B with perplexity only and an IQ2_KT made from Q8_0; Qwen3-14B with perplexity only, all files from Q8_0, where B4 loses to IQ2_KT (K3); no other family. Nothing above 14B was evaluated: a 70B file at IQ2_XXS size does not fit in the 12 GB of VRAM of our only GPU. Allocation controls are one build per arm, on one model and one text. IQ2_S, IQ2_M and IQ3_XXS fail on our GPU and are excluded (Appendix E.1); an unpatched build of the same commit and toolchain gives the same failure (the CPU backend does not), so the cause is upstream or the toolchain (nvcc 13.2 with cuBLAS 12), which we cannot separate here. EXL3, Proteus, CurveTQ, Tetra and XOR-Trellis were not run, and QTIP, QuIP# and AQLM were not rerun on Qwen3-4B.

7. Conclusion

Ungerboeck’s 8-state coset code, used as a 4-dimensional TCVC, fits in a llama.cpp file of IQ2_XXS size and, on Qwen3-4B, decodes at batch 1 at least as fast as IQ2_XXS. At no more bytes it beats IQ2_XXS on three models and IQ2_KT on two (Qwen3-4B, post hoc; Llama-3.1-8B, pre-registered), and it loses to IQ2_KT on Qwen3-14B, pre-registered. Equal-bytes rebuilds locate the gain: the trellis memory supplies most of the margin over a tuned IQ2_KT but little of the gain over IQ2_XXS. On Qwen3, the margin over IQ2_XXS shrinks from 0.54 nats at 4B to 0.10 at 14B, and prompts are slower than with IQ2_XXS.

Impact Statement

This paper presents work whose goal is to advance the field of machine learning, specifically the storage and memory cost of running language models. A 2-bit weight file lowers the hardware needed to run open-weight models locally (calculated, not run: about 2.4 GB for an 8B model, and a 70B model on one 24 GiB card for a single 8K-token sequence with 0.6% of usable memory to spare, if a 1 GiB reserve covers the 249.6 MB decode slab and compute buffers, untested; Appendix L.4), which helps users with small budgets, poor connectivity or privacy needs, and reduces memory traffic per generated token (0.13 to 0.15 of fp16, calculated). It is a trade of quality for memory, not a free gain, and we claim no net energy saving: our prompt path uses $2.05\times$ the energy per prompt token of IQ2_XXS (median of 3 rounds, 1.70–4.10; measured from nvidia-smi board power), encoding is a one-off GPU cost, and cheaper inference may

raise total use. We release a format and a runtime, not new model capabilities; misuse considerations are those of the underlying open-weight models, which easier local deployment makes slightly cheaper to run without provider-side safeguards. Quantization can change refusal, bias and hallucination behaviour in ways perplexity does not show; we did not evaluate these, and deployers should evaluate quantized models for their own safety requirements.

Reproducibility. The runtime is a patch set on llama.cpp commit 7ac59a6e3ad851cd41af00f678effab0598ba9a8. Code: <https://github.com/Genrius-AI/pingala8-llm> (Apache-2.0). Models: <https://huggingface.co/Genrius-AI/Qwen3-4B-Pingala8-2bit> (Apache-2.0, inherited from Qwen3-4B) and <https://huggingface.co/Genrius-AI/Llama-3.1-8B-Pingala8-2bit> (Llama 3.1 Community License; built with Llama). Appendix K lists every setting.

References

- Anonymous. Computed permutation trellis coding. Submission to ICLR 2027, OpenReview id 1W5aP7VZOd, 2026.
- Ben David, N., Meir, O., and Ordentlich, O. OSFP4: Joint optimization of diagonal smoothing and block scales for NVFP4 quantization. *arXiv preprint arXiv:2610.08231*, 2026.
- Brown, C. P. Sanskrit prosody and numerical symbols explained. London, 1869.
- Chee, J., Cai, Y., Kuleshov, V., and De Sa, C. QuIP: 2-bit quantization of large language models with guarantees. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2307.13304.
- Cheng, W., Zhang, W., Shen, H., Cai, Y., He, X., Lv, K., and Liu, Y. Optimize weight rounding via signed gradient descent for the quantization of LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 11332–11350, 2024. arXiv:2309.05516.
- Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.
- Dettmers, T., Svirschevski, R., Egiazarian, V., Kuznedelev, D., Frantar, E., Ashkboos, S., Borzunov, A., Hoefler, T., and Alistarh, D. SpQR: A sparse-quantized representation for near-lossless LLM weight compression. In *International Conference on Learning Representations*, 2024. arXiv:2306.03078.
- Ding, G., Wang, Z., Li, R., and Liu, J. CurveTQ: Rotation-free trellis quantization of LLM weights via curvature-weighted search, 2026. URL <https://arxiv.org/abs/2610.09212>. arXiv:2610.09212.
- Dong, Z., Yao, Z., Gholami, A., Mahoney, M., and Keutzer, K. HAWQ: Hessian AWARE quantization of neural networks with mixed-precision, 2019. arXiv:1905.03696.
- Dutta, A., Krishnan, S., Kwatra, N., and Ramjee, R. Accuracy is not all you need, 2024. arXiv:2407.09141.
- Egiazarian, V., Panferov, A., Kuznedelev, D., Frantar, E., Babenko, A., and Alistarh, D. Extreme compression of large language models via additive quantization. In *International Conference on Machine Learning*, 2024. arXiv:2401.06118.
- Fischer, T. R., Marcellin, M. W., and Wang, M. Trellis coded vector quantization. *IEEE Transactions on Information Theory*, 37(6):1551–1566, 1991.
- Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023. arXiv:2210.17323.
- Frantar, E., Castro, R. L., Chen, J., Hoefler, T., and Alistarh, D. MARLIN: Mixed-precision auto-regressive parallel inference on large language models, 2024. arXiv:2408.11743.
- Gerganov, G. and llama.cpp contributors. llama.cpp: LLM inference in C/C++. <https://github.com/ggml-org/llama.cpp>, 2023.
- Guo, H., Brandon, W., Cholakov, R., Ragan-Kelley, J., Xing, E. P., and Kim, Y. Fast matrix multiplications for lookup table-quantized LLMs, 2024. arXiv:2407.10960.
- Haase, P., Schwarz, H., Kirchhoffer, H., Wiedemann, S., Marinc, T., Marban, A., Müller, K., Samek, W., Marpe, D., and Wiegand, T. Dependent scalar quantization for neural network compression. In *IEEE International Conference on Image Processing (ICIP)*, pp. 36–40, 2020. doi: 10.1109/ICIP40778.2020.9190955.
- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. arXiv:2009.03300.
- Huang, W., Qin, H., Liu, Y., Li, Y., Liu, Q., Liu, X., Benini, L., Magno, M., Zhang, S., and Qi, X. SliM-LLM: Saliency-driven mixed-precision quantization for large language models, 2024. arXiv:2405.14917.

- Johnson, G. TurboPress: rotation-aware equilibration and trellis-coded post-training weight quantization. <https://github.com/godsonj64/turbopress>, 2026.
- Kak, S. Yāmātārājabhānasalagām: An interesting combinatoric sūtra. *Indian Journal of History of Science*, 35: 123–127, 2000.
- Kawrakow, I. Trellis quantization (IQ2_KT, IQ3_KT, IQ4_KT), ik_llama.cpp pull request 113, 2024a. URL https://github.com/ikawrakow/ik_llama.cpp/pull/113. Opened 2024-11-15.
- Kawrakow, I. SOTA 2-bit quants (llama.cpp pull request 4773, IQ2_XXS). <https://github.com/ggml-org/llama.cpp/pull/4773>, 2024b.
- Kawrakow, I. New IQ4_KT trellis implementation (integer trellis), ik_llama.cpp pull request 505, 2025a. URL https://github.com/ikawrakow/ik_llama.cpp/pull/505. Opened 2025-06-08.
- Kawrakow, I. New IQ2_KT, ik_llama.cpp pull request 511, 2025b. URL https://github.com/ikawrakow/ik_llama.cpp/pull/511. Opened 2025-06-09, closed in favour of pull request 529.
- Kawrakow, I. New IQ2_KT, IQ3_KT and IQ4_KT, V2, ik_llama.cpp pull request 529, 2025c. URL https://github.com/ikawrakow/ik_llama.cpp/pull/529. Merged 2025-06-18.
- Kawrakow, I. Adding IQ1_KT, 1.75 bpw SOTA quants, ik_llama.cpp pull request 616, 2025d. URL https://github.com/ikawrakow/ik_llama.cpp/pull/616. Merged 2025-07-20.
- Kawrakow, I. and ik_llama.cpp contributors. ik_llama.cpp: a llama.cpp fork with additional quantization types and improved performance, 2024. URL https://github.com/ikawrakow/ik_llama.cpp. Accessed 2026-10-14.
- Kim, J., El Halabi, M., Park, W., Schaefer, C. J., Lee, D., Park, Y., Lee, J. W., and Song, H. O. GuidedQuant: Large language model quantization via exploiting end loss guidance. In *International Conference on Machine Learning*, volume 267 of *PMLR*, pp. 30011–30037, 2025. arXiv:2505.07004.
- Kim, S., Hooper, C., Gholami, A., Dong, Z., Li, X., Shen, S., Mahoney, M. W., and Keutzer, K. SqueezeLLM: Dense-and-sparse quantization, 2023. arXiv:2306.07629.
- Kirchhoffer, H., Haase, P., Samek, W., Müller, K., Rezadegan-Tavakoli, H., Cricri, F., et al. Overview of the neural network compression and representation (NNR) standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(5):3203–3216, 2022. doi: 10.1109/TCSVT.2021.3095970.
- Knuth, D. E. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Addison-Wesley, 2011.
- Kurtic, E., Marques, A., Pandit, S., Kurtz, M., and Alistarh, D. “give me BF16 or give me death”? accuracy-performance trade-offs in LLM quantization, 2024. arXiv:2411.02355.
- Lee, D. and Song, H. O. Q-Palette: Fractional-bit quantizers toward optimal bit allocation for efficient LLM deployment, 2025. URL <https://arxiv.org/abs/2509.20214>. arXiv:2509.20214.
- Li, X., Chou, T., Fromm, J., Liu, Z., Pan, Y., and Fragoali, C. ScaleBITS: Scalable bitwidth search for hardware-aligned mixed-precision LLMs, 2026. URL <https://arxiv.org/abs/2602.17698>. arXiv:2602.17698.
- Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., Yu, F., Wang, W., and Gu, S. BRECQ: Pushing the limit of post-training quantization by block reconstruction, 2021. arXiv:2102.05426.
- Lifar, E., Savkin, S., Ordentlich, O., and Polyanskiy, Y. WaterSIC: Information-theoretically (near) optimal linear layer quantization. *arXiv preprint arXiv:2603.04956*, 2026.
- Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024a. arXiv:2306.00978.
- Lin, Y., Tang, H., Yang, S., Zhang, Z., Xiao, G., Gan, C., and Han, S. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving, 2024b. arXiv:2405.04532.
- Liu, Y., Wen, J., Wang, Y., Ye, S., Zhang, L. L., Cao, T., Li, C., and Yang, M. VPTQ: Extreme low-bit vector post-training quantization for large language models, 2024. arXiv:2409.17066.
- Malandrino, P.-J. Tetra: Serving Leech-lattice quantized LLMs at 2.7 bits per parameter, 2026. URL <https://arxiv.org/abs/2609.35465>. arXiv:2609.35465.
- Malinovskii, V., Mazur, D., Ilin, I., Kuznedelev, D., Burlachenko, K., Yi, K., Alistarh, D., and Richtárik, P. PV-Tuning: Beyond straight-through estimation for extreme LLM compression. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2405.14852.

- Malinovskii, V., Panferov, A., Ilin, I., Guo, H., Richtárik, P., and Alistarh, D. HIGGS: Pushing the limits of large language model quantization via the linearity theorem. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10857–10886, 2025. arXiv:2411.17525.
- Marcellin, M. W. and Fischer, T. R. Trellis coded quantization of memoryless and Gauss-Markov sources. *IEEE Transactions on Communications*, 38(1):82–93, 1990.
- Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models, 2016.
- Narita, Y. and Sato, I. Understanding LLM quantization through activation-guided compensation and orthogonal residuals. *arXiv preprint arXiv:2609.21450*, 2026.
- Ordentlich, O. and Polyanskiy, Y. High-rate quantized matrix multiplication II. *arXiv preprint arXiv:2605.13768*, 2026.
- Panter, P. F. and Dite, W. Quantization distortion in pulse-count modulation with nonuniform spacing of levels. *Proceedings of the IRE*, 39(1):44–48, 1951.
- Park, G., Park, B., Kim, M., Lee, S., Kim, J., Kwon, B., Kwon, S. J., Kim, B., Lee, Y., and Lee, D. LUT-GEMM: Quantized matrix multiplication based on LUTs for efficient inference in large-scale generative language models, 2022. arXiv:2206.09557.
- Pingala. *Chandahśāstram of Śrī Piṅgalanāga, with the commentary Mṛtasañjīvanī of Halāyudha Bhaṭṭa*. Number 91 in Kāvyaṃālā. Nirṇaya Sagar Press, Bombay, 3rd edition, 1938. Edited by Paṇḍit Kedāranāth of Jaipur; 3rd edition, revised with notes by Anant Yajñeśvar Śāstrī Dhupkar.
- Plofker, K. *Mathematics in India*. Princeton University Press, Princeton, NJ, 2009.
- Que, X., Elkayam, N., Pyakurel, S., Pan, S., and Gope, D. XOR-Trellis: Ultra-low-complexity dequantization and curvature-aware Hadamard-free LLM quantization, 2026.
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21 (140):1–67, 2020.
- Sakaguchi, K., Le Bras, R., Bhagavatula, C., and Choi, Y. WinoGrande: An adversarial Winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. arXiv:1907.10641.
- Sieberling, O., Kuznedelev, D., Kurtic, E., and Alistarh, D. EvoPress: Accurate dynamic model compression via evolutionary search, 2024. arXiv:2410.14649.
- Tseng, A., Chee, J., Sun, Q., Kuleshov, V., and De Sa, C. QuIP#: Even better LLM quantization with Hadamard incoherence and lattice codebooks. In *International Conference on Machine Learning*, 2024a. arXiv:2402.04396.
- Tseng, A., Sun, Q., Hou, D., and De Sa, C. QTIP: Quantization with trellises and incoherence processing. In *Advances in Neural Information Processing Systems*, 2024b. arXiv:2406.11235.
- Tseng, A., Sun, Z., and De Sa, C. Model-preserving adaptive rounding. In *International Conference on Machine Learning*, 2026. arXiv:2505.22988.
- turboderp-org. ExLlamaV3: EXL3 quantization, a streamlined variant of QTIP, 2025. URL <https://github.com/turboderp-org/exllamav3>. Repository README, section “EXL3 quantization”. Accessed 2026-10-14.
- Ungerboeck, G. Channel coding with multilevel/phase signals. *IEEE Transactions on Information Theory*, 28(1): 55–67, 1982.
- van Baalen, M., Kuzmin, A., Koryakovskiy, I., Nagel, M., Couperus, P., Bastoul, C., Mahurin, E., Blankevoort, T., and Whatmough, P. GPTVQ: The blessing of dimensionality for LLM quantization, 2024. arXiv:2402.15319.
- van der Ouderaa, T. F. A., van Baalen, M., Whatmough, P., and Nagel, M. Leech lattice vector quantization for efficient LLM compression, 2026. ICML 2026 AdaptFM workshop poster; arXiv preprint 2603.11021.
- van Nooten, B. Binary numbers in Indian antiquity. *Journal of Indian Philosophy*, 21:31–50, 1993.
- Wei, J., Cao, S., Cao, T., Ma, L., Wang, L., Zhang, Y., and Yang, M. T-MAC: CPU renaissance via table lookup for low-bit LLM deployment on edge, 2024. arXiv:2407.00088.
- Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, 2023. arXiv:2211.10438.
- Yan, X., Wu, Z., Zhang, K., Yang, K., and Zhang, Y. SAGE: Pushing 2-bit vector-quantized LLMs toward real-world usability, 2026. URL <https://arxiv.org/abs/2610.14745>. arXiv:2610.14745.
- Yang, A. et al. Qwen3 technical report, 2025. arXiv:2505.09388.

Yang, Z., Li, X., Cheng, K., Liu, K., et al. Pro-teus: Lookup-free trellis-coded quantization by lattice-breaking compute codes for 2-bit LLMs. International Conference on Machine Learning (ICML) 2026, poster, 2026. URL <https://icml.cc/virtual/2026/poster/63948>.

Yuan, Z., Shang, Y., Zhou, Y., Dong, Z., Zhou, Z., Xue, C., Wu, B., Li, Z., Gu, Q., Lee, Y. J., Yan, Y., Chen, B., Sun, G., and Keutzer, K. LLM inference unveiled: Survey and rooftop model insights, 2024. arXiv:2402.16363.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. arXiv:1905.07830.

Zhang, H., Zhang, Q., Cao, Q., Fan, H., Huang, Y., Sha, K., and Feng, Y. AlignQuant: Tile-aligned mixed-precision quantization for efficient LLM generation, 2026. URL <https://arxiv.org/abs/2610.07457>. arXiv:2610.07457.

Zhou, Z., Li, X., Li, M., Zhang, H., Wang, H., Chang, W., Liu, Y., Dang, Q., Yu, D., Ma, Y., and Wang, H. CCQ: Convolutional code for extreme low-bit quantization in LLMs, 2025. arXiv:2507.07145.

Build names. The body reports one file, B4. On Qwen3-4B four builds of the same code preceded it. *Model B*, the first shipped build, gives each 32×256 tile one of 8 rate classes chosen by a weight-error proxy. The flat builds give each tensor one class: *B2*, a first size-matched flat build; *B3*, defined after B2 and pre-registered; *B3+*, B3 untrimmed in a leaner layout; and *B4*, B3+ with the derived column scale. “Our previous build” is B3+ and “our first build” or “our per-tile build” is Model B (details: Appendix D).

A. What Is Borrowed and Related Work in Detail

A.1. Trellis and TCQ quantizers for LLMs: what already exists

We state only what each work’s abstract, README or pull request states. **QTIP** (Tseng et al., 2024b) applies TCQ with a bitshift trellis and incoherence processing to LLM weights; its lookup-only or computed codes decouple codebook size from the effective dimension. **EXL3** (turboderp-org, 2025) is, in its own words, a streamlined variant of QTIP with a fused GPU Viterbi kernel; it is a separate inference library, not a llama.cpp format. **The IQ*_KT types of ik_llama.cpp** (Kawrakow & ik_llama.cpp contributors, 2024) are trellis types inside a llama.cpp fork. The first version (pull request 113 (Kawrakow, 2024a)) took QTIP’s “3INST” generator: IQ2_KT uses a 16-bit seed per group of 8 weights, blocks of 32 with a 4-bit scale, and 2.125 bits per weight, with seeds found by clustering, no Hadamard transform and no Viterbi search (194 against 208 tokens/s for IQ2_XXS on an RTX 4080, model not stated). The current version (pull request 505, superseded by the merged pull request 529 (Kawrakow, 2025a;c)) sums masked bytes of an integer generator, which the pull request says allows quantized GEMM and GEMV on the GPU and much faster CPU kernels. IQ1_KT (1.75 bits per weight, pull request 616 (Kawrakow, 2025d)) is reported to give nearly IQ2_XXS quality at about 10% smaller size on Llama-3.1-8B-Instruct. In pull request 511 (Kawrakow, 2025b), IQ2_KT on Llama-3.1-8B-Instruct at context 512 has a perplexity ratio to f32 of about 1.50 (11.0029 against 7.3210; 10.9466 with the later multiplier) and generates at 205.5 tokens/s on an RTX 4080. These models, contexts and GPUs differ from ours, so the numbers are not comparable. They do show that a trellis type at IQ2_XXS size, with CUDA generation speed close to IQ2_XXS, exists in a llama.cpp-derived code base. We make no such first claim; Table 7 gives the same-harness rows. **Proteus** (Yang et al., 2026) (ICML 2026) computes trellis values by integer arithmetic and a per-layer affine transform, with no table loads at decode time, on Llama 2 7B to 70B. Its public repository was a placeholder when we checked, so we could not run it. **CCQ** (Zhou et al., 2025) quantizes LLMs to 2.0–2.75 bits with a convolutional code, hybrid encoding, code clustering and a lookup-free bit-shift encoding space. It reports DeepSeek-V3 at 184 GB and ERNIE-4.5-300B-A47B at 89 GB and open-sources its 2-bit ERNIE model and engine. Its abstract does not mention llama.cpp; we did not run it. **Dependent quantization** (Haase et al., 2020) applies TCQ with context-modelled entropy coding of the indices to neural-network parameters, against the MPEG-7 Part 17 test model; it is a tool of ISO/IEC 15938-17:2022 (Kirchhoffer et al., 2022), where it uses an 8-state trellis. It is not an LLM method, but it means that trellis quantization of network weights is not new. **Tetra** (Malandrino, 2026) uses a 64-state trellis of the Golay code on the Leech lattice in its own engine. Its Qwen3-4B file (2.73 bits per parameter, larger than ours) scores 63.37 on full MMLU, 23.6 points above llama.cpp IQ2_XXS (2.48 bits per parameter), on an L40S GPU. That MMLU is 5-shot from the logits of the four answer letters (its Section 6.1), so it is not comparable with our 0-shot continuation scores (Q8_0: 38.74, Table 16). **XOR-Trellis** (Que et al., 2026) gives a low-complexity state-to-value mapping and a curvature-aware path objective in the original coordinates, with no Hadamard incoherence. **CurveTQ** (Ding et al., 2026) removes the rotation, puts the diagonal of the Hessian’s LDL factor into the Viterbi branch metric, stores a start state per block, and reports 1 to 3 points higher mean downstream accuracy than QTIP and Proteus on three 4B to 8B Instruct models at two bits, with the fastest decoder of the three. As far as their abstracts state, none of these targets a llama.cpp file format. Their results are on other models and harnesses; we do not rank them against ours and did not rerun them. **Search.** On 2026-10-14 we searched arXiv, GitHub and the web for a small-state coset TCQ with per-tile rate classes in a fixed-size llama.cpp file and found none (about 35 sources checked, about 16 read in depth). This is a negative search result, not a proof of absence. A search of the ggml-org/llama.cpp pull requests for “trellis” that day returned no trellis quantization type.

A.2. Bit allocation under a storage budget

Q-Palette (Lee & Song, 2025) derives an information-theoretic bit allocation over fractional-bit quantizers, from trellis-coded to scalar, with CUDA kernels. ScaleBITS (Li et al., 2026) allocates bitwidths greedily over hardware-aligned blocks under a memory budget. AlignQuant (Zhang et al., 2026) allocates precision per GPU-compatible two-dimensional tile by prefill and decode sensitivity, with INT8 kernels. SAGE (Yan et al., 2026) splits a fixed budget between the direction codebook and the radius precision of each matrix in a 2-bit vector quantizer. llama.cpp recipes already mix types across tensors (IQ2_XXS keeps attn_v as Q4_K and four ffn_down as Q2_K). Our classes are the 8 rate classes of one trellis code, with one byte count per class and a class uniform across a warp. We do not compare our rule with those above, which differ in objective, tile shape and kernels. HAWQ (Dong et al., 2019) and BRECQ (Li et al., 2021) could supply the λ_j of Eq. 3, which we set to 1. Slim-LLM (Huang et al., 2024) reports a gain from per-group salience allocation, unlike our equal-bytes result (Section 5.4). EvoPress (Sieberling et al., 2024) finds that per-layer error estimates assume an inter-layer independence that does not hold, and searches allocations end to end.

A.3. Kernels, vector codes and evaluation

Our batch-1 GEMV tabulates activation–codeword dot products and adds table entries, following LUT-GEMM (Park et al., 2022), T-MAC (Wei et al., 2024) and FLUTE (Guo et al., 2024). MARLIN (Frantar et al., 2024) keeps 4-bit kernels near their peak speed-up up to batch 16–32, and QServe (Lin et al., 2024b) reports 20–90% dequantization overhead in large-batch serving. Our slab path is far slower than IQ2_XXS at 2 to 32 tokens per micro-batch ($0.13\text{--}0.35\times$, Table 13); large-batch serving was not measured. Single-batch generation is memory-bound (Kim et al., 2023; Yuan et al., 2024); both 2-bit files generate far below the bandwidth bound on our GPU (Table 22). GPTVQ (van Baalen et al., 2024) and VPTQ (Liu et al., 2024) are Hessian-guided vector quantizers with their own decoders; neither targets a llama.cpp file, and we did not run them. Compressed models of similar accuracy can change many answers (Dutta et al., 2024), which is why we report KL divergence beside perplexity; Kurtic et al. (2024) evaluate 8- and 4-bit formats across a model family.

B. Pingala’s Ganas and a Note on the History of Mathematics

Pingala’s Chandahsutra (c. 3rd–2nd century BCE; edition with Halayudha’s commentary in Pingala (1938)) opens by naming eight trisyllabic units, the ganas. Sutras 1.1–1.8 list *ma* (GGG), *ya* (LGG), *ra* (GLG), *sa* (LLG), *ta* (GGL), *ja* (LGL), *bha* (GLL) and *na* (LLL), with G a heavy (guru) and L a light (laghu) syllable; sutra 1.9 defines the symbols. The text gives a set of eight named triples and no transitions between them. Chapter 8 gives the *prastara*, an ordered enumeration of all G/L sequences, and the rules *nashta* and *uddishta* that map between a sequence and its position. Read with Halayudha’s 10th-century commentary (the *Mrtasanjivani*), these rules are equivalent to binary place value (van Nooten, 1993; Plofker, 2009).

Write $\mathcal{G} = \{0, 1\}^3$ with L = 1, G = 0 and the first syllable as the least significant bit: *ma* is 000 and *na* 111, and the sutra order is 0, 1, . . . , 7. Pingala’s *prastara* (Chandahsutra 8.20–8.23) lists the 8 ganas in binary counting order, which is why we use them as the trellis states. A shift-register state set $\{0, 1\}^\nu$ is the *prastara* of ν syllables ($\nu = 2, 3, 4, 5$: 4, 8, 16, 32 states; the ganas are $\nu = 3$).

Three objects are often merged. (1) The ganas, a set of eight named triples. (2) The *prastara* with *nashta* and *uddishta*, an enumeration equivalent to binary counting. (3) The cyclic mnemonic *ya-maa-taa-raa-ja-bhaa-na-sa-la-gaa*, whose sliding 3-syllable windows visit every gana once: a de Bruijn sequence $B(2, 3)$. (3) is not in the sutra text; Kak (2000) traces it only to Brown (1869), and Knuth (2011) calls its origin obscure. A shift-register trellis on 8 states needs (3) or an equivalent modern construction; it does not follow from (1) or (2). Here the state set is (1), indexed by (2), and the edges and labels are Ungerboeck’s (Ungerboeck, 1982). The count 8 is not load-bearing either: 16 states are better at the same kernel cost (Section 5.3). The ancient text contributes the name and the motivation of the 8-state arm, not an optimum. Our search for Pingala-related prior work did not cover Sanskrit or regional-language journals.

C. Method and Runtime Details

Viterbi recursion. With $J_0(s) = 0$ for the reset state and $+\infty$ otherwise,

$$J_t(s') = \min_{s \rightarrow s'} \left[J_{t-1}(s) + \min_{i \in \mathcal{C}_{z_0(s)} : \text{msb}(i) = z_1(s, s')} d_t(x_t, \sigma \mathcal{C}_{z_0(s)}[i]) \right], \quad (4)$$

over the two predecessors of each of the 8 states. A step evaluates 8×2 branches, each a nearest-neighbour search over the 128 codewords with the required index MSB: 2048 4-dimensional distances, or 512 per weight ($O(n S |\mathcal{C}|/4)$ per row with $S = 8$, $|\mathcal{C}| = 256$), and $O(ST)$ survivor memory. The Viterbi part is linear in the parameter count and rows are independent. The Hessian part is not: Cholesky costs $O(n^3)$ per tensor and error feedback $O(m n^2)$. The whole build of 3.63×10^9 weights took 4623 s on one laptop GPU, 2768 s of it in the quantiser (70B and 405B: Appendix L). The state is reset every 32 symbols, so the start state costs no bits.

Decoder cost. One symbol: recover u_t from z_1 and the known state (two XORs), shift the state, read one table entry. For the GEMV the table holds the 256 dot products $\langle A_k, a \rangle$ of activation chunk a with the codebook, so a base-class symbol needs one table read and one add, plus $\langle c, a \rangle$ for the coset from subset-sum tables and integer work for state, index and sign. On sm_120 this is 278 SASS instructions per 32 class-3 symbols, 48 of them shared-memory loads (Appendix L.3): $O(1)$ per weight, independent of m , n and model size.

Rate classes. Per 4 weights, stage 1 is a 4-, 6- or 8-bit index into A ; classes at 2.5–6 bits per weight add residual stages of 2, 4 or 8 bits, so the value is $s_1 2^{\text{SH}_1} + s_2 2^{\text{SH}_2} + s_3$. Each tile stores one 32-bit word (payload offset and class). In Model B the 4 most salient tile-blocks per tensor are forced to class ≥ 6 , which keeps $\max |W_q| / \max |W| \geq 0.9$ on all 252 linear tensors, and a sparse fp32 delta (0.2 MB) corrects 27,437 clamped outliers. In preliminary experiments fixed-rate codes without rate allocation lost $1.8\text{--}3.1\times$ in importance-weighted distortion against entropy-coded scalar quantization, which motivated the per-tile classes; end to end they do not pay (Section 5.4).

Bit accounting. *Linear bpw* is the stored bytes of the 252 decoder linears (indices, scales, and for Model B the row permutations, column prescales and clamp deltas) times 8, over their 3,633,315,840 weights (Model B: 967.4 MB of GGUF tensor-table bytes, 2.130 bits per weight; kernel blob alone 2.1205). *All-tensor bpw* is the file size times 8 over all 4,022,468,096 parameters, so it includes the token embedding (Q5_K in the IQ2_XXS mix and in Model B), the norms and about 6 MB of GGUF metadata. Allocation arms are compared by *linear-blob bytes*, the logged container blobs of the 252 linears (966.3 MB shipped, 972.0 MB for the flat and mirror arms), not GGUF tensor-table bytes.

Token generation kernel. Activations are int8 with one scale per 128; per slice the kernel builds the chunk–codebook dot-product table in shared memory (32 KB per thread block) and reads it without bank conflicts (layout: supplementary material, Sec. SA.3).

Prompt processing (PPC2). For batches above one, a pre-pass kernel (K1) decodes each matrix once per micro-batch into an 8-bit slab in the Q8_0 layout; the stock llama.cpp Q8_0 MMQ kernel then multiplies. Per micro-batch K1 takes about 25 ms and the Q8_0 multiply about 51 ms, more than the IQ2_XXS multiplies (about 46 ms). Stock Q8_0 MMQ alone ran at $0.95\text{--}1.00\times$ IQ2_XXS, so this route cannot reach $1.00\times$ even with a free decode. These phase times come from instrumented profiling (supplementary material, Sec. SA.3), are not additive with the llama-bench rate (88.6 ms per 512-token micro-batch), and only rank the costs. Batched decoding of several sequences also pays K1 once per step (B4: Appendix E.4, with the multi-column kernel that removes it). Peak memory during prompt processing is 56 MiB higher at depth 0 and 60 MiB at depth 4096. The CUDA patch touches 12 source files (+2954 and -102 lines).

D. Build History and Controls

This appendix is in the supplementary material under the same letter.

E. Additional Results on Qwen3-4B

This appendix is in the supplementary material under the same letter.

E.4. B4: a bound-derived column scale

B3+ prescales input column j by $c_j = \sqrt{\max(s_j/\bar{s}, 10^{-2})}$, with s_j the importance-matrix entry. We derive the scale instead. Model the Hessian as diagonal ($H_{jj} \propto s_j$) and let $E_j = \sum_r W_{rj}^2$ be the column energy. After the 256-point rotation the scaled weights of a block b are close to i.i.d., so the coding error per scaled weight is proportional to the block’s mean energy $\frac{1}{B} \sum_{j \in b} c_j^2 E_j$, and column j ’s share of the output error is that error times s_j/c_j^2 . QuIP’s incoherence upper bound on LDLQ’s error (Chee et al., 2023) replaces the mean of s_j/c_j^2 by $(\frac{1}{B} \sum \sqrt{s_j}/c_j)^2$, so the proxy loss of block b , an upper bound rather than the error itself, is

$$L_b(c) = \left(\frac{1}{B} \sum_{j \in b} c_j^2 E_j \right) \left(\frac{1}{B} \sum_{j \in b} \frac{\sqrt{s_j}}{c_j} \right)^2 \geq \left(\frac{1}{B} \sum_{j \in b} (s_j E_j)^{1/3} \right)^3, \quad (5)$$

by Hölder’s inequality with exponents $(3, \frac{3}{2})$, with equality iff $c_j^3 \propto \sqrt{s_j}/E_j$, that is $c_j \propto s_j^{1/6} E_j^{-1/3}$. The minimum is the power mean of order $\frac{1}{3}$ of $u_j = s_j E_j$, the same cube-root form as the optimal point density of high-rate scalar quantization (Panter & Dite, 1951); B3+’s scale ignores E_j . Its relation to SmoothQuant, AWQ and QuIP’s own exponent is discussed with Eq. 1. Our derivation fixes $\alpha = 1/3$ only for the bound L_b ; with error feedback the bound’s exponent is not shown optimal for LDLQ’s exact error, and we did not sweep α , so a different exponent may give lower perplexity. B4 uses this scale, normalised to geometric mean 1 and clipped to $[1/64, 64]$. It is stored where B3+’s scale is (one fp16 value per column), so the file layout is unchanged, and it is the only change: same tables, rate classes, damping and LDLQ, with 256 calibration sequences instead of 512 (disk limit), which we expect to hurt rather than help B4 (not tested).

Pre-registration and result. Before the build we wrote down the prediction (17.45 [17.00, 17.85]) and the rule: B4 replaces B3+ only if its paired ΔNLL against B3+ is negative with a 95% interval excluding zero (supplementary material). The full file reaches 16.64 on WikiText-2 (below the interval: the prediction missed), 24.36 on held-out C4 and a 72-chunk KL divergence of 0.585 ± 0.008 , with paired ΔNLL -0.066 ± 0.009 (70/73 windows) (WikiText-2) and -0.068 ± 0.007 (73/73, corrected) (C4) against B3+, and -0.542 ± 0.020 (73/73 windows) and -0.425 ± 0.013 (73/73, corrected) against IQ2_XXS. Speed equals B3+’s (tg128 $1.007\times$, pp512 $1.009\times$ (median of 3 interleaved rounds)). The cube-root scale is optimal for the bound L_b ; it is not shown optimal for LDLQ’s exact error or for perplexity. Across 16 Qwen3-4B builds the summed output error ranks perplexity only loosely (Spearman 0.63, leave-one-out error 0.054 nats), and it does not detect the Llama failure of Appendix F (supplementary material).

Control without the scale (pre-registered). B4 rebuilt with $c_j = 1$, otherwise unchanged, reaches 16.58 on WikiText-2 (ΔNLL $c_j=1$ minus B4 -0.004 ± 0.009 , lag-1 corrected) and 25.56 on C4 ($+0.048 \pm 0.005$; B4 better on 72 of 73 windows). By the pre-registered rule the derived scale is not separated from $c_j = 1$ on WikiText-2; it helps on C4.

Same-path evaluation. Perplexity and KL divergence elsewhere in this paper use the slab path and generation speed the batch-1 GEMV (Section 3). On the GEMV path itself (micro-batch 1) B4 gives perplexity 16.68 against 16.64 on the slab path ($+0.26\%$; paired ΔNLL $+0.0026 \pm 0.0007$, 73 windows) and KL divergence 0.586 against 0.585 (72 chunks), at 182.8 tokens/s, $1.11\times$ IQ2_XXS and IQ2_KT at B3’s bytes (per-round paired ratios 1.109 [1.099, 1.119] and 1.106 [1.094, 1.117], 10 exclusive rounds in a B4-only session); on that path it stays below the tuned IQ2_KT by -0.091 ± 0.010 (72/73 windows). B4 does not show B3’s batch-1 gap (Appendix L.3). It shares the batched weakness: at 2, 4 and 8 parallel sequences its aggregate decode rate on the shipped path is 0.18, 0.26 and $0.35\times$ IQ2_XXS (llama-batched-bench, a separate session; 0.34 at 8 sequences in Table 4), since only the batch-1 GEMV avoids the slab.

Multi-column decoding (post hoc). With 2 to 8 sequences the shipped runtime takes the slab path, whose pre-pass decodes the whole model every step (34.3–34.8 ms per step at 2, 4 or 8 sequences, against 5.53 ms for the batch-1 GEMV). We added a GEMV for 2, 4 and 8 activation columns that decodes each weight group once and accumulates into every column; larger batches keep the slab path. On all 252 tensors and 2 to 8 columns, every column equals the batch-1 kernel bit for bit where both sum in a fixed order (256-column blocks), and within 6.3×10^{-7} relative otherwise (float atomics). Perplexity on 8 windows moves by at most 0.0027 nats per window, inside twice the shipped kernel’s own run-to-run spread (0.0018); our pre-registered gate of 0.0005 was below that spread, so this acceptance is post hoc. In the one-session bench with all files (Figure 3b) the kernel gives 0.92, 1.06 and $1.09\times$ IQ2_XXS at 2, 4 and 8 sequences. IQ2_XXS’s batch-1 rate ranges from 139.4 to 203.1 tokens/s across our sessions (46%), so we report only same-session ratios; the predicted absolute rates (supplementary material) were met at 8 sequences only.

Trellis ablation. To measure what the Pingala-8 trellis contributes inside B4, we rebuilt B4 with only the trellis memory removed: the encoder runs the one-state greedy LDLQ path, the decoder’s coset vector is zero, and the codebook is our memoryless 256-point four-dimensional table (8 bits per 4 weights, as in B4). Everything else is B4’s, and the blob bytes equal B4’s exactly (972,043,712 B, all 252 tensors). One limit: the memoryless table’s own residual table for later stages cannot be stored in the coset container, so all stages read the first table, as they do in B4. Before the build we predicted $+0.045$ [$+0.025$, $+0.075$] nats (90% interval) against B4 and wrote the rule: the trellis counts as load-bearing if the 95% interval excludes zero, and as the main source of B4’s gain only if the cost exceeds 15% of B4’s gain over IQ2_XXS (supplementary material). The memoryless file reaches 17.96 on WikiText-2 (73 windows) against 16.64 for B4 re-measured with the same binary (identical to the original run), a paired ΔNLL of $+0.076$ [$+0.062$, $+0.090$] (lag-1 corrected), with B4 better on 70 of 73 windows (Figure 4c). The prediction missed (the interval’s upper end was 0.075). The trellis is therefore load-bearing but not the main source of B4’s gain over IQ2_XXS (14%, -0.466 for the memoryless file); against the tuned IQ2_KT, the memoryless file keeps only -0.018 ± 0.017 (45 of 73 windows) of B4’s -0.094 (both on the slab path, -0.094 ± 0.010 (71/73, corrected)), so the trellis supplies about 80% of that margin.

F. Llama-3.1-8B: the Failed Flat Build and B4

We applied the flat rule blind to Llama-3.1-8B (base) with frozen scripts and a pre-registration written before the build finished (predicted perplexity of this flat build 9.6 [8.6, 10.9] on 70 WikiText-2 windows; IQ2_XXS measured 12.35). The native file (2,389,851,808 bytes, 9.36 MB under IQ2_XXS) and its Q8_0 carrier both fail: the first window scores 11.19 against 12.14 for IQ2_XXS, but from the second window on the perplexity diverges (cumulative 6,490 after two windows).

Five of the 224 linear tensors, all attention projections of layers 0 and 1, have a relative Frobenius error above the weights' own norm (layer 0 query 6.82; per-tensor values in the supplementary material), although the build's Hessian-weighted statistics rated them better than IQ2_XXS, because with damping 0.1 of the mean Hessian diagonal the encoder placed the error in input directions that the C4 calibration text barely excites; WikiText-2 excites them. By the pre-registered rules the flat rule does not transfer to Llama-3.1-8B as built. These tensors are not the whole cause (re-encoding them does not remove the failure, below); we report the build as a failure of a calibration that never shows the encoder a BOS-less sequence, and we now check every tensor's unweighted error against the reference before any evaluation. Post hoc, we traced the failure to the protocol's BOS-less windows: llama.cpp's strided mode writes the BOS only after decoding, so every window after the first starts with an ordinary token as the attention sink, while every calibration sequence of our build starts with a BOS. Without a BOS the first-token massive activation does not form in our file (median peak activation 74 against 393 for Q8_0; IQ2_XXS is close to Q8_0). With a BOS at every chunk (standard llama.cpp perplexity) on 20 windows of WikiText-2 *validation* text, the same pre-registered file scores 10.16 against 11.29 for IQ2_XXS, but IQ2_KT at the same budget scores 9.27 there and beats it. Re-encoding the five worst tensors with Frobenius weighting does not remove the BOS-less failure (70-window test perplexity 80,206). IQ2_KT at the same budget (2,389,350,432 bytes, from the Q8_0) reaches 10.04 on the pre-registered protocol (IQ2_XXS 12.35), so it is the bar a robust build has to beat.

B4 on Llama-3.1-8B (pre-registered). After the flat build failed we applied B4 to Llama-3.1-8B with the same build engine and evaluation protocol and one calibration change, also used by the intermediate build: every second calibration sequence starts mid-text without a BOS, so that the encoder sees the plain-token attention sink of the evaluation windows. We wrote down before the build the prediction (9.85 [9.50, 10.25]) and the rule: B4 counts as a win only if its paired ΔNLL against IQ2_KT at the Llama budget (2,389,350,432 bytes, built from the Q8_0) is negative with a 95% interval excluding zero. The native file has 2,388,210,816 bytes (1,139,616 fewer than IQ2_KT) and puts 32.3 MB more into the token embedding and 33.5 MB less into the linear layers than IQ2_KT. On 70 WikiText-2 windows it reaches 9.14 (IQ2_KT 10.04, IQ2_XXS 12.35, an intermediate build with the column scale in layers 0–1 only 10.38): paired ΔNLL -0.094 ± 0.005 against IQ2_KT, -0.301 ± 0.010 against IQ2_XXS and -0.127 ± 0.006 against that intermediate build, each better on 70 of 70 windows (intervals corrected for lag-1 autocorrelation). The prediction missed: the result lies below its interval. Two details of the comparison (the three caveats are in Section 4.3): IQ2_KT was quantized from the Q8_0, which on Qwen3-4B we estimate costs it about 0.026 nats against a bf16 source, well below the margin; and the build ran in six runs, five of them resumed after memory stops by replaying the finished layers.

G. Qwen3-14B: the Pre-registered Loss in Detail

Before any 2-bit 14B file or evaluation existed we fixed the recipe (B4's, unchanged: rate-off classes, column scale, 256 calibration sequences), the sources (Q8_0 for all three files), IQ2_KT's recipe (the ik-rule allocation, at no more bytes than B4) and the rule: B4 wins iff its paired WikiText-2 ΔNLL against IQ2_KT is negative with a 95% interval excluding zero, and loses iff it is positive with such an interval. It loses. B4 reaches 9.5684 against 9.3985 for IQ2_KT and 10.5413 for IQ2_XXS (Table 1), ΔNLL $+0.018 \pm 0.008$ against IQ2_KT (better on 12 of 73 windows) and -0.097 ± 0.007 against IQ2_XXS (73 of 73; intervals corrected for lag-1 autocorrelation), with 633,312 more bytes than IQ2_KT and 10,489,888 fewer than IQ2_XXS. On held-out C4 the two tie: 15.2490 against 15.2349, ΔNLL $+0.001 \pm 0.006$ (better on 33 of 73), and B4 beats IQ2_XXS (16.6968) by -0.091 ± 0.006 . On WikiText-2 our prediction for B4 held (10.6 [9.2, 12.6]) and those for IQ2_KT (11.3 [9.7, 13.6]) and IQ2_XXS (15.0 [11.5, 21.0]) missed low; on C4 all three missed low (B4: 19.0 [15.5, 24.0]); both margin predictions missed high (against IQ2_KT -0.06 [-0.11 , -0.01], against IQ2_XXS -0.35 [-0.55 , -0.18]): both codebook files gained far more from scale than we assumed. On Qwen3, B4's margin over IQ2_XXS shrinks from 0.54 nats at 4B to 0.10 at 14B (WikiText-2, same protocol). We have not diagnosed the loss. The comparison is at equal file bytes, not equal linear bytes: IQ2_KT's recipe stores the token embedding as IQ2_KT where B4 inherits IQ2_XXS's Q2_K, which leaves it 2.14 bits per linear weight against B4's 2.11, and it gives attn_v 3.44 bits on 24 of 40 layers while B4's classes were chosen on Qwen3-4B (on Qwen3-4B, IQ2_KT's own allocation was worth 0.030 nats to it: 18.27 against 18.83 with B3's). A pre-registered check of the first, IQ2_KT rebuilt with a Q2_K token embedding (48,011,776 B larger), changes its WikiText-2 NLL by -0.0003 ± 0.0005 (perplexity 9.3958 against 9.3985), so the 2-bit embedding costs IQ2_KT nothing measurable; the attn_v allocation is untested. No speed or KL divergence was measured at 14B.

H. CPU Backend

This appendix is in the supplementary material under the same letter.

I. Ablations in Detail

This appendix is in the supplementary material under the same letter.

J. Quality reference: Model A (entropy-coded, not deployable)

This appendix is in the supplementary material under the same letter.

K. Reproducibility details

A notation table for the symbols of the method is in the supplementary material.

Settings. Table 19 lists the settings of Model B, the first shipped build, and of the evaluation. B4 differs in its recipe (one rate class per tensor, the column scale of Appendix E.4, 256 calibration sequences; Section 3) and in its evaluation counts, which are those of the main text (72 KL chunks, 229 speed runs).

K.1. Pre-registration files

Every prediction and decision rule that this paper calls pre-registered was written to a plain-text file before the run it governs, and the files will be released with the code. Table 20 lists the first 16 hexadecimal digits of the SHA-256 hash of each file the paper relies on, with the file’s last-modified time (IST). The hashes fix the contents of the files as shipped; several files carry result lines appended after their runs, so a hash alone does not date a prediction. The pre-registration of B3+ is a section of a design note rather than a separate file and is not listed.

L. Scalability and impact of the format

This appendix is in the supplementary material under the same letter.

Table 6. What this paper borrows and what it adds. Claims about other work are limited to what their abstracts, READMEs or pull requests state (Appendix A.1).

Component	Origin	Status here
Trellis-coded quantization; set partitioning; the 8-state code and its labels	Ungerboeck (1982); Marcellin & Fischer (1990); Fischer et al. (1991) (vector TCQ); dependent quantization of network weights (Haase et al., 2020)	Borrowed unchanged (Ungerboeck’s feed-forward labelling)
Trellis codes for LLM weights; Hadamard incoherence; bitshift trellis lineage	QTIP (Tseng et al., 2024b); EXL3 (turboderp-org, 2025); ik_llama.cpp IQ*_KT (Kawrakow & ik_llama.cpp contributors, 2024); Proteus (Yang et al., 2026); CCQ (Zhou et al., 2025)	Borrowed in concept. Ours is a 4-dimensional coset code with a 256-entry table and 8 states (QTIP: 2^L states, $L \geq 8$)
Hessian-aware rounding along the trellis (LDLQ)	QuIP (Chee et al., 2023)	Borrowed
Calibration statistics: importance matrix, layer-wise Hessians	llama.cpp imatrix (Gerganov & llama.cpp contributors, 2023); GPTQ (Frantar et al., 2023)	Borrowed
Size target, GGUF container and tensor mix (attn_v Q4_K, 4 ffn_down Q2_K, Q5_K embedding)	llama.cpp IQ2_XXS (Kawrakow, 2024b)	Borrowed as the target; the IQ2_XXS grid code is replaced
Bit allocation under a storage budget	Q-Palette (Lee & Song, 2025); ScaleBITS (Li et al., 2026); Align-Quant (Zhang et al., 2026)	Idea borrowed. 8 rate classes per 32×256 tile of one trellis code, a fixed byte count per class, warp-uniform; measured here: no perplexity gain at equal bytes against one class per tensor (Section 5.4)
State names	Pingala, Chandahsutra 1.1–1.8, and the prastara (ch. 8) (Pingala, 1938)	Naming only; the state count is not load-bearing (Section 5.3)
Runtime and file	this paper	New as a deployment, not as a trellis type (IQ2_KT is a trellis GGUF type in a fork; CCQ, EXL3 and QTIP use their own kernels or engines): a GGUF file at IQ2_XXS size and a CUDA and AVX2 patch set on llama.cpp commit 7ac59a6 (not merged upstream) with batch-1 decoding as fast as IQ2_XXS or faster (B4: $1.096 [1.034, 1.158] \times$, one exclusive session with all files; B3+: 1.07); prompts through the stock Q8_0 matrix multiply ($0.74 \times$) and batched decoding of 2–8 sequences at $0.18\text{--}0.34 \times$ ($0.92\text{--}1.09 \times$ with a post hoc multi-column kernel)
Allocation finding	this paper	New: an equal-bytes control in which the weight-error proxy and perplexity rank per-tile and flat allocation in opposite orders; the flat build B3
Comparison	this paper	New: same-harness evaluation against stock quants, ik_llama.cpp IQ2_KT and third-party checkpoints, losses included

Table 19. Build and evaluation settings of Model B (B4: see the text above). All side information listed here is stored in the model file.

Item	Value
Model	Qwen3-4B (Hugging Face Qwen/Qwen3-4B); fp16 GGUF reference for perplexity, Q8_0 for KL divergence
Calibration text	English C4 train shard 00000, documents from index 200000, 6.0M characters; no word 10-gram shared with the imatrix text or WikiText-2 test
Hessian sequences	512 fit + 32 held-out sequences of 2048 tokens; layer-by-layer fp16 propagation
Importance matrix	llama.cpp imatrix on C4 calibration text, context 512, 200 chunks
Column prescale	$c_j = \sqrt{\max(s_j/\bar{s}, 10^{-2})}$ from the imatrix column energies s_j , rounded to fp16
Rotation	random-sign fast Walsh–Hadamard transform over 256 columns, sign seed 0
Hessian damping	$0.1 \times$ mean diagonal added before the float64 Cholesky factorisation
Scale	one fp16 scale per 256 weights per row, $1.05 \times$ the block RMS divided by the class gain
Trellis	8 states ($\nu = 3$), Ungerboeck feedforward labels, coset $B = A + c$, state reset every 32 symbols
Codebook	256 int8 4-vectors (tables in the release, tables_vrc)
Rate classes	0, 1, 1.5, 2, 2.5, 3, 4, 6 bits per weight on tiles of 32×256 ; residual shifts $SH_1 = (0, 0, 0, 0, 2, 3, 2, 4)$, $SH_2 = (0, 0, 0, 0, 0, 0, 0, 2)$ per class
Salient protection	4 most salient tile-blocks per tensor forced to class ≥ 6 ; sparse fp32 delta for clamped outliers
Byte budget	per tensor, IQ2_XXS-slot tensors at the entropy-coded reference bytes, others at their IQ2_XXS-mix bytes, minus a 96-byte margin
Perplexity	WikiText-2 test (raw, joined), llama-perplexity strided mode, windows of 4096 tokens (stride 4095), 73 windows; fp16 reference 12.4567
KL divergence	against Q8_0 base logits, context 512, 12 chunks
Zero-shot	llama-perplexity task modes (-hellaswag, -winogrande, -multiple-choice), 0-shot, continuation log-likelihood: HellaSwag validation (1000 items), WinoGrande debiased evaluation file (1267), ARC-Easy and ARC-Challenge validation (570, 299), MMLU validation file (1548), full MMLU test (14,042)
Speed	llama-bench -ngl 99 -fa 1 -ctk f16 -ctv f16; tg128 as -p 0 -n 128 (batch-1 decoding) and pp512 as -p 512 -n 0, separate runs; -r 15 in two earlier IQ2_KT sessions and -r 5 in all others; default batch and micro-batch; Table 4 and Figure 3: 10 rotated rounds, mean of the per-round ratios with a 95% t -interval; earlier sessions: 3 rounds with rotated arm order; per round the median of the llama-bench samples, then the ratio to IQ2_XXS in the same round, then the median over rounds. Quiet: no other GPU job; exclusive: the GPU held by the benchmark (CPU shared with a quantize job); shared: other GPU jobs active. Exclusive 10-round bench (Tables 4 and 13): 10 rounds with rotated arm order, -r 5, mean of the same-round ratios to IQ2_XXS with a 95% t -interval, no other GPU process in any of its 126 runs; micro-batch sweep -p 64 -n 0 -b 2048 -ub 1,2,4,8,16,32 -r 3, one pass per file
Power, memory	nvidia-smi sampled every 100 ms during each llama-bench run (power draw, SM and memory clocks, temperature, memory used). Energy per token: mean power over samples with SM clock above 1000 MHz divided by tokens/s. Peak VRAM: maximum memory used on the whole GPU (other processes included)
Significance	paired per-window NLL differences over the 73 windows, mean ± 1.96 standard errors
Software	llama.cpp 7ac59a6 (build32) with our CUDA patch, built with GGML_CUDA=ON, CMAKE_CUDA_ARCHITECTURES=120, CMAKE_BUILD_TYPE=Release, GGML_CUDA_FA=ON, GGML_CUDA_GRAPH=ON, GGML_NATIVE=ON and nvcc 13.2 (from the build’s CMake cache); ik_llama.cpp 77bef17 for IQ2_KT
Hardware	one NVIDIA GeForce RTX 5070 Ti Laptop GPU (12 GB), driver 592.01, WSL2
Build time	4623 s for the whole Qwen3-4B build (2768 s in the quantiser)

Table 20. Pre-registration files cited in this paper: SHA-256 prefix and last-modified time (IST, 2026). Files are listed in order of the work they govern.

File	SHA-256 prefix	Modified	Governs
prereg_mirror.txt	66997f2e924d0a8f	10-14 19:31	equal-bytes per-tile arm of the proxy control (Appendix D.5)
prereg_flat.txt	0e8d265f4d7a1e08	10-14 23:31	B2, the first size-matched flat build
prereg_b3.txt	c11f6ce51f094de6	10-17 13:38	B3
prereg_b3_heldout.txt	42f7d72677036ad6	10-17 17:20	checks (i)–(iii) of B3 (Appendix D.3)
prereg_llama8b.txt	78a30524e4afcb25	10-17 21:39	flat build on Llama-3.1-8B (Appendix F)
prereg_ktfair.txt	b72d386399dd5e94	10-18 00:59	three perplexity-tuned IQ2_KT recipes, check (iv)
prereg_ktc4imat.txt	57f326959b4d3588	10-18 02:24	C4-calibrated IQ2_KT, check (v)
prereg_ktllama.txt	75e124af9651b2c7	10-18 04:56	IQ2_KT on Llama-3.1-8B
prereg_cscale_T1.txt	b8ca66dbb2fb02d6	10-18 11:30	column scale in layers 0–1 of Llama-3.1-8B (intermediate build)
prereg_cscale_qfull.txt	50a9aa35a84ec413	10-18 12:43	B4 on Qwen3-4B (rule: beat B3+)
prereg_samepath_b4.txt	018f2b0d0cc1effd	10-18 15:50	same-path (GEMV) evaluation of B4
prereg_trellis_ablation.txt	a0dff7b738ed22439	10-18 15:53	trellis ablation of B4
prereg_gemv_mc.txt	3724d749de4ace30	10-18 18:11	multi-column GEMV
prereg_review_evals.txt	3e6171a0f1102816	10-18 20:43	B4 zero-shot, KL divergence of the C4-calibrated IQ2_KT, one-session speed bench
prereg_cscale_lfull.txt	4856347f95e6a988	10-18 22:34	B4 on Llama-3.1-8B (rule: beat IQ2_KT)
prereg_b4_q14.txt	52b03cc88b91ee3d	10-18 13:47	B4 on Qwen3-14B
prereg_b4_q14_addendum.txt	63c5185f8ae0a8d3	10-18 16:36	addendum to the 14B rules
prereg_llama_bos.txt	603228a5246716e1	10-19 00:47	B4 vs IQ2_KT on Llama-3.1-8B, BOS at every window
prereg_w1_unpatched.txt	cd31595b8515097d	10-19 02:20	IQ2_S, IQ2_M, IQ3_XXS on unpatched llama.cpp
prereg_cscale_c1.txt	c3ca81afd709f47f	10-19 01:47	B4 without the column scale ($c_j = 1$), Qwen3-4B
prereg_te14.txt	f841c3cfaa683565	10-19 05:25	Q2_K token embedding for IQ2_KT, Qwen3-14B (Appendix G)