COMPUTED PERMUTATION TRELLIS CODING: NEAR-OPTIMAL QUANTIZATION DESIGNED FOR NATIVE HARDWARE ACCELERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

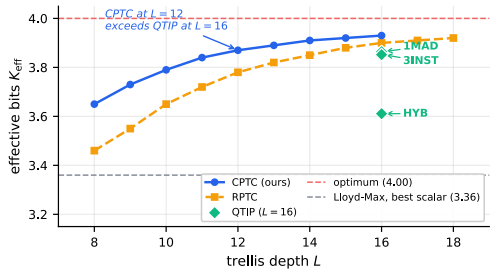
We propose **Computed Permutation Trellis Coding (CPTC)**, a low-bit weight format whose decoder multiplies the trellis state by two 16-bit coefficients and uses the top bits of the products to index a 64-byte table. Formats that round each value independently, such as INT4, FP4, and NF4, waste 0.6 or more of their 4 bits against the information-theoretic ideal; trellis codes recover this loss by encoding values jointly, but prior decoders either store a large table (Random Permutation Trellis Coding, RPTC) or replace it with a hash (QTIP) that fixes the distribution of the decoded values. CPTC computes the table index instead: the index distribution the two multiplications induce is a near-optimal rule for placing codepoints in a small table. One decoder with 16-bit multipliers, a 64-byte table and a coefficient pair loaded per configuration serves every bit width $K \in [1, 6]$, reproduces the standard integer and floating-point formats exactly, and loses at most 0.07 bits against the information-theoretic ideal for all $K \leq 4$, matching RPTC run two state bits deeper from an $8{,}192\times$ smaller table and exceeding QTIP's best variant (3.93 vs. 3.86 effective bits at $K{=}4$). In LLM weight quantization without fine-tuning, CPTC has the lowest KL divergence to the unquantized model at every tested bit width on LLaMA 3.1 8B and 70B Instruct against QTIP's HYB (by up to 27%) and 1MAD (by up to 5%) codebooks, and at 3–4 bits surpasses even YAQA's fine-tuned 70B releases. A fused persistent GPU Viterbi kernel encodes an 8B-parameter model in about two GPU-minutes, and the decoder has been built into the weight-load path of a commercial systolic-array accelerator, where, post-layout, it runs at native FP4/FP8 throughput for about 3% of the systolic array's area.

1 INTRODUCTION

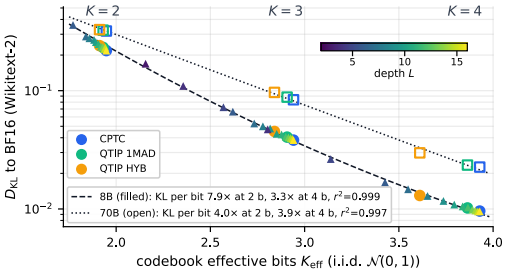
When a large language model generates text for a few users at a time, the speed limit is not arithmetic but memory: every token requires streaming all model weights from memory into the compute units, so halving the bits per weight roughly doubles tokens per second. At four bits and below, though, every 0.1 bit of wasted precision shows up in model quality.

How much can K bits per value achieve? For Gaussian data, which LLM weights are made to resemble by standard preprocessing (Section 2), information theory sets a floor on the error: the distortion (MSE) D of any K-bit quantizer satisfies $D \geq 4^{-K}$ (Berger, 1971). We measure quantizers in *effective bits*, $K_{\text{eff}} = -\log_4(D)$ (Section 2), so the ideal 4-bit quantizer scores 4.00. Effective bits are the paper's quality metric because they determine the quantized model's quality: across codebooks, rates, depths and two model scales, the quantized LLM's KL from its BF16 original is one decreasing function of K_{eff}, 3–8$\times$ per bit, with the codebook order preserved in every comparison (Figure 1b). A better Gaussian quantizer is a better LLM quantizer, so the rest of the paper can compare codebooks on K_{eff}. Formats that round each value on its own fall well short: the best possible, the Lloyd-Max quantizer (Lloyd, 1982; Max, 1960), scores 3.36, and INT4, FP4, and NF4 lower still (Table 1). The lost ~ 0.6 bits can only be recovered by encoding *groups* of values jointly. *Bitshift trellis codes* (Marcellin & Fischer, 1990; Tseng et al., 2024b) do exactly that while keeping decoding trivial: each value is recovered from an L-bit window of the code stream by a fixed map cheap enough to run per weight. Prior versions each miss one requirement of a hardware decoder

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



(a) Effective bits (↑) against trellis depth L at 4 bits: CPTC and RPTC at each depth, the QTIP variants at L=16, the best scalar format (Lloyd-Max) and the optimum 4.00 (Tables 2 and 11).



(b) Model quality follows the codebook's effective bits: KL to BF16 (↓) of LLaMA 3.1 8B (filled) and 70B (open) against K_{eff} at 2, 3 and 4 bits, plus the CPTC depth sweep (triangles). More effective bits, lower KL, in every codebook comparison; each bit is worth 4× in KL at 70B and 3–8× at 8B (curves; Section 4.2).

Figure 1: Effective bits as the paper's quality metric.

(Section 2): RPTC (Mao et al., 2011) needs a 2^L-entry table, and QTIP (Tseng et al., 2024b) hashes the table away but fixes the output distribution.

We show the table can be *computed* instead of stored or hashed away. Our decoder multiplies the trellis state by two 16-bit coefficients, takes the top six bits of each product, adds them, and uses the sum to index a 127-entry symmetric table (64 bytes) holding the target distribution's values. The two multiplications scramble the state like RPTC's random permutation, but their *sum* concentrates near the middle of its range (approximately triangular), so the table is sampled most densely where a bell-shaped target puts most of its probability; 64 bytes keep the full trellis advantage at every $K \in [1, 6]$.

One L=16 decoder serves every bit width from a searched coefficient pair and 64 table bytes per rate: within 0.07 bits of the ideal for all $K \leq 4$ (Table 3; 3.93 at K=4), matching RPTC run two state bits deeper from 8,192× less table storage and beating QTIP at equal L, at a decode cost independent of depth and bit width. With L=K it reproduces the standard $[1, 6]$-bit integer and floating-point formats bit for bit; $L > K$ improves on the scalar format in every tested setting but one tie.

Trellis coding's traditional cost is the encoder's Viterbi search over 2^L states per value, usually written off as an offline expense; one persistent fused GPU kernel encodes 59 million values/s at L=16 (an 8B model in about two GPU-minutes) and nearly a billion at L=12 (Section 4.3). Our main contributions are:

- **Decoder.** Two 16-bit multiplications and a 64-byte symmetric table give near-ideal trellis quality at constant cost for every $K \leq 6$ (Section 3); the index distribution they induce is a near-optimal codepoint-placement rule (Section 3.5).
- **Theory.** Every $[1, 6]$-bit scalar format and the prior trellis codes are special cases of this decoder (Section 3.4); the full-table reduction meets the marginal condition for every coefficient choice (Appendix B.3); floors on table size and data type leave no cheaper design point without measurable loss (Table 15).
- **Quality.** At L=16 under one protocol, CPTC leads RPTC and all QTIP variants for $K \in [2, 5]$ and matches 4–128 KiB searched codebooks within 0.002 bits for $K \in [2, 4]$ (Table 2); the same 64 bytes retarget it to other distributions (Section 4.1 and Appendix C.5). In YAQA (Tseng et al., 2026) without fine-tuning it has the lowest KL at every rate on LLaMA 3.1 8B and 70B against QTIP's HYB and 1MAD codebooks, beating the fine-tuned 70B releases at $K \in \{3, 4\}$ (Section 4.2).
- **Systems.** The decoder runs in a commercial systolic-array accelerator's weight-load path at about 3% of the array's area with FP4/FP8 throughput parity (Section 4.3); a fused GPU Viterbi encoder sustains 59M values/s at L=16, a 16× method-level and 1,200× end-to-end advantage over QTIP's encoder at equal quality (Appendix D.1).

2 BACKGROUND

LLM weight quantization and the standard-normal reduction. When few tokens are processed at once, an LLM layer's speed is set by how fast its weights stream from memory, so weights are compressed offline and decompressed on the fly in the weight-load path. The *Random Hadamard Transform* (RHT) (Chee et al., 2023; Tseng et al., 2024a) reduces this to a clean statistical problem: it mixes the weights so thoroughly that, after rescaling, they closely approximate i.i.d. standard-normal draws. We adopt this preprocessing unchanged, so CPTC's quality is directly comparable to prior work built on it, and what remains is quantizing i.i.d. Gaussian samples as accurately as possible.

Effective bits. All quality numbers use one calibrated scale grounded in the source's *rate-distortion function*. Let X be the source random variable (after the preprocessing above, the unit-variance target distribution, quantized as i.i.d. copies), $\widehat{X}$ its reconstruction, $D = \mathbb{E}\big[(X - \widehat{X})^2\big]$ the distortion (MSE), and $R_X(\cdot)$ the rate-distortion function of X under squared error (Berger, 1971): $R_X(D)$ is the minimum rate, in bits per value, at which *any* coder can reconstruct X with distortion at most D. We define *effective bits* as the Shannon lower bound on R_X at the measured distortion, the left end of the classical sandwich

$$K_{\text{eff}} \;\coloneqq\; -\log_4 \frac{D}{N(X)} \;\le\; R_X(D) \;\le\; -\log_4 \frac{D}{N(X+Z)}, \qquad N(X) = \frac{e^{2h(X)}}{2\pi e}, \tag{1}$$

where N is entropy power, h is differential entropy in nats, and $Z \sim \mathcal{N}(0, D)$ is independent test-channel noise. The quantizer under test is itself a coder spending K bits per value, so $K_{\text{eff}} \le R_X(D) \le K$: the deficit $K - K_{\text{eff}}$ never understates the wasted rate, and one effective bit is a $4\times$ factor in distortion. For the Gaussian (our default target) the bound is exact ($N(X) = 1$, $K_{\text{eff}} = R_X(D)$, ceiling K achievable as block length grows); for the other targets we use, Appendix C.2 verifies the sandwich is thin at the rates that matter, so measured deficits are genuine coding loss.

From scalar to joint quantization. The scalar ceiling of Section 1 (Lloyd-Max at 3.36 effective bits, with INT4, FP4, and NF4 (Dettmers et al., 2023) below it) binds every method that rounds values one at a time. GPTQ (Frantar et al., 2023), AWQ (Lin et al., 2024), and SqueezeLLM (Kim et al., 2024) improve the *assignment* into a fixed scalar format and inherit its ceiling; CPTC replaces the format itself, standing where INT4 or NF4 stand, so they compose with it. Joint quantization escapes the ceiling (Conway & Sloane, 1999): QuIP# (Tseng et al., 2024a) uses the E_8 lattice, NestQuant (Savkin et al., 2025) nests it self-similarly, LLVQ (van der Ouderaa et al., 2026) extends to the 24-dimensional Leech lattice, and learned-codebook vector quantization (VQ; AQLM, Egiazarian et al., 2024; VPTQ, Liu et al., 2024) trains megabyte-scale codebooks. CPTC instead encodes values jointly as a walk through a 2^L-state trellis at a per-value decode cost of two multiplies and a lookup.

Trellis-coded quantization. Trellis-coded quantization (Marcellin & Fischer, 1990) quantizes jointly while spending exactly K bits per value: each value's reconstruction depends on its neighbors' code bits too. The decoder keeps an L-bit state q, simply the most recent L bits of the code stream, and maps it to a value through a fixed function ϕ (Figure 2). In the *bitshift* subfamily relevant here, reading the next value's K bits just shifts them into the state register, which is why decoding stays hardware-trivial. Although only K bits are stored per value, consecutive values share $L - K$ state bits, so the decoder effectively selects from a 2^L-entry codebook, the source of the quality beyond scalar formats. The price is a Viterbi encoding search over all 2^L states (Section 3.2).

Prior bitshift trellis codes. The map ϕ from state to value is where prior designs differ. *Linear congruential* (LC) trellis codes (Eriksson et al., 2007), the earliest computed codebooks, take branch labels from congruential maps of the state and read values at the labels' quantiles from a stored table. *RPTC* (Mao et al., 2011) shuffles the 2^L states with a stored random permutation P and assigns state q the value at quantile $(P[q] + \frac{1}{2})/2^L$ of the target; this provably meets the marginal condition of asymptotically optimal codes and approaches the Shannon bound as L grows, but the shuffle is 2^L entries. *QTIP* (Tseng et al., 2024b) computes ϕ with a hash: its 1MAD and 3INST variants need no table but can only emit the approximately Gaussian law the hash produces, and its HYB variant regains the choice of distribution through a stored k-means codebook.

Table 1: What each codebook costs and delivers, one column per requirement of Section 2: decoder work and storage per lane, whether one circuit serves every bit width, whether the output distribution can be changed by loading a table, how the encoder searches, and how much of the 4 bits is wasted (↓, bits consumed minus K_{eff} on i.i.d. Gaussians; sources Tables 2 and 12, Figure 1a, and Appendix C.1; n/r: not reported). †QTIP's quality at $L-4$, $16\times$ less search. ‡At 4.25 bits with a per-block scale index.

Codebook	Decoder (ops, storage)	All K	Retargetable	Encode/value	Waste at 4 bits
INT4 / FP4 / NF4	cast, 0 / 0 / 32 B	×, per width	×	round	0.85 / 0.84 / 0.66
MXFP4 / NVFP4 (4.25/4.5 b)	cast + scale, 0	×, per width	×	round	1.09 / 1.11
E8P (QuIP#)	signs + 2 KiB LUT	×, per rate	×	E8 structured search	0.56
Nested E8 (NestQuant)	E8 closest point, 0	✓, any q	×	E8 closest point, $O(T)$	$0.36^{\ddagger}$
Learned VQ	MiB LUT	×, per rate	✓	beam search, 2^{TK}	n/r
LC trellis	1 mul + 4–128 KiB LUT	✓	✓	Viterbi, 2^L	0.07
RPTC	128 KiB LUT	✓	✓	Viterbi, 2^L	0.10
QTIP HYB	hash + 2 KiB LUT	×, $K \leq 4$	✓	Viterbi, 2^L	0.39
QTIP 1MAD / 3INST	1–3 int ops, 0	✓	×, fixed law	Viterbi, 2^L	0.14 / 0.15
CPTC (ours)	2 mul + add + 64 B LUT	✓, $K \in [1, 6]$	✓, exact scalar	Viterbi, 2^L $(2^{L-4})^{\dagger}$	0.07

Requirements for a hardware codebook. Weight formats are decoded in a datapath fixed at tape-out and replicated per lane beside the multiply-accumulate units, so every lane pays for whatever the decoder holds, which sets four requirements. It must be *cheap*: a few integer operations and register-resident state per value; a kilobyte-scale table per lane, or one shared with a read port per lane, has no place in the weight-load path. It must be *general* across bit widths: one circuit serving every rate, ideally reproducing the scalar formats it replaces. It must be *flexible* in its output distribution: the target law will change over the chip's lifetime, so retargeting must mean loading state, not respinning silicon. And it must deliver *high quality*, or the block is not worth its area. Encoding is not exempt: KV caches must be quantized online at token rate (encoder throughput, Section 4.3), so the encoder's search cost counts too. Table 1 scores existing codebooks on these axes; every prior design fails at least one. Encoding a T-dimensional VQ searches 2^{TK} codewords unless it is a lattice with a fast closest-point algorithm; trellis codes pay a Viterbi search over 2^L states, affordable online for CPTC thanks to its quality at small L and a fused encoder. The next section shows a *computed* permutation meets all four.

3 Computed Permutation Trellis Coding

CPTC replaces RPTC's stored random permutation with a computed one: the L-bit state is multiplied by two fixed constants, the top bits of the two products are added, and the sum indexes a 64-byte table of output values. This section gives the decoder (Section 3.1), the encoder (Section 3.2), how the table is filled (Section 3.3), which existing formats the decoder reproduces (Section 3.4), and why two multiplications suit bell-shaped targets (Sections 3.5 and 3.6).

3.1 Decode primitive

The CPTC decoder is parameterized by two coefficients a_0, a_1 of $L_{\max}$ bits (we use $L_{\max} = 16$ throughout), a codebook table $\mathcal{T}$ of $2^{Q+1}-1$ entries with tap width $Q \leq L_{\max}$, and an optional shift $s \in \{0, \ldots, Q-1\}$ (default 0). Given the $L_{\max}$-bit state q (only the low L bits carry information, Section 3.2), one value decodes as in Algorithm 1: the state is multiplied by two coefficients and the top bits of the two products are added to form the table index, all native integer operations.

No floating-point unit is required at decode time. $L_{\max} = 16$ is where the returns diminish: encoding costs $O(2^L)$ while quality saturates (the last doubling buys 0.01 bits, Figure 1a), and 16-bit operands align with SIMD lane widths and hardware datapaths. When the codebook is mirror-symmetric ($\mathcal{T}[i] = -\mathcal{T}[2^{Q+1}-2-i]$), only the lower half ($2^Q$ entries) is stored, and lines 6–10 of Algorithm 1 serve the read with one comparison and a conditional sign flip (in hardware, a mux on the index and sign bit). We use $Q = 6$ unless stated (a 127-entry codebook stored as 64 bytes): the storage baked into a commercial ML accelerator's decode path (Appendix D.4), and the smallest size retaining trellis gain across $K \in [1, 6]$ (Section 3.3).

Algorithm 1 CPTC decode (per value), with multiplier width $L_{\max} = 16$. Lines 6–10 are an optional storage optimization: for mirror-symmetric codebooks only the lower half $\mathcal{T}_\downarrow$ (2^Q entries) is stored and the upper half is served by a mirrored lookup (Section 3.3).

Require: $L_{\max}$-bit state q; coefficients $a_0, a_1 \in \{0, \dots, 2^{L_{\max}}-1\}$; codebook table $\mathcal{T}$, or its lower half $\mathcal{T}_\downarrow$ under symmetric storage; optional shift s (default $s = 0$).

1: $Q \leftarrow \log_2\big(|\mathcal{T}| + 1\big) - 1$ ▷ tap width; $|\mathcal{T}| = 2^{Q+1}-1$, $s \in \{0, \dots, Q-1\}$
2: $x_0 \leftarrow (a_0 \cdot q) \bmod 2^{L_{\max}}, \quad x_1 \leftarrow (a_1 \cdot q) \bmod 2^{L_{\max}}$
3: $x_0 \leftarrow x_0 \gg (L_{\max} - Q + s), \quad x_1 \leftarrow x_1 \gg (L_{\max} - Q + s)$ ▷ $x_0, x_1 \in [0, 2^{Q-s}-1]$
4: $i \leftarrow x_0 + x_1$ ▷ $i \in [0, 2^{Q-s+1}-2]$
5: **return** $\mathcal{T}[i]$ ▷ if the full table is stored; else lines 6–10
6: **if** $i < 2^Q$ **then** ▷ symmetric-storage lookup (optional)
7: **return** $\mathcal{T}_\downarrow[i]$
8: **else**
9: **return** $-\mathcal{T}_\downarrow[2^{Q+1} - 2 - i]$
10: **end if**

Cross-implementation compatibility. The parameters compose into a dominance rule. Say data type $\mathcal{D}$ *dominates* $\mathcal{D}'$ if some scale $c > 0$ makes every $\mathcal{D}'$-representable codepoint $\mathcal{D}$-representable ($c \cdot \mathcal{D}' \subseteq \mathcal{D}$; e.g. E2M5, the 8-bit float with 2 exponent and 5 mantissa bits, dominates INT6, Section 3.3).

Proposition 1 (Dominance)**.** *A decoder with parameters $(L_{\max}, Q, \mathcal{D})$ bit-exactly emulates any decoder with $L'_{\max} \le L_{\max}$, $Q' \le Q$, and $\mathcal{D}'$ dominated by $\mathcal{D}$: replace each coefficient a_j by $a_j \cdot 2^{L_{\max}-L'_{\max}}$, set the shift $s = Q - Q'$, and load the table rescaled by c.*

Proofs are in Appendix A.3, verified exhaustively against a golden reference decoder: hardware built once for $(L_{\max}, Q, \mathcal{D})$ serves every configuration below it.

3.2 Trellis state and Viterbi encoder

A block is decoded by sliding an $L_{\max}$-bit window over its TK code bits in steps of K and passing each window to Algorithm 1; Figure 2 and Algorithm 2 (both in Appendix A.1) show the layout and procedure. Two properties make the raw-window read exact. First, no masking to L bits is needed: the coefficients carry $L_{\max} - L$ trailing zeros, which annihilate any stream bits above a window's low L under multiplication modulo $2^{L_{\max}}$. Second, *tail-biting*: the block's first $L_{\max} - K$ bits are logically appended to its end, closing the trellis into a cycle, so every value decodes from a full L-bit state with no start-up overhead. A running state register equivalently replaces the window (the products then update incrementally).

The encoder is the standard Viterbi search for the lowest-MSE walk on the 2^L-state trellis: $O(T \cdot 2^L)$ time, $O(T \cdot 2^{L-K})$ backtrack memory. Block boundaries use tail-biting (Tseng et al., 2024b; Ma & Wolf, 1986) to remove the $(L-K)$-bit boundary overhead at the cost of running Viterbi twice per block. Weights are static, so the cost is paid once per model, and our fused persistent GPU kernel makes it modest: about two GPU-minutes for an 8B model at L=16 (Section 4.3).

3.3 Symmetric lookup table and output types

The index $i = x_0 + x_1$ takes values in $[0, 2^{Q+1}-2]$ ($[0, 126]$ at Q=6). For target distributions symmetric about zero, CPTC stores only the lower half of the codebook and derives the upper half by the mirrored lookup (Algorithm 1, lines 6–10); the fold is exact because the coefficient search selects pairs whose index histogram is perfectly mirror-symmetric (Appendix A.2). Symmetry is a storage optimization, not a requirement: with $a_0 = 0$ the mirror branch is inactive and the table holds *any* distribution. Since the decoder must reproduce any 2^K-entry scalar codebook (Section 3.4), the table needs 2^K entries to surpass scalar quality at the largest K; for $K \le 6$ this fixes $Q = 6$, matching the accelerator hardware path, and the entries are stored in 8-bit types (E2M5 or INT6; quality-neutral through K=4, Appendix C.3).

The table is populated offline to match a target distribution with CDF F by composition with the (a_0, a_1)-specific empirical index distribution. Let $h[i]$ count the 2^L states whose index equals i (by

enumeration), and let $G(i) = 2^{-L} \sum_{i' \le i} h[i']$ be the empirical index CDF, $G(-1) = 0$. Cell i owns the probability interval $[G(i{-}1), G(i)]$, and its entry is the *conditional mean* of the target over that interval,

$$\mathcal{T}[i] = \mathbb{E}_{X \sim F}\big[X \,\big|\, F(X) \in [G(i{-}1),\, G(i)]\big], \tag{2}$$

then rounded to the table data type. Each entry is thus the average of the target values its cell represents: Lloyd's optimality condition for the given partition, so the table is the best possible for this index map (the oracle ablation of Appendix B.1 confirms it). The target enters solely through its CDF, so retargeting changes only the 2^Q stored values: one coefficient pair, searched once, serves every target (the tables use per-rate pairs, worth ≤ 0.015 bits).

3.4 Multiplicative coefficients, scalar emulation, and RPTC subsumption

The trellis depth L is implicit in the coefficients: $L = 16 - \mathrm{ctz}(a_0 \mid a_1)$, where ctz is the trailing-zero count. For $L \le Q$, a single non-zero coefficient suffices ($a_0 = 0$, a_1 an odd multiple of 2^{16-L} as in Table 5, $s = Q{-}L$; $a_1 = 2^{16-L}$ itself is the scalar embedding of Proposition 2); for $L > Q$, two distinct non-zero coefficients produce the near-triangular index distribution the table exploits.

Best coefficients. We search (a_0, a_1) per L on 2^{19}–2^{20} Philox-generated $\mathcal{N}(0,1)$ samples, exhaustively at small L and sampled above (Appendix A.2 tabulates pairs and protocol); quality is monotone in L, saturating at $K_{\text{eff}}{=}3.93$ ($L{=}16$).

CPTC generalizes scalar quantization. Every scalar format is a degenerate instance:

Proposition 2 (Scalar embedding). *For $K \in \{1, \dots, Q\}$ and any 2^K-entry codebook $\mathcal{C} = \{c_q\}$ with c_q representable in the table data type, the CPTC decoder with $a_0 = 0$, $a_1 = 2^{16-K}$, $s = Q{-}K$, and $\mathcal{T}[q] = c_q$ produces $\hat{w} = c_q$ for every q.*

The proof is direct from Algorithm 1 ($a_0{=}0$ zeros x_0; the shift recovers $x_1 = q$), with $L = K$ and the trellis degenerate. As a corollary, every INTK, FPK, NF4, Lloyd-Max, and any distribution-optimal K-bit scalar format is a CPTC instance with $L = K$ (NF4-like irrational codepoints need a table type matching the implementation's output precision, e.g. FP16). At $L > K$ the trellis couples consecutive values through shared state, so alphabet containment alone does not *guarantee* improvement; but $L{=}K$ recovers the scalar format exactly, so adopting the decoder never costs quality, and $L > K$ was empirically better in every tested configuration but one tie ($\beta{=}\infty$, $K{=}1$; Table 3). Likewise, for $L \le Q$ RPTC's 2^L-entry codebook fits in CPTC's table verbatim (Proposition 2 with $K \leftarrow L$); for $L > Q$ CPTC computes a structured permutation in place of RPTC's random one, trading 2^L entries for two multiplications.

3.5 The triangular index distribution is a near-optimal companding law

With only $2^{Q+1}{-}1$ distinct codepoints, the table itself sets a floor on quality that the trellis cannot dig below: probability mass beyond the largest codepoint is simply clipped, no matter how cleverly the encoder chooses paths, so the key design question is *where the codepoints land*. The table rule of Equation (2) places them according to how often each index cell is hit: the index distribution acts as a *companding law* (Bennett, 1948), with codepoint density $\lambda(x) \propto p(x)/g(x)$ for target density p and index law g.

The obvious choice is a *uniform* index law (equal probability mass per codepoint), and it is what a single multiplication produces. But it fails in the tails: with $M{=}127$ equal-mass Gaussian codepoints the largest sits at only 2.66σ, and the clipped mass beyond it accounts for 83% of the codebook's error. The truly optimal placement, $\lambda^\star \propto p^{1/3}$ (Panter & Dite, 1951), deliberately spreads codepoints *wider* than the data ($\sqrt{3}\times$ for a Gaussian) to protect the tails, and this is precisely what the second multiplication buys: each product's top bits are equidistributed, so their *sum* has a *triangular* distribution whose rarely-hit edge cells stretch the composed codebook to an effectively $\sqrt{2}$-wider profile (3.67σ outermost quantile codepoint at $M{=}127$), between the source width and the optimal $\sqrt{3}$ (Figure 3). This captures nearly all of the available gain: the triangular-composed codebook stays within 7–17% of the Lloyd-Max optimum at every table size, while end-to-end at $L{=}16$, $K{=}4$ the equal-mass table costs 0.72 effective bits ($3.92 \to 3.20$ at $T{=}128$, below the Lloyd-Max scalar: the trellis cannot rescue a clipped tail) and an oracle-optimized table improves nothing.

Appendix B.1 develops the rest, and Appendices A.2 and C.1 explain why searched coefficients beat *random* permutations at equal L (Figure 1a).

3.6 Theoretical limit and extensions

Three further threads are developed in the appendices. *Achievable rate* (Appendix B.3): with $a_0 = 0$ and $Q = L$, CPTC is exactly RPTC with the algebraic permutation $P_a(q) = aq \bmod 2^L$ in place of a random one, and it provably satisfies, for *every* coefficient choice, the necessary marginal-optimality condition of Mao et al. (2011) (Proposition 3), with the open joint conditions stated as Conjecture 1. *Matmul-aware tables* (Appendix C.5): the table encodes the reproduction distribution, not the decode path, so the bimodal matmul-optimal point density of Ang et al. (2026) loads by the same composition rule with everything else unchanged. *In-stream metadata* (Appendix A.4): parity constraints let the encoder commit auxiliary bits (block scale, zero point, flags) into the codeword itself, at $1/g$ effective bits per auxiliary bit per group of g values.

4 Experiments

We evaluate CPTC on (i) i.i.d. source quantization (Section 4.1), the regime the RHT creates; (ii) LLM weight-only quantization (Section 4.2); and (iii) implementation throughput on GPU and in hardware (Section 4.3). A fourth (Appendix C.5) retargets the table to the bimodal matmul-aware density of Ang et al. (2026) with no decode-path change, up to $2.3\times$ lower error than the best standard scalar format. Unless noted, CPTC uses $L{=}16$, the coefficients of Table 5, $T{=}256$ (i.i.d.) or $T{=}128$ (matmul), and tail-biting encoding.

4.1 I.i.d. source quantization

We generate 2^{22} i.i.d. samples per target and seed and report effective bits (Equation (1)).

Standard normal at $K{=}4$ (Figure 1a). CPTC with $L{=}16$ and a 64-byte table reaches $K_{\text{eff}} = 3.93$, matching RPTC even two trellis-depth doublings up (3.92 at $L{=}18$, Table 11) and QTIP's best 4-bit variant (3.86), whose score CPTC reaches already at $L{=}12$. Scalar and block-scaled formats trail by 0.5–0.8 bits (Table 1).

Codebook quality across bit widths (Table 2). We re-ran every prior trellis codebook at $L{=}16$ under one protocol: QTIP rows run the authors' released code (HYB emits value pairs from a $Q{=}9$ k-means table), RPTC is the random-permutation quantile codebook, and LC is the stronger of two searched single-multiplication forms per column (Appendix C.1). The CPTC$_{Q7}$ rows double the table (E2M5 and FP16). CPTC leads every QTIP variant for $K \geq 2$, by a margin growing from 0.017 bits at $K{=}2$ to 0.25 at $K{=}6$: a hash controls its marginal but not codepoint *placement* (Section 3.5), and HYB's fixed alphabet caps it past $K{=}4$ (Theorem 1, Appendix B.2). With 64–$2000\times$ the storage, LC leads CPTC by 0.001–0.002 bits for $K \in [2, 4]$ (0.024/0.15 at $K{=}5/6$, cut to a third by CPTC$_{Q7}$); at matched table size (128 vs. 127 cells) a single multiplication reaches only 3.78–3.88 at $K{=}4$ (Appendix B.2).

Beyond the Gaussian (Table 3). The unit-variance generalized Gaussian $p_\beta(x) \propto \exp(-|x/\alpha|^\beta)$ interpolates from the uniform ($\beta \to \infty$) through the normal ($\beta{=}2$) to the Laplacian ($\beta{=}1$) (Banner et al., 2019); per β only the 64 table bytes change, and K_{eff} is entropy-power normalized (Equation (1)). The trellis advantage over the per-(β, K) Lloyd-Max scalar optimum holds everywhere but one tie ($\beta{=}\infty$, $K{=}1$) and *grows* with tail weight (0.19 bits at $\beta{=}\infty$ to 0.93 at $\beta{=}1$, $K{=}4$); at $\beta{=}1$ CPTC nears scalar quality with a full bit less rate (2.89 at $K{=}3$ vs. scalar's 2.91 at $K{=}4$). CPTC matches or exceeds RPTC across the family, and the equal-mass-composition penalty grows with tail weight, confirming Section 3.5 end-to-end (Table 14). The $\beta{=}2$ row is the bit-width sweep: gap to the bound 0.05–0.07 bits for $K \leq 4$, growing to 0.24 at $K{=}6$ as the 127-codepoint floor approaches.

No cheaper design point without loss (Table 15, Appendix C.3). CPTC's decode cost (the per-value map, the table size and its data type) sits at a provable or matched minimum on all three: one

Table 2: Effective bits (↑) of every trellis codebook at each bit width, all run at L=16 under one protocol (rows described in the text), with the table each decoder must store (CPTC in 8-bit E2M5 unless marked FP16, others FP16; "—": nothing stored). Bold: within 0.003 of the underlined column best.

Codebook	Storage	K=1	K=2	K=3	K=4	K=5	K=6
QTIP HYB	2 KiB	0.932	1.912	2.840	3.611	3.936	3.936
QTIP 3INST	—	0.936	1.931	2.902	3.852	4.748	5.469
QTIP 1MAD	—	0.933	1.931	2.907	3.864	4.765	5.514
RPTC	128 KiB	**0.940**	1.937	2.917	3.896	4.872	5.840
LC	4–128 KiB	**0.940**	**1.950**	**2.942**	**3.931**	**4.919**	**5.908**
CPTC	64 B	0.935	**1.948**	**2.941**	**3.929**	4.895	5.759
CPTC$_{Q7}$	128 B	0.935	**1.947**	**2.941**	**3.930**	4.911	5.860
CPTC$_{Q7\text{-FP16}}$	256 B	0.935	**1.947**	**2.941**	**3.931**	**4.917**	5.884

Table 3: Effective bits (↑) of CPTC on targets from uniform (β=∞) to Laplacian (β=1), with the best scalar quantizer for each cell in parentheses; only the 64 table bytes change between rows (L=16, T=256, 3 seeds; entropy-power normalized, so the limit is K in every column).

β	K=1	K=2	K=3	K=4	K=5	K=6
∞	0.75 (0.75)	1.87 (1.75)	2.92 (2.75)	3.94 (3.75)	4.94 (4.75)	5.89 (5.75)
2.0	0.93 (0.73)	1.95 (1.54)	2.94 (2.43)	3.93 (3.36)	4.89 (4.32)	5.76 (5.30)
1.5	0.92 (0.63)	1.94 (1.42)	2.93 (2.29)	3.91 (3.21)	4.85 (4.17)	5.64 (5.15)
1.0	0.88 (0.40)	1.90 (1.15)	2.89 (1.99)	3.84 (2.91)	4.68 (3.86)	5.22 (4.84)

multiplication protects the tails only with a tuned scale and trails at finite depth (Appendix B.2); by Proposition 4, $Q = K-1$ or an INT(K) table leaves 2^K-1 codepoints, one short of a rate-K scalar quantizer, so *no trellis depth can match scalar quality*; and two doublings above that floor ($Q = K+1$ in INT(K+2)) costs at most 0.04 bits against the Q=6 FP16 reference for $K \leq 5$ and exceeds it at K=6, where the reference's 127 codepoints bind.

4.2 LLM WEIGHT-ONLY QUANTIZATION

We quantize LLaMA 3.1 Instruct models in the YAQA Sketch-B pipeline of Tseng et al. (2026) without fine-tuning, swapping only the codebook; evaluations use the OPTQ Wikitext-2/C4 splits at sequence length 8,192 (Table 4; details and controls in Appendix C.4). CPTC$_{Q5}$ is the 63-codepoint table of the GPU kernel (Section 4.3, Appendix D.2); all rows use the pipeline's published loading factor and the 70B rows its released recipe (Appendix C.4). The harness reproduces their BF16 baselines and released-checkpoint numbers and regenerates their HYB table within 0.007 KL. Every row but 70B HYB is a mean over three RHT seeds (Table 4 marks it), and both CPTC arms beat 1MAD on every seed at every rate and scale (Appendix C.4). KL tracks the codebook's i.i.d. effective bits (Figure 1b, Appendix C.4): $\log_2 D_{\mathrm{KL}}$ is a smooth function of K_{eff} (quadratic fits, $r^2 \geq 0.997$; each effective bit is 4× in KL at 70B at every rate, as in MSE, and 8× to 3× at 8B from 2 to 4 bits), and across the three codebooks the KL order is the K_{eff} order within every rate and, up to one near-tie, along an 8B depth sweep, so the i.i.d. gains carry to the model. CPTC has the lowest KL divergence to the unquantized model at every rate and scale against QTIP's HYB and 1MAD codebooks. Against HYB, the codebook YAQA releases, the margin *grows* with rate (9% at K=2, 14% at K=3, 27% at K=4 at 8B) as its 512-entry alphabet saturates (Theorem 1); against the table-free 1MAD it is 3–5% at 8B and 1–5% at 70B (full-table arm), the ordering of Table 2. At 70B, CPTC at $K \in \{3, 4\}$ also beats YAQA's *released* checkpoints, which include its fine-tuning stage (0.0225 vs. 0.0294; 0.0856 vs. 0.0910, CPTC$_{Q5}$ arm), while at K=2 fine-tuning keeps the release ahead (0.2672 vs. 0.3145). The Q=5 arm costs at most 0.001 KL at 8B and 0.002 at 70B, and leads at 70B $K \in \{2, 4\}$: the deployment table is no compromise. Zero-shot accuracy separates rates, not formats (within about a point of BF16 at $K \geq 3$, dropping together at K=2). A preliminary run on Qwen3.8-27B, a hybrid gated-DeltaNet model, shows the same ordering (Table 20).

Table 4: LLaMA 3.1 Instruct quantized in YAQA's pipeline (Tseng et al., 2026) with only the codebook swapped, no fine-tuning: KL to the BF16 model (↓), perplexities (↓) and mean 0-shot accuracy (↑; ARC-c/e, BoolQ, HellaSwag, PIQA; within noise, unbolded). QTIP rows are our runs of its HYB and 1MAD codebooks; the HYB rows match the published HYB numbers within 0.007 KL (Appendix C.4). KL and perplexities average three RHT seeds (Table 18; spread 0.3–4.1%); †one seed; Acc from seed 0. Bold: best per rate and model.

K	Codebook	$D_{\mathrm{KL}}\downarrow$	Perplexity ↓		Acc ↑	$D_{\mathrm{KL}}\downarrow$	Perplexity ↓		Acc ↑
		W2	W2	C4	0-shot	W2	W2	C4	0-shot
		8B-Instruct				70B-Instruct			
16	BF16	0	6.502	8.022	71.4	0	3.524	6.460	77.1
4	QTIP HYB	0.0131	6.608	8.117	71.2	0.0298†	3.685†	6.516†	76.8
	QTIP 1MAD	0.0100	6.579	8.092	71.5	0.0238	3.647	6.500	76.8
	CPTC$_{Q5}$	0.0096	6.570	8.088	71.3	**0.0225**	**3.631**	6.500	76.9
	CPTC	**0.0095**	**6.566**	**8.086**	71.3	0.0227	3.642	**6.499**	76.9
3	QTIP HYB	0.0450	6.852	8.344	70.8	0.0967†	4.035†	6.649†	76.3
	QTIP 1MAD	0.0405	6.824	8.314	70.3	0.0876	3.965	6.628	76.4
	CPTC$_{Q5}$	0.0396	6.828	**8.294**	70.6	0.0856	3.968	**6.621**	76.6
	CPTC	**0.0387**	**6.797**	8.297	70.5	**0.0845**	**3.946**	**6.621**	76.6
2	QTIP HYB	0.2424	8.402	9.798	66.7	0.3281†	5.259†	7.316†	74.3
	QTIP 1MAD	0.2271	8.254	9.696	67.3	0.3221	5.225	7.286	74.3
	CPTC$_{Q5}$	0.2212	**8.228**	9.653	67.1	**0.3145**	**5.191**	**7.265**	74.7
	CPTC	**0.2202**	8.238	**9.634**	67.4	0.3176	5.210	7.269	74.5

4.3 Implementation throughput and hardware

Decode. The decode primitive maps to two integer multiplies, an add, a shift, and a 64-byte indexed load on x86, Arm, and GPUs; in a systolic array's weight-load path, throughput parity to FP4 at $K \leq 4$ and FP8 at $K \leq 6$ costs about 3% of the systolic array's area (Appendix D.4). On stock GPUs, a fused kernel decodes the stream straight into the tensor-core operand registers (batch $\leq$8, Appendix D.2): within 2.4% of an INT4 baseline sharing its pipeline on the bandwidth-bound L40S (3.6× cuBLAS FP16); on the bandwidth-rich H100 the ALU limits instead (61% over it, 27% for a Hopper-only state-table variant; L=10 decoders), the case for a native decode instruction. On CPUs, CPTC at K=4 streams a LLaMA 3.1 8B forward as fast as an INT4 kernel (Appendix D.3).

Encode. Our kernel fuses the tail-biting Viterbi search of Section 3.2 (two passes per length-T block) into one persistent Triton launch. On one L40S, throughput runs from 59 million values/s at L=16, K=4 (T=128; an 8B model in two GPU-minutes) to 2.4 billion at L=10, at MSE within 0.01% of a `torch.compile` reference on identical inputs (Appendix D.1). Against QTIP's shipped encoder, equal-quality encoding runs 1,200× faster on the same GPU (Appendix D.1); the durable part is *method-level*: equal quality at L=12 means 16× less search, since work is $\propto 2^L$. Encoding this fast invites online uses like KV caches at small L: 3.79 bits at L=10, all layers' new blocks in one measured 134 μs launch per LLaMA 3.1 8B token (Appendix D.1).

5 Conclusions

CPTC replaces the stored trellis codebook with a computed one: two 16-bit multiplications and a 64-byte symmetric table, where the triangular index law of the two products does the codepoint placement that a hash cannot. One decoder serves every $K \in [1, 6]$ and the standard scalar formats, meeting all four requirements of Section 2; at K=4 it reaches 3.93 effective bits, above RPTC two depth-doublings up and above QTIP. In YAQA's pipeline it has the lowest KL against QTIP's HYB and 1MAD codebooks on LLaMA 3.1 8B and 70B without fine-tuning, beating fine-tuned 70B releases at $K \in \{3, 4\}$; the decoder runs at FP4/FP8 throughput parity for about 3% of a systolic array's area and at INT4 pace on x86 and Arm CPUs.

Limitations and future work. Viterbi encoding costs $O(2^L)$ per value; the accelerator figures cannot be verified under blind review. KV-cache quantization at L=10 is the subject of a concurrent submission; quantized training and further architectures are next.

REPRODUCIBILITY STATEMENT

Our source release, to be made public upon publication, regenerates from single `make` targets with fixed seeds: all codebook-level analyses of Appendix B.1 (deterministic, CPU-only) and the GPU experiments behind Tables 2, 3, 8, 10, 11 and 13 to 16, the matmul-aware experiment of Appendix C.5, and the enumerated per-L/per-K/Q=7 coefficient searches behind Table 5 and Appendix A.2. All harness experiments ran on one NVIDIA L40S; each table-level experiment completes in minutes, while the coefficient searches run for one to several hours each and are therefore invoked individually. The LLM experiments of Section 4.2 ran YAQA's released pipeline on 8×H100 (their Hessians for LLaMA, our own collection for Qwen3.8-27B) and are not in the harness; the golden reference decoder, the Viterbi encoder (Listing 1), the fused GPU kernel, the CPU kernels and their bit-exactness tests (Appendix D.3), and all experiment scripts will be released as open source upon publication. Algorithm 1 plus Table 5 suffice for an independent reimplementation of the decoder.

AI USE STATEMENT

Generative AI tools were used in three ways: to aid and polish the writing (editing for clarity, grammar and concision, formatting references); to draft sections of the paper from the authors' notes and results; and to assist research ideation and execution (exploring analyses, writing experiment and figure scripts, and running experiments). The authors directed this work, verified every claim and number, and take responsibility for the final content, including all text produced with the aid of generative AI.

REFERENCES

Calvin Ang, Sungyoon Kim, and Mert Pilanci. Optimal scalar quantization for matrix multiplication: Closed-form density and phase transition. *arXiv preprint arXiv:2603.19559*, 2026.

Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. arXiv:1810.05723.

W. R. Bennett. Spectra of quantized signals. *Bell System Technical Journal*, 27(3):446–472, 1948. doi: 10.1002/j.1538-7305.1948.tb01340.x.

Toby Berger. *Rate-Distortion Theory: A Mathematical Basis for Data Compression*. Prentice-Hall, 1971.

Richard E. Blahut. Computation of channel capacity and rate-distortion functions. *IEEE Transactions on Information Theory*, 18(4):460–473, 1972.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. QuIP: 2-bit quantization of large language models with guarantees. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

John H. Conway and Neil J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer, 3 edition, 1999.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Martin Dietzfelbinger, Torben Hagerup, Jyrki Katajainen, and Martti Penttonen. A reliable randomized algorithm for the closest-pair problem. *Journal of Algorithms*, 25(1):19–51, 1997. doi: 10.1006/jagm.1997.0873.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2401.06118.

Tomas Eriksson, John B. Anderson, and Norbert Görtz. Linear congruential trellis source codes: Design and analysis. *IEEE Transactions on Communications*, 55(9):1693–1701, 2007.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

Robert M. Gray. Time-invariant trellis encoding of ergodic discrete-time sources with a fidelity criterion. *IEEE Transactions on Information Theory*, 23(1):71–83, 1977.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W. Mahoney, and Kurt Keutzer. SqueezeLLM: Dense-and-sparse quantization. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2306.07629.

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

Yifei Liu, Jicheng Wen, Yang Wang, Shengyu Ye, Li Lyna Zhang, Ting Cao, Cheng Li, and Mao Yang. VPTQ: Extreme low-bit vector post-training quantization for large language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2024. arXiv:2409.17066.

Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28 (2):129–137, 1982. originally circulated as Bell Labs Technical Memorandum, 1957.

Howard H. Ma and Jack K. Wolf. On tail biting convolutional codes. *IEEE Transactions on Communications*, 34(2):104–111, 1986. doi: 10.1109/TCOM.1986.1096498.

Mark Z. Mao, Robert M. Gray, and Tamás Linder. Rate-constrained simulation and source coding i.i.d. sources. *IEEE Transactions on Information Theory*, 57(7):4516–4529, 2011. doi: 10.1109/TIT.2011.2145930.

Michael W. Marcellin and Thomas R. Fischer. Trellis coded quantization of memoryless and Gauss-Markov sources. *IEEE Transactions on Communications*, 38(1):82–93, 1990.

Joel Max. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6(1):7–12, 1960.

Saralees Nadarajah. A generalized normal distribution. *Journal of Applied Statistics*, 32(7):685–694, 2005. doi: 10.1080/02664760500079464.

Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1992.

P. F. Panter and W. Dite. Quantization distortion in pulse-count modulation with nonuniform spacing of levels. *Proceedings of the IRE*, 39(1):44–48, 1951. doi: 10.1109/JRPROC.1951.230419.

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

Semyon Savkin, Eitan Porat, Or Ordentlich, and Yury Polyanskiy. NestQuant: Nested lattice quantization for matrix products and LLMs. In *International Conference on Machine Learning (ICML)*, 2025. arXiv:2502.09720.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *International Conference on Machine Learning (ICML)*, 2024a. arXiv:2402.04396.

Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. QTIP: Quantization with trellises and incoherence processing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024b. arXiv:2406.11235.

Albert Tseng, Zhaofeng Sun, and Christopher De Sa. Model-preserving adaptive rounding. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2505.22988.

Tycho F. A. van der Ouderaa, Mart van Baalen, Paul Whatmough, and Markus Nagel. Leech lattice vector quantization for efficient LLM compression. *arXiv preprint arXiv:2603.11021*, 2026.

Andrew J. Viterbi and James K. Omura. Trellis encoding of memoryless discrete-time sources with a fidelity criterion. *IEEE Transactions on Information Theory*, 20(3):325–332, 1974.

APPENDIX

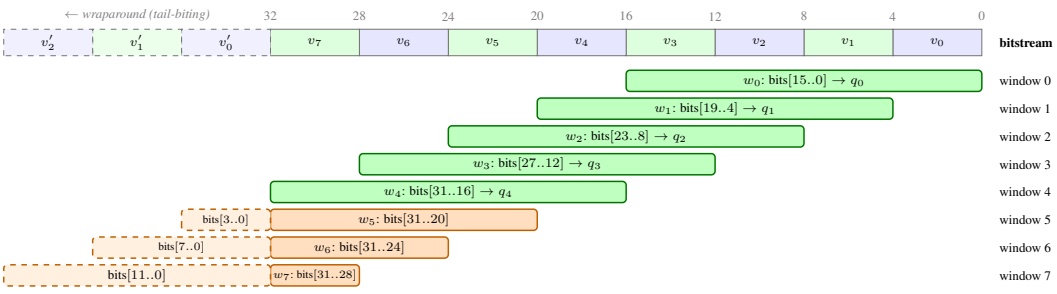
The appendix has four parts, in the order the main text uses them. Appendix A gives the details of the decoder and encoder of Section 3: the tail-biting stream layout (Appendix A.1), the coefficient tables and search protocol (Appendix A.2), the scalar-format emulation proofs (Appendix A.3), and in-stream metadata (Appendix A.4). Appendix B is the theory behind Sections 3.5 and 3.6: the index-law ablation (Appendix B.1), the alphabet-ceiling theorem (Appendix B.2), and the achievable-rate reduction with its open conjecture (Appendix B.3). Appendix C extends the experiments of Sections 4.1 and 4.2 (Appendices C.1 to C.4) and holds the matmul-aware experiment (Appendix C.5). Appendix D covers the systems side of Section 4.3: the fused Viterbi encoder (Appendix D.1), the GPU and CPU decode kernels (Appendices D.2 and D.3), and the accelerator implementation (Appendix D.4).

A DECODER AND ENCODER DETAILS

A.1 TAIL-BITING CONSTRUCTION

This appendix gives the stream layout and the block decode procedure behind Section 3.2: Figure 2 shows the layout and Algorithm 2 the procedure.

Construction. Tail-biting trellis codes were introduced by Ma & Wolf (1986) in convolutional coding and adapted to bitshift trellis quantization by Tseng et al. (2024b); we use the QTIP construction unchanged. The encoder appends the block's first $L_{\max} - K$ bits onto its own end (line 1 of Algorithm 2; only the low $L - K$ bits of each window carry state), so the L-bit windows read at positions $0, K, \ldots, (T-1)K$ close into a cycle: the start and end states agree by construction and no bits are spent on start-up. The encoder runs Viterbi twice per block (once to find the best ending state for an unconstrained start, once with the start state pinned to that ending state), per Tseng et al. (2024b).



Algorithm 2 CPTC block decode from the packed bitstream (tail-biting).

Require: code bits $b[0 \ldots TK{-}1]$ of one block (K bits per value, T values); parameters of Algorithm 1.

1: $b' \leftarrow b \,\|\, b[0 \ldots L_{\max}{-}K{-}1]$ ▷ tail-biting: wrap the first $L_{\max}{-}K$ bits
2: **for** $t = 0, 1, \ldots, T-1$ **do**
3: $\quad q \leftarrow \sum_{k=0}^{L_{\max}-1} b'[tK{+}k] \cdot 2^k$ ▷ little-endian window; no masking to L needed
4: $\quad \hat{w}_t \leftarrow \textsc{Decode}(q)$ ▷ Algorithm 1
5: **end for**
6: **return** $\hat{w}_0, \ldots, \hat{w}_{T-1}$

Figure 2: How a block is decoded from the packed bitstream. **Top:** bit layout for $K{=}4$, $T{=}8$, $L_{\max}{=}16$: value v_t's code occupies bits tK to $tK{+}K{-}1$ (bit 0 rightmost), window t reads the 16 bits starting at bit tK, and the last three windows (orange) wrap into the dashed tail-biting copy of the first 12 bits. **Bottom:** the procedure (Algorithm 2).

A.2 RECOMMENDED MULTIPLICATIVE COEFFICIENTS

Table 5 gives recommended (a_0, a_1) pairs, searched over pairs in $[0, 2^L)^2$ whose OR is odd on 2^{19}–2^{20} Philox-generated $\mathcal{N}(0,1)$ samples (a selection seed disjoint from every evaluation seed) at the paper's table-size parameter $Q = 6$: per depth, a per-K best pair for every $K \in [1, 6]$ and a single *unified* pair selected on all rates jointly (minimum mean deficit $K - K_{\text{eff}}$ over the six rates). The K=4 column is the shipped pair, from a search exhaustive through L=10 and sampled above (which the plateau below justifies); the other columns re-score 2^{16} mirror-symmetric candidates per depth on all six rates. Coefficients are in the 16-bit form of Algorithm 1 ($16 - L$ trailing zeros). For $L \leq 6$, $a_0 = 0$ suffices, since all 2^L states map directly into the table; for $L \geq 7$ both coefficients are non-zero, producing the triangular index law of Section 3.5. All searches require a perfectly mirror-symmetric index histogram ($h[i] = h[126 - i]$), which makes the fold of Section 3.3 exact. At low rates and small depths the per-K winners are equal pairs ($a_0 = a_1$, a single rotation): there center resolution matters more than tail protection. Quality per pair is monotone in L and saturates at K_{eff}=3.93 (L=16, K=4); the unified pair is within 0.015 bits of the per-K best at every rate for $L \geq 12$.

Table 5: The coefficient pairs to use, in hex, for each trellis depth L: the best pair for each bit width K and one unified pair for all rates, searched at Q=6. The K=4 column is the shipped pair; for $L \leq 6$ one degenerate pair serves every K.

L	K=1	K=2	K=3	K=4 (shipped)	K=5	K=6	Unified
1				0000,8000 (one pair serves every K)			
2				0000,C000 (one pair serves every K)			
3				0000,A000 (one pair serves every K)			
4				0000,5000 (one pair serves every K)			
5				0000,2800 (one pair serves every K)			
6				0000,A400 (one pair serves every K)			
7	8600,8600	C600,C600	2200,2200	5E00,BC00	0A00,1400	0400,8200	6200,C400
8	8900,8900	4700,4700	A100,A100	5200,6900	6100,8400	0400,4100	9700,AE00
9	8980,8980	4280,4280	B980,E600	3A80,B500	6500,9280	1D80,FB00	9900,AC80
10	8940,8940	BE40,BE40	6AC0,7580	39C0,4E00	5240,C480	0D80,B6C0	A880,C440
11	76A0,76A0	41E0,41E0	8540,9AA0	7560,7AC0	6040,9820	7820,E040	9BE0,A7C0
12	8E70,B4E0	7820,D010	7DA0,E2D0	6D30,8260	5040,B810	1B10,BE20	4550,82A0
13	8D68,9ED0	86D0,B968	8320,DCC8	CEB0,E158	D820,F008	E3F0,E7F8	43F8,83F0
14	866C,9AD8	3068,8734	74B0,AB2C	D1FC,DDF8	07F0,CFFC	B208,DA04	7E48,C824
15	6B6E,73DC	792E,C75C	1D48,8252	5130,7F26	539A,B434	5A84,9DC2	483E,677C
16	8C22,9651	7AE6,C8B3	7474,AB1D	81D3,CE26	483E,9C5F	0AEE,D337	9282,A981

What makes a coefficient pair bad: a two-sided damage law. Exhaustive Viterbi evaluation of every admissible pair at L=8 (24,640 pairs) and L=10 (393,472), plus a 40k-pair sample at L=12, shows that failure is governed by one quantity, the sum $\Sigma = a_0 + a_1 \bmod 2^L$, through two norms. *Archimedean:* when $|\Sigma|$ is small, $a_1 q = -a_0 q + \Sigma q$ and negation forces $\text{top}(a_0 q) + \text{top}(-a_0 q) \approx 2^{Q+1} - 2$ up to carries, so the index depends on q only through the slow rotation Σq and the code degenerates toward a single rotation with multiplier Σ (at $\Sigma = 0$ the index is constant and K_{eff} collapses to 0.4–0.7; mean quality recovers beyond $|\Sigma| \gtrsim 2^{L-5}$). *2-adic:* when the valuation $\nu_2(\Sigma) = k$ is large, $\Sigma q \bmod 2^L$ takes only 2^{L-k} values, the index support collapses to $\sim 2^{L-k}$ cells, and the alphabet floor of Proposition 4 binds. The valuation of the difference $a_1 - a_0$ is nearly harmless: it is the sum that cancels. Rejecting pairs with $|\Sigma| < 2^{L-5}$ or $\nu_2(\Sigma) > 2$ discards $\approx$14% of the space with no false rejections: no discarded pair is within 0.05 bits of the optimum at either depth.

Towards optimal pairs: a plateau. Outside the damage set the landscape is a broadening plateau, not a peak: the fraction of surviving pairs within 0.05 bits of the optimum grows from 15% (L=8) to 30% (L=10) to 63% (L=12), and the max-to-median gap shrinks $0.11 \to 0.07 \to 0.04$ bits, so the multiplicative family exhibits the same distortion concentration as random permutations (Appendix C.1), tightening with depth. The search's job is therefore to avoid the damage set and pick from a large near-optimal set: in the original unconstrained K=4 search, the shipped L=16 pair was within 0.0004 bits of the best of 3,000 random candidates under the full protocol, and the only mirror-symmetric pair in the top tier; the 2^{16}-pair symmetric sweep behind Table 5 confirms the picture at scale.

All-K selection. At L=16 the unified pair scores $(0.934, 1.944, 2.937, 3.928, 4.901, 5.808)$ for $K = 1, \ldots, 6$ (16-bit tables), within 0.002 bits of the shipped pair for $K \in [2, 4]$ and within 0.015 bits of the per-K optimum at every rate, so joint selection is essentially free at the design point. The stakes are larger at small depths: at L=10 the unified pair trades 0.02 bits at K=4 for 0.08–0.21 at the other rates, the relevant trade for online small-L uses (Section 4.3). The paper's quality tables load the per-K column for each rate; the unified column serves deployments that fix one pair for all rates.

Coefficients for $Q = 7$. Where the 127-codepoint alphabet floor binds ($K \geq 5$), one table doubling recovers most of the deficit while keeping the symmetric fold. Re-searching pairs whose 255-cell Q=7 histogram is full and mirror-symmetric yields, with FP16 tables, 4.917 at K=5 (pair $(\texttt{0xBEC5}, \texttt{0xC514})$) and 5.884 at K=6 (pair $(\texttt{0x06CE}, \texttt{0x13E7})$), with $K \leq 4$ unchanged; in E2M5 under the no-clipping scale rule of Appendix C.3 the same pairs score 4.911/5.860, the CPTC$_{Q7}$ row of Table 2. A single unified Q=7 pair $(\texttt{0x518D}, \texttt{0x781A})$ is within 0.006 bits of the per-K winners at every rate. By Proposition 1, Q=7 hardware serves every $Q \leq 7$ configuration.

Block size T. Larger T gives the Viterbi encoder more freedom to find a good cyclic walk; smaller T adds tail-biting boundaries. Empirically (Table 6) quality saturates near $T = 128$ on the standard normal at K=4, L=16.

Table 6: How effective bits (↑) depend on the block length T (K=4, L=16, standard protocol).

Block size T	16	32	64	128	256
K_{eff}	3.23	3.72	3.88	3.92	3.93

A.3 Scalar format emulation

This appendix gives the constructions behind the scalar-format reductions of Section 3.4 and the proof of the dominance rule.

INT6 via CPTC at K=6. Setting $K = 6$, $a_0 = \texttt{0x0000}$, $a_1 = \texttt{0x0400}$, $s = 0$, and $\mathcal{T}[q] = q - 32$ for $q \in [0, 63]$ (a signed INT6 ramp) reduces Algorithm 1 to a direct INT6 lookup. Since $L = 16 - \text{ctz}(\texttt{0x0400}) = 6 = K$, the trellis is degenerate. All 64 indices fall in the lower half of the table; the symmetric upper half is unused.

FP4 via CPTC at K=4. Setting $K = 4$, $a_0 = \texttt{0x0000}$, $a_1 = \texttt{0x1000}$, $s = 2$, and populating the first 16 entries of $\mathcal{T}$ with the scaled-INT6-encoded FP4 codepoints (value $\times 4$) reduces the decoder to an FP4 lookup.

Dominance-rule mechanics (proof of Proposition 1). The bit-exact emulation works direction by direction. *Table direction:* the shift s lets a Q-sized implementation emulate one built with a smaller table parameter $Q' = Q - s$: for $s \geq 1$ every index in Algorithm 1 falls below 2^Q, so the mirror branch of Algorithm 1 (lines 6–10) is inactive and the emulated codebook, signs stored explicitly, occupies the first $2^{Q-s+1}-1$ entries of $\mathcal{T}$ (both embeddings above use this). *Width direction:* replacing $a_j \pmod{2^{L'_{\max}}}$ by $a_j \cdot 2^{L_{\max}-L'_{\max}}$ shifts every product's bits up in lockstep, and the top-bits extraction of Algorithm 1 line 3 removes the difference; this is verified exhaustively against the golden reference decoder of the source release (Appendix D.1). *Data-type direction:* the dominated table loads after the common rescaling c that a per-table scale absorbs, e.g. $\frac{1}{4} \cdot \text{INT6} \subset$ E2M5 (Section 3.3).

A.4 Encoding auxiliary information into the CPTC bitstream

This appendix develops the in-stream metadata thread of Section 3.6. CPTC's encoder is free to choose, among all bit sequences that decode to a low-MSE reproduction of the input, one whose bits also satisfy a chosen linear constraint. This lets the encoder commit auxiliary metadata (a per-block scale, a zero point, an exponent, a flag) *into the CPTC codeword itself*, with no separate storage and no extra encoder time.

Construction. Commit auxiliary bit b_j at the boundary of window j of g values by constraining the parity (XOR) of that window's gK emitted code bits to equal b_j. The constraint is enforced inside Viterbi without enlarging the state space: track the cumulative parity as one extra bit alongside each state's accumulated cost, and at every window boundary set the cost of any state whose recorded parity disagrees with the target bit to $+\infty$. Subsequent steps propagate only consistent paths, so the encoder stays $O(T \cdot 2^L)$, the same complexity as auxiliary-free CPTC. This is a heuristic rather than the exact constrained optimum, which would double the state space with the parity bit: when two paths of different parity merge into one state, only the survivor's parity is kept. The measured cost below includes whatever optimality this forfeits.

Rate cost. Each auxiliary bit fixes one linear constraint on the gK code bits committed to it, removing one degree of freedom from the encoder's search space: $1/g$ effective bits per value per auxiliary bit. The canonical use is the per-block exponent of an MX-style format (Rouhani et al., 2023): at L=16, K=4, g=64, an 8-bit scale costs $8/64 = 0.125$ effective bits per value in theory and 0.132 measured, leaving CPTC at K=4 with an in-stream scale at 3.80 effective bits, still well above the 3.34–3.36 ceiling of NF4 and the Lloyd-Max scalar. The same construction carries per-block zero points (asymmetric quantization), per-block format flags, or any other small-bit-budget metadata.

B THEORY

B.1 INDEX-LAW ABLATION: UNIFORM VS. TRIANGULAR COMPOSITION

This appendix quantifies the claim of Section 3.5 that composing the table with the (near-triangular) index CDF, rather than assuming a uniform index law, is what lets a small table retain trellis gain. Figure 3 shows the two placements and the two laws side by side.

Relation to 1MAD and LC. QTIP's 1MAD also sums equidistributed quantities (hashed bytes) to obtain a bell-shaped law, but it uses the sum *as the value*: the decoded number is the sum itself, so the reproduction distribution is whatever the central-limit sum gives and cannot be retargeted. CPTC uses the sum as an *index* into a table composed with that sum's law (Equation (2)), which is a companding rule: the law determines where codepoints are dense and the table determines what they are. That separation is what lets a 64-byte table follow a Gaussian, a generalized Gaussian or a bimodal matmul-aware target (Appendices C.2 and C.5), and what LC and single-multiply CPTC lack, since one congruential map induces a near-uniform index law (Proposition 3, Appendix B.3).

Codebook level. Table 7 reports exact scalar nearest-codepoint MSE on $\mathcal{N}(0,1)$ for M-point codebooks $v_i = F^{-1}\big((G(i{-}1) + G(i))/2\big)$, the midpoint quantile of each cell of the index law's CDF G, under three index laws: uniform (equal probability mass), symmetric triangular, and the Lloyd-Max optimum (Lloyd, 1982; Max, 1960) as the reference. Triangular tracks Lloyd-Max within 7–17% at every size, while uniform degrades from $2.2\times$ (at M=15) to $29\times$ (at M=511) worse than triangular. The mechanism is overload: the uniform law's largest codepoint sits at the $\frac{M-0.5}{M}$ sample quantile (2.66 at M=127), and the two unbounded tail cells contribute 83% of its MSE; the triangular law reaches 3.67 (tail share 16%) and Lloyd-Max 4.20 (tail share 2%). Because overload decays only polynomially in M while granular error decays as M^{-2} (Panter & Dite, 1951; Bennett, 1948), the uniform law also loses the *rate*: doubling M buys only $\sim 2.3\times$ instead of $\sim 4\times$. Matching the 127-entry triangular codebook with a uniform-composed one takes $M > 511$: the composition is worth more than $4\times$ in table size.

Beyond the Gaussian. The same comparison at M=127 (all sources unit variance): uniform-over-triangular MSE ratio is $9.6\times$ for $\mathcal{N}(0,1)$, $15.6\times$ for Laplacian, and $7.4\times$ for Student-t_5. Heavier tails make the uniform law's truncation more costly, not less. The composition benefit is a property of mean-concentrated targets generally, not of the Gaussian.

End-to-end trellis ablation. Table 8 repeats the comparison inside the full CPTC pipeline (L=16, K=4, T=128, the paper's (a_0, a_1), Viterbi encoding of 2^{22} i.i.d. $\mathcal{N}(0,1)$ samples), varying only how the table is populated. Two conclusions. First, the uniform law costs 0.72 effective bits ($3.92 \rightarrow$

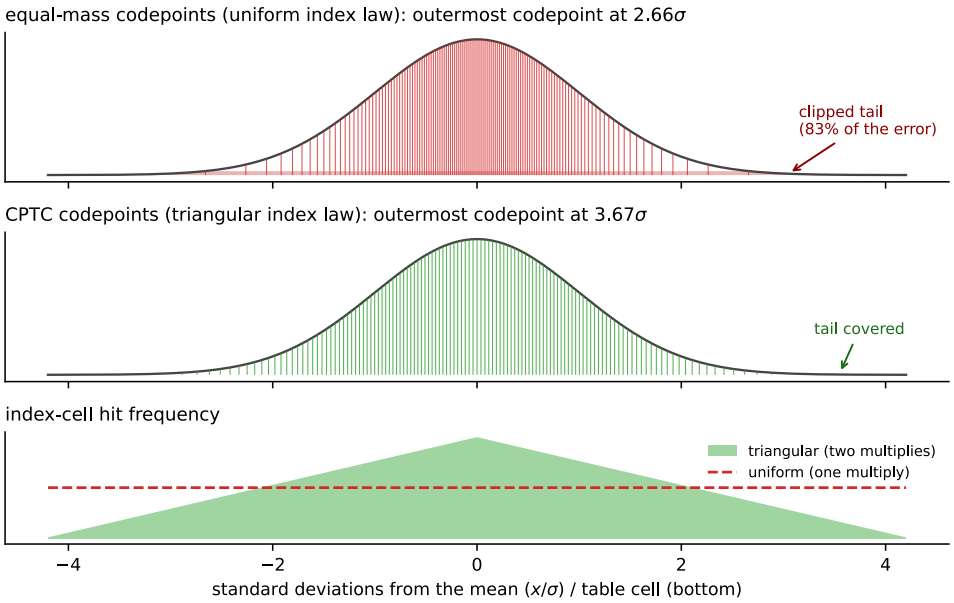


Figure 3: Where the two index laws put their codepoints (M=127, standard normal). **Top:** equal-mass placement from a uniform index law, red marking the clipped tail. **Middle:** the triangular law's placement. **Bottom:** the two index laws themselves.

Table 7: Exact MSE (↓) of M-point Gaussian codebooks built from a uniform vs. a triangular index law, against the Lloyd-Max optimum.

M	Uniform	Triangular	Lloyd-Max	$\frac{\text{Unif}}{\text{Tri}}$	$\frac{\text{Tri}}{\text{LM}}$
15	2.52e-2	1.14e-2	1.07e-2	2.2	1.07
31	1.00e-2	2.90e-3	2.66e-3	3.5	1.09
63	4.17e-3	7.37e-4	6.65e-4	5.7	1.11
127	1.79e-3	1.88e-4	1.66e-4	9.6	1.13
255	7.85e-4	4.77e-5	4.15e-5	16.5	1.15
511	3.50e-4	1.21e-5	1.04e-5	28.9	1.17

3.20) and lands *below the Lloyd-Max scalar* (3.36): the trellis cannot route around a missing tail codepoint, so the ∼1% of samples in the clipped tails carry an error the encoder cannot influence, and the trellis gain is spent on it. Second, replacing the closed-form composed table with an oracle Lloyd-Max table of the same size changes nothing (3.917 vs. 3.919): Equation (2) already sets each entry to the conditional mean of its cell, the Lloyd centroid condition for the index-induced partition, and the ablation shows the partition itself is also near-optimal, with no iterative codebook tuning.

Table 8: Effective bits (↑) of the full CPTC pipeline when only the law that fills the table changes (L=16, K=4, T=128, 2^{22} samples).

Effective entries	Table law	K_{eff}
127 (s=0)	Uniform composition	3.203
127 (s=0)	Lloyd-Max oracle	3.917
127 (s=0)	Triangular composition (ours)	**3.919**
63 (s=1)	Uniform composition	3.196
63 (s=1)	Triangular composition (ours)	**3.906**
31 (s=2)	Uniform composition	3.049
31 (s=2)	Triangular composition (ours)	**3.822**

The uniform-state approximation. One approximation in the composition argument deserves naming: the rule of Equation (2) matches the codebook to the index distribution *under uniformly random states*, whereas the Viterbi encoder visits states non-uniformly, so the scalar-MSE analysis is a proxy for the trellis-coded objective. Empirically the proxy is tight: the encoder-visited and uniform-state index distributions differ by only 0.005 in total variation at $L{=}16$, $K{=}4$ (measured in Section 4.1's protocol), and the oracle-table ablation above confirms no quality is left on the table.

Would a third multiplication help? The sum of three permutation images has an Irwin-Hall index law whose composed codepoint density decays like the Panter-Dite optimum $\lambda^\star \propto p^{1/3}$ (Panter & Dite, 1951). At the table sizes CPTC uses, however, it is numerically indistinguishable from triangular: across $M \in [31, 511]$ the Irwin-Hall-composed codebook is 1.12–$1.16\times$ the Lloyd-Max MSE where triangular is 1.09–$1.17\times$. The asymptotic advantage never materializes below $M \approx 512$, so the third multiplication would add decode cost for no measurable quality.

Visual check. Figure 4 plots the quantiles of the composed $L{=}16$ decoded-state codebook against $\mathcal{N}(0,1)$ quantiles: the 2^{16} state values track the target distribution closely across the full support, including the $> 3\sigma$ tails that the uniform composition truncates.

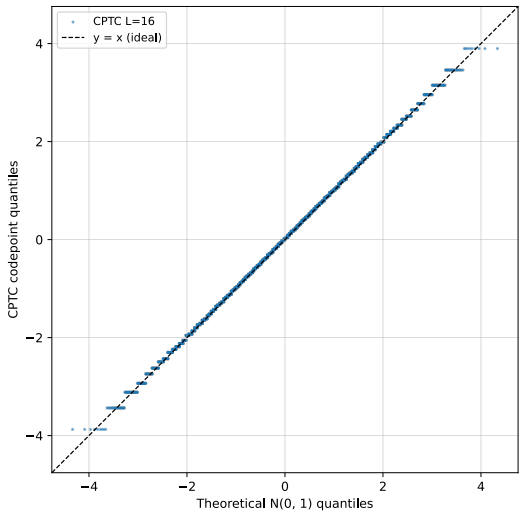


Figure 4: Quantile-quantile plot of CPTC's $L{=}16$ decoded values against a standard normal; a perfect match lies on $y = x$.

B.2 Alphabet ceilings: what the table can and cannot cost

CPTC's decoder emits values from a fixed table, 127 entries at $Q{=}6$. Proposition 4 bounds what such a table can achieve when it has at most 2^K values. This appendix gives the general form of that bound (a restatement of the classical alphabet-constrained rate-distortion function, Berger, 1971), computes it for every codebook in Table 2, and uses it to separate the two sources of CPTC's gap to the Shannon limit: the table and the finite trellis depth.

Theorem 1 (Alphabet ceiling)**.** *Let $X_1, \dots, X_n$ be i.i.d. with finite variance and let a code of rate K bits per value (any block length, any encoder) reproduce each X_i by some $\hat{X}_i \in C$, where $C \subset \mathbb{R}$ is a fixed finite set. Then*

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{E}(X_i - \hat{X}_i)^2 \;\geq\; D_C(K) \coloneqq \min\big\{\mathbb{E}(X - \hat{X})^2 :\; \hat{X} \in C,\; I(X;\hat{X}) \leq K\big\},$$

the alphabet-constrained rate-distortion function. In particular $K_{\mathit{eff}} \leq K_C(K) \coloneqq -\frac{1}{2}\log_2\big(D_C(K)/N(X)\big)$ for every trellis depth L and block size T, and $D_C(K) \geq N(X)\,4^{-K}$, with $N(X) \leq \sigma^2$ the entropy power of Equation (1) *(equality for the Gaussian).*

Proof. Let B be the nK-bit codeword, so $I(X^n;\hat{X}^n) \le I(X^n;B) \le H(B) \le nK$. Since the X_i are independent and $\hat{X}^n$ is a function of B, $I(X^n;\hat{X}^n) \ge \sum_i I(X_i;\hat{X}_i)$ (independence and the chain rule; Berger, 1971). Each $\hat{X}_i$ lies in C, so $I(X_i;\hat{X}_i) \ge R_C(D_i)$ with $D_i = \mathbb{E}(X_i - \hat{X}_i)^2$ and R_C the rate-distortion function with reproduction alphabet C, which is convex and nonincreasing because the alphabet constraint is per letter and preserved under time sharing. Hence $nK \ge \sum_i R_C(D_i) \ge nR_C(\bar{D})$, i.e. $\bar{D} \ge D_C(K)$. Dropping the alphabet constraint gives the Shannon bound. □

The ceiling is tight in the limit: for a memoryless source and a finite reproduction alphabet, time-varying trellis codes with random labels drawn from C approach $D_C(K)$ as the constraint length grows (Viterbi & Omura, 1974), so $K_C(K)$ is the exact $L \to \infty$ asymptote of a codebook's alphabet under random labeling, not only a bound. It is also computable: $D_C(K)$ is a convex program over the output law on C, and the Blahut-Arimoto iteration (Blahut, 1972) converges to it. We discretize $\mathcal{N}(0,1)$ on 17,001 points of $[-8.5, 8.5]$ (step 0.001, far below every codepoint spacing) and bisect the slope until the rate equals K to 10^{-4} bits. As a check, the 16-point Lloyd-Max alphabet at K=4 gives $K_C = 3.3589$, the scalar Lloyd-Max value to four decimals, as it must (with $|C| = 2^K$ the optimal assignment is deterministic). For HYB the alphabet is a set of value pairs, so the same computation runs in two dimensions at rate $2K$ per pair on a 401×401 grid over $[-6, 6]^2$, with the pair table normalized to unit variance as in Table 2.

Table 9: The most effective bits (↑) any code of rate K can reach with a given reproduction alphabet, at any trellis depth (Theorem 1, standard normal). Measured L=16 rows repeat Table 2. HYB's alphabet is 512 k-means pairs with a sign flip on the first component (1024 pairs at rate $2K$ per pair).

Alphabet	$\lvert C\rvert$	K=1	K=2	K=3	K=4	K=5	K=6
Shannon limit	∞	1.000	2.000	3.000	4.000	5.000	6.000
Lloyd-Max optimal 127-point table	127	1.000	2.000	3.000	3.999	4.993	5.930
CPTC per-K table, exact values	127	1.000	2.000	2.999	3.997	4.981	5.886
CPTC per-K table, E2M5 (deployed)	127	1.000	2.000	2.999	3.997	4.976	5.832
CPTC measured, L=16		0.935	1.948	2.941	3.929	4.895	5.759
Uniform law (one multiply), midpoints	127	1.000	1.995	2.955	3.786	4.316	4.521
Uniform law (one multiply), midpoints	64	1.000	1.987	2.898	3.577	3.894	3.966
QTIP HYB pair table	1024 pairs	1.000	1.994	2.944	3.687	3.936	3.936
HYB measured, L=16		0.932	1.912	2.840	3.611	3.936	3.936

What the table costs. Every measured value in Table 9 lies below its ceiling, as the theorem requires, and the ceilings split CPTC's gap to Shannon into a table part ($K - K_C$) and a depth part (K_C- measured). At $K \le 4$ the deployed 64-byte table costs at most 0.003 bits at *any* depth; the entire 0.05–0.07-bit gap of Table 2 is trellis depth, which deeper trellises or better coefficients could recover and no table change can. At K=5 the table costs 0.024 of the 0.105-bit gap, at K=6 it costs 0.168 of 0.241, of which E2M5 rounding is 0.054 (exact vs. E2M5 rows), the rigorous form of the alphabet floor that Section 4.1 and Appendix C.3 observe. The Lloyd-Max 127-point row shows what the best possible 127-value table could reach: the triangular placement gives up nothing measurable through K=4 and 0.04–0.10 bits at K=6. For HYB the ceiling saturates at 3.936 from K=5 on (the rate constraint stops binding once it exceeds the alphabet's nearest-neighbor entropy of 9.92 bits per pair), and the measured plateau is 3.936: HYB's fixed 512-pair alphabet is provably what caps it past K=4, and no trellis depth can lift it.

One multiplication, scale, and depth. The uniform-law rows quantify the tail argument of Section 3.5. With the equal-mass composition at its natural scale the extreme codepoint sits at the quantile $1 - 1/(2M)$, and the overload error alone bounds $K_{\text{eff}} \le \frac{1}{2}\log_2(M \ln M) + o(1)$ regardless of K or L (the exact overload integral gives 4.16 at M=64 and 4.74 at M=127; the closed form alone 4.03 and 4.63), while the two-tap law places the extreme cell at mass 2^{-2Q}, an overload of 2.8×10^{-5} at Q=6, below the Shannon distortion for every $K \le 7$. This floor is a property of the composition, not of the multiplication count: stretching the 127-point uniform-law table by a factor κ raises its ceiling at K=4 from 3.82 at κ=1 (conditional-mean entries; 3.79 with quantile

midpoints, Table 9) to 3.92 (κ=1.1) and 3.98 (κ=1.3). What remains for a one-multiply table is a finite-depth cost (Table 10): with a tuned stretch it recovers from 3.78 to 3.88 at K=4 and from 4.37 to 4.80 at K=5, while the two-multiply table is scale-free (+0.005 and +0.012 from tuning) and still leads by 0.06 and 0.12 bits at matched table size (128 vs. 127 cells), and by 0.11 and 0.24 at the 64-byte point. One multiplication can protect the tails, but only with a tuned scale, and then trails at finite depth; the triangular law needs no tuning.

Table 10: Effective bits (↑) at L=16 when the table is stretched by a factor κ: one-multiply tables (best of 16 random odd multipliers, chosen at κ=1 on held-out data) against the two-multiply CPTC table with exact values (T=256, 3 seeds).

Table	K	κ=1.0	1.05	1.1	1.2	1.3	1.4
One multiply, 64 cells	4	3.612	3.697	3.755	3.817	**3.825**	3.802
One multiply, 128 cells	4	3.777	3.832	3.862	**3.878**	3.858	3.820
Two multiplies (CPTC), 127 cells	4	3.929	**3.934**	3.931	3.908	3.871	3.826
One multiply, 64 cells	5	3.986	4.154	4.292	4.502	4.622	**4.672**
One multiply, 128 cells	5	4.374	4.526	4.629	4.760	**4.796**	4.790
Two multiplies (CPTC), 127 cells	5	4.904	**4.916**	**4.916**	4.898	4.863	4.818

The index law is a lattice. The two taps have an exact description that explains the coefficient damage law of Appendix A.2.

Lemma 1. *For odd a, $q \mapsto aq \bmod 2^L$ is a bijection of $\mathbb{Z}_{2^L}$; more generally, if $\nu_2(a) \le L - Q$ (true of every pair in Table 5 once the 2^{16-L} normalization is removed) it covers the multiples of $2^{\nu_2(a)}$ uniformly. In both cases each tap $t_j(q) = \lfloor (a_j q \bmod 2^L)/2^{L-Q} \rfloor$ is exactly uniform on $\{0, \dots, 2^Q - 1\}$. The pairs $(t_0(q), t_1(q))$ are the cells hit by the two-dimensional rank-1 lattice $P = \{(a_0 q, a_1 q)/2^L \bmod 1 : q \in \mathbb{Z}_{2^L}\}$, and the index law $i = t_0 + t_1$ is exactly triangular if and only if P puts 2^{L-2Q} points in every cell of the $2^Q \times 2^Q$ grid. In general $|N_{ij} - 2^{L-2Q}| \le 2^L D^*(P)$, and the star discrepancy of a rank-1 lattice is $O\big((\log 2^L)^2/\rho\big)$ (Niederreiter, 1992, Ch. 5), where $\rho = \min \max(1, |h_0|) \max(1, |h_1|)$ over nonzero integer (h_0, h_1) with $h_0 a_0 + h_1 a_1 \equiv 0 \pmod{2^L}$.*

The first two statements are immediate; the last is the classical discrepancy bound for lattice rules. The pairs the damage law rejects are exactly those with a short dual-lattice vector (small ρ): $a_1 = -a_0$ has $\rho = 1$ and $a_1 = 3a_0$ has $\rho = 3$, and such pairs leave most of the $2^Q \times 2^Q$ cells empty at L=16, while every shipped pair hits all cells within 7 of the flat count of 16. The lemma gives a Viterbi-free screen, since ρ is computable by two-dimensional lattice reduction, but it does not rank the good pairs: the coefficient search selects for trellis quality, about which the index law says nothing, and that is the content of Conjecture 1. In sum: the table costs nothing measurable through K=4 and binds only at $K \ge 5$, the one-multiply alternative needs a tuned scale and still trails at finite depth, and the two-tap index law is a lattice whose discrepancy screens bad coefficient pairs but does not rank good ones.

B.3 Achievable rate and the limit $L \to \infty$

The classical achievability results are Viterbi & Omura (1974) (time-varying trellises with random branch labels achieve the rate-distortion function of an i.i.d. source) and Gray (1977) (a non-constructive existence proof for time-invariant trellis codes). Mao et al. (2011) complement these with *necessary* conditions on every asymptotically optimal stationary code (the marginal reproduction distribution converges to a Shannon-optimal one; finite-order joints factor; the reproduction whitens; the code bits approach fair coins) and prove that RPTC satisfies the marginal condition. Their proof uses one property of the random permutation, bijectivity, so it transfers to CPTC deterministically, for every coefficient choice:

Proposition 3 (Reduction and marginal optimality)**.** *With $a_0 = 0$, odd $a_1 = a$, $Q = L$, $s = 0$, and $\mathcal{T}$ populated by Equation* (2)*, CPTC is RPTC with the specific permutation $P_a(q) = aq \bmod 2^L$ in place of a random one: P_a is a bijection for every odd a, and the decoder computes $\phi(q) = \mathcal{T}[P_a(q)]$. Consequently, for any target with CDF F and finite variance, the decoded-state marginal under uniform states is the exact 2^L-point quantile discretization of F, independent of a, and con-*

verges weakly (and in Wasserstein-2) to the target as $L \to \infty$, so CPTC satisfies the necessary marginal condition of Mao et al. (2011) for every coefficient choice.

The condition is necessary, not sufficient, and does not discriminate among coefficients: the identity permutation ($a = 1$) satisfies it and yields a poor code (Appendix A.2). It concerns the *optimal reproduction* distribution, $\mathcal{N}(0, \sigma^2 - D)$ for Gaussian MSE, not the source; since Proposition 3 holds for any loaded F, the table can be composed to the test channel, and the paper's source-composed tables are numerically indistinguishable at our rates ($\sigma^2 - D \approx 0.996$ at $K{=}4$). The same reduction places single-multiplication CPTC inside the linear congruential trellis family of Eriksson et al. (2007): P_a commutes with the bitshift dynamics, $P_a((q \ll K) \mid u) = (P_a(q) \ll K) + au \bmod 2^L$, so the permuted state follows their generating LC recursion and the destination-state labels realize their affine congruential label rule. In neither family is a decode-time recursion implied; for CPTC the sequential update (Equation (5)) is an optional optimization.

What neither family has is an achievability theorem: the joint conditions are open even for RPTC: Mao et al. (2011) anticipate "a source coding theorem using random coding ideas", and none has appeared. We state the CPTC form as a conjecture:

Conjecture 1. *For each target distribution symmetric about zero with finite variance σ^2 and a bounded, continuous density, there exist sequences of CPTC parameters $(a_0^{(L)}, a_1^{(L)}, \mathcal{T}^{(L)})$ with coefficients modulo 2^L and table size $Q(L) \to \infty$ such that the induced distortion converges to $\sigma^2 \cdot 4^{-K}$ as $L \to \infty$. ($Q(L) \to \infty$ is necessary: a fixed reproduction alphabet has a rate-distortion function strictly above the Shannon bound; Theorem 1 computes that function and Proposition 4 is its quantizer-level form.)*

By Proposition 3 the marginal condition is met, so the conjecture is entirely about the joint conditions, i.e. the pseudorandomness of the index sequence along trellis paths. Two facts locate the difficulty. The index pair $(i(q), i(q'))$ over a uniformly random odd coefficient pair is empirically close to independent triangular-triangular for all but a vanishing fraction of state pairs (total variation ≈ 0.01 at $L{=}12$–14, $Q{=}4$, by exact enumeration), the exceptions being pairs related by small-height dyadic rationals, the multiplicative analogue of multiply-shift hashing's collision structure (Dietzfelbinger et al., 1997); an average-case theorem would need to show that these structured pairs, which include the shift-related states consecutive on every path, contribute vanishing excess distortion. And the exhaustive evaluation of Appendix A.2 shows the right target is a *concentration* statement: outside the damage set, distortion concentrates on a plateau whose max-to-median gap shrinks $0.11 \to 0.07 \to 0.04$ bits from $L{=}8$ to 12. Proving that the generic surviving pair is good and that the plateau persists as $L \to \infty$ is the open problem. All experimental results are at finite $L \leq 16$, where the per-L match with RPTC (Table 11) is direct evidence and the conjecture is not needed.

How fast $Q(L)$ must grow. The distortion of a finite-(L, Q) coder splits into the loss of the trellis coder with a continuous codebook, which needs no table and converges to $\sigma^2 4^{-K}$ under Conjecture 1, plus the granular error of rounding that codebook to $M = 2^{Q+1} - 1$ table values, about $2.72\,(\sigma^2 - D)/M^2$ (Panter & Dite, 1951). With $Q = K + C$ the second term is a K_{eff} gap of roughly $\frac{1}{2}4^{-C}$ bits, independent of K: 0.12 bits at $C{=}1$, 0.03 at $C{=}2$, 0.008 at $C{=}3$. So $Q = K + 2$ already lands within 0.03 bits of the limit, and growth as slow as $Q(L) = K + \Theta(\log L)$ drives the gap to zero. The estimate is an upper one: the minimal-codebook experiment (Table 15), the $C{=}1$ instance, measures 0.02–0.04 bits, and the rigorous ceiling of Table 9 at $C{=}2$ is 0.003, because the trellis visits the composed codepoints non-uniformly and rounding them to 127 levels costs less than the granular formula assumes; Table 16's FP16 row isolates the first term empirically. In sum: the marginal optimality condition is proved for every coefficient choice, the joint conditions are the open conjecture, and the table-size term of the gap is small and independent of K.

C Extended results

C.1 Extended i.i.d. source-quantization results

This appendix collects the i.i.d. results that Section 4.1 summarizes: the trellis-depth sweep behind Figure 1a (Table 11), the block-scaled scalar formats, the RPTC search-budget control, the LC baseline construction with the headline-margin precision, and the table-size sweep.

Table 11: Effective bits (↑) at each trellis depth L for K=4 (T=256, 2^{22} samples). RPTC is our reimplementation, values at the quantiles of a random permutation, under the same encoder and protocol; the QTIP entry runs the authors' 1MAD code the same way (Table 2). CPTC's 16-bit coefficients stop its family at L=16, where it already matches RPTC at L=18. Every trellis code stores exactly K bits per value; scalar formats stop at 3.36 (Lloyd-Max; INT4 3.15, FP4 3.16, NF4 3.34) and the limit is 4.00.

L	8	9	10	11	12	13	14	15	16	17	18
QTIP	—	—	—	—	—	—	—	—	3.86	—	—
RPTC	3.46	3.55	3.65	3.72	3.78	3.82	3.85	3.88	3.90	3.91	3.92
CPTC	3.65	3.73	3.79	3.84	3.87	3.89	3.91	3.92	**3.93**	—	—

Block-scaled scalar formats. The 4-bit formats hardware executes natively attach one 8-bit scale to each group of G values: MXFP4 (Rouhani et al., 2023) an E8M0 (power-of-two) scale with G=32, 4.25 bits per value, and NVFP4 an E4M3 scale with G=16, 4.5 bits. Table 12 scores both on the standard normal under the protocol of Section 4.1 (2^{22} samples, round-to-nearest, the accelerator's bit-exact microscaling library, each format's own per-group max-abs scale rule: the shared E8M0 exponent of the group maximum for MX, an E4M3 scale for NVFP4), together with the INT4 element type and the 6-bit variants. A power-of-two scale buys nothing on the whitened i.i.d. source (MXFP4 equals per-tensor FP4 to three digits); an E4M3 scale acts as a mild compander, worth 0.1–0.4 bits, but costs 0.5 bits of rate. CPTC carries a block scale without leaving its K bits per value: the in-stream 8-bit scale per 64 of Appendix A.4 keeps 3.80 effective bits at exactly 4 bits, 0.4 above NVFP4 at 4.5. Block scaling earns its keep on unrotated weights with outliers, which the RHT removes, so these are statements about the i.i.d. regime only.

Table 12: Block-scaled scalar formats on a standard normal: element type, scale type and group size G, total bits per value including the scale, and effective bits (↑; standard protocol, round-to-nearest, bit-exact microscaling library). Reference rows repeat Table 2 and the scalar ceilings.

Element	Scale	G	bits/value	K_{eff}	
FP4 E2M1	E8M0	32	4.25	3.16	MXFP4
FP4 E2M1	E8M0	16	4.50	3.18	
FP4 E2M1	E4M3	32	4.25	3.30	
FP4 E2M1	E4M3	16	4.50	3.39	NVFP4
INT4	E8M0	32	4.25	2.86	
INT4	E4M3	16	4.50	3.53	
FP6 E2M3	E8M0	32	6.25	5.15	MXFP6
FP6 E2M3	E4M3	16	6.50	5.33	
FP4 E2M1	per-tensor	—	4.00	3.16	
Lloyd-Max	per-tensor	—	4.00	3.36	scalar optimum
CPTC K=4	in-stream 8-bit	64	4.00	3.80	Appendix A.4
CPTC K=4	per-tensor	—	4.00	3.93	
CPTC K=6	per-tensor	—	6.00	5.76	

Is CPTC's edge just search budget? The trellis-depth sweep of Table 11 (plotted as Figure 1a) shows CPTC's searched coefficients beating our RPTC reimplementation at every L. CPTC's coefficients are the best of an exhaustive search, while each RPTC number is a random draw, so we gave RPTC a search too: the best of 100 random permutations on a held-out 2^{20}-sample selection set, evaluated on the standard 2^{22}-sample set. Selection helps far less than the structured family's edge: best-of-100 RPTC reaches 3.49 at L=8 (vs. 3.46 single-draw, 3.65 CPTC), 3.78 at L=12 (vs. 3.78, 3.87), and 3.90 at L=16 (vs. 3.90, 3.93). Random-permutation distortion concentrates tightly (worst-of-100 to best-of-100 spans under 0.04 bits at every L), so no realistic search budget closes the gap: the modular-multiplicative family contains permutations that random draws essentially never produce. Extending the reimplementation to $L \in \{17, 18\}$ gives the RPTC L=17/18 entries of Table 11 (3.91 and 3.92), so CPTC at L=16 (3.93) matches RPTC two state bits deeper. The structured family's edge has no theorem yet; Lemma 1 gives the index-law part of the picture.

The LC baseline and headline-margin precision. The LC row of Table 2 reports, per rate, the stronger of two searched single-multiplication constructions, both run under the standard protocol with our tail-biting encoder. *(i) Axiom SR/LC*, the construction of Eriksson et al. (2007): $M_S = 2^{16-K}$ shift-register states, per-branch labels $l_i^{(u)} = (g \cdot i + a^{(u)}) \bmod M_L$ with $M_L = M_S$, one shared odd generator and maximally separated offsets $a^{(u)} = a_0 + uM_L/2^K$ (their Theorems 1–2, extended to $K \geq 3$), label-quantile reproducers (their Eq. 17), best of 150 random (g, a_0) on a held-out seed. It wins the row at $K \leq 4$ (0.940/1.950/2.942/3.931, ± 0.0007 over 10 seeds). *(ii) Full-codebook computed permutation* (Proposition 3: $a_0 = 0$, $P_a(q) = aq \bmod 2^{16}$, 2^{16}-entry quantile codebook, best of 200 odd multipliers per rate), a strengthening beyond their paper; it wins at K=5 (4.919) and K=6 (5.908, where the axiom form collapses to 5.62). Their trained scale factor and Lloyd refinement (<0.04 dB for Gaussians in their Table II) are omitted, matching the other quantile arms, so the row if anything understates them. At matched 64-byte storage the axiom construction collapses at high rates (2.915/3.781/4.367 at K=3/4/5): its single congruential map induces a near-uniform index law (Appendix B.1). For the headline margins we also report 10-seed precision: CPTC L=16, K=4 scores 3.9287 ± 0.0005 and RPTC L=18, K=4 scores 3.9231 ± 0.0004, a 0.006-bit margin an order of magnitude above seed noise. At K=1 the searched pair reaches 0.935–0.936 vs. RPTC's 0.9400 ± 0.0004, one of the two cells where a random permutation wins (the other is K=6, 5.840 vs. 5.759 in Table 2).

Effect of the table-size parameter. Table 13 varies the effective table-size parameter Q_{eff} across K (per-K coefficients of Table 5, L=16, standard protocol, 16-bit table entries to isolate the size effect from the storage dtype, which is why its Q_{eff}=6 row reads above Table 2's E2M5 CPTC row at K=5; Appendix C.3). Rows $Q_{\text{eff}} < 6$ are served from the fixed Q=6 implementation via the shift $s = 6 - Q_{\text{eff}}$ (Section 3.1); the $Q_{\text{eff}} = 7$ row is a doubled-table control requiring Q=7. The pattern generalizes Table 15: shrinking is nearly free while $Q_{\text{eff}} \geq K+1$ (at most 0.012 bits down to that point at every K), the last allowed halving $Q_{\text{eff}} = K$ costs 0.05–0.09 bits, and below it the alphabet floor of Proposition 4 binds hard, visible as rows going flat in K (e.g. 2.247 for all $K \geq 3$ at $Q_{\text{eff}} = 2$). Doubling *above* the shipped size buys at most 0.008 bits. (At K=1 tiny tables are marginally *better*: 0.942 at $Q_{\text{eff}} = 2$, edging RPTC's 0.940.)

Table 13: Effective bits (↑) as the table shrinks: Q_{eff} gives $2^{Q_{\text{eff}}+1}-1$ entries (L=16, per-K coefficients, 16-bit entries, 3 seeds).

Q_{eff}	Entries	K=1	K=2	K=3	K=4	K=5
7	255	0.935	1.948	2.941	3.931	4.912
6	127	0.936	1.948	2.941	3.929	4.904
5	63	0.936	1.948	2.939	3.917	4.829
4	31	0.936	1.947	2.929	3.843	4.245
3	15	0.937	1.943	2.865	3.249	3.252
2	7	0.942	1.894	2.247	2.247	2.247

Lattice VQ baselines (Table 1). The E8P and nested-E8 entries are our runs of the authors' released code on the protocol of Table 2 (2^{22} samples, three seeds). QuIP#'s E8P is its 2-bit codebook (256 absolute-value entries with sign flips and a $\pm\frac{1}{4}$ shift, 2^{16} points in 8 dimensions); its 3- and 4-bit codebooks quantize the residual with a second, rescaled E8P stage. Sweeping the input and residual scales around the released values: K=2 gives 1.727 at the released scale (0.0913 MSE, against the 0.089 Tseng et al. (2024b) report for the same codebook), K=3 2.541, and K=4 3.444 (3.351 at the released scales). NestQuant's nested Gosset code maps each 8-vector to the E8 closest point, reduces modulo q (q=16 at 4 bits) and decodes with a second closest-point search; its encoder picks one of k scales β per block, a side channel of $\log_2 k/8$ bits per value, under a shared pseudo-random dither. With a single scale (exactly 4 bits, β swept, overload rate 10^{-4}) it reaches 3.03 effective bits, below the Lloyd-Max scalar ceiling, because one scale's Voronoi shaping region must enclose the Gaussian tails; with k=2 and k=4 scales chosen by exhaustive subset search it reaches 3.60 at 4.125 bits and 3.89 at 4.25 bits, deficits of 0.53 and 0.36 bits against its own rate. At 2 bits (q=4) the same runs give 1.09, 1.54 (2.125 bits) and 1.77 (2.25 bits), against 1.727 for E8P and 1.948 for CPTC at exactly 2 bits. The Leech-lattice codebook of van der Ouderaa et al. (2026)

has no released implementation; its paper reports 0.078 MSE at 2 bits (1.84 effective bits) and no 4-bit figure. Scripts: `e8p_keff.py`, `nestquant_keff.py`.

C.2 Generalized-Gaussian targets: normalization and extended results

This appendix supports the beyond-the-Gaussian experiment of Section 4.1 (Table 3): the entropy-power normalization and its verified tightness, the $K{=}4$ method comparison (Table 14, including the RPTC and equal-mass-composition columns quoted there, and the uniform source as its $\beta{=}\infty$ row), and additional observations. For each β the table is built with the standard recipe of Equation (2) (empirical index intervals, per-cell conditional mean under p_β by numerical integration, since only $\beta{=}2$ has a closed form, then rounding to E2M5), with the same (a_0, a_1) and encoder throughout.

Table 14: Effective bits (↑) at $K{=}4$, $L{=}16$ on generalized-Gaussian targets, entropy-power normalized so the limit is 4.00 in every row ($T{=}256$, 3 seeds, std below 0.01). "Equal-mass table" is the uniform-composition ablation of Appendix B.1, not a uniform target; RPTC uses a per-β quantile codebook under the same encoder and protocol (Appendix C.1, 3 permutation seeds).

β	Lloyd-Max scalar	CPTC equal-mass table	RPTC	CPTC (E2M5)	CPTC (FP16)
∞ (uniform)	3.75	3.53	3.92	3.94	3.94
2.0 (normal)	3.36	3.22	3.90	3.93	3.93
1.5	3.21	3.07	3.88	3.91	3.91
1.0 (Laplacian)	2.91	2.74	3.83	3.84	3.84

Normalization and its tightness. For the unit-variance generalized Gaussian the differential entropy has the closed form $h(X) = \beta^{-1} + \ln\big(2\alpha\Gamma(1/\beta)/\beta\big)$ with $\alpha = \sqrt{\Gamma(1/\beta)/\Gamma(3/\beta)}$ (Nadarajah, 2005), which is what Equation (1)'s $N(X)$ uses; as $\beta \to \infty$ it gives the uniform's $N(X) = 12/(2\pi e) \approx 0.703$, the classical quarter-bit advantage of the uniform source over the Gaussian at equal variance. For $\beta \neq 2$ the Shannon ceiling is not exactly achievable at finite rate, but nearly so at our rates: bisecting Equation (1)'s upper end for the distortion it certifies achievable (with $N(X + Z_D)$ by FFT convolution) places the true limit at $K{=}4$ above 3.995 effective bits across $\beta \in [1, 2]$ (above 2.98 at $K{=}3$) and above 3.95 for the uniform, whose discontinuous density slows the bound's convergence. The gaps in Table 14 are therefore genuine coding and table loss, not slack in the ceiling.

Further observations on Table 14. On the tail-free end scalar quantization is already near-optimal (Lloyd-Max for the uniform *is* the uniform quantizer), which is why the trellis gain is smallest at $\beta{=}\infty$; there a single-multiplication variant ($a_0 = 0$, uniform index law over 64 cells) matches the two-multiplication decoder through $K \leq 5$ and cedes the trellis gain only at $K{=}6$ (5.74 vs. 5.90 in FP16), where its 64 codepoints equal 2^K exactly. The E2M5 table is not a bottleneck if its scale is calibrated: a per-β scale chosen on one held-out seed keeps the rounded table within 0.01 bits of the FP16 one at every β. The distance to the 4.00 limit widens moderately with tail weight (0.07 bits at $\beta{=}2$ vs. 0.16 at $\beta{=}1$): the more sharply peaked source rewards codes that exploit its low entropy, which a fixed 127-codepoint symmetric table follows only partially (Appendix B.1).

C.3 Effect of the table data type

This appendix supports the table-data-type knob of Section 4.1 (Table 15) and proves the alphabet floor used there. The 8-bit table entries are themselves quantized codepoints, so the table data type sets a second floor on quality: rounding the composed codepoints of Equation (2) to the nearest representable value perturbs each by up to half an output unit in the last place. Table 16 sweeps table data types at $L{=}16$ across K under the identical encoder and protocol ($T{=}256$, 2^{22} samples, 3 seeds; seed std $< 10^{-3}$ bits): signed integers INT3–INT8, the mini-floats FP4 (E2M1, 15 values), FP6 (E2M3, 63 values) and E4M3 (253 values), the shipped type E2M5 (253 values), and the codepoints kept in BF16 or FP16 as references. Because the codebook is one fixed object independent of K, each type gets a single storage scale, chosen deterministically from the codebook's maximum entry: the largest 3-mantissa-bit float such that after rounding no codepoint saturates, which preserves the codebook's norm and the mirror symmetry of its outermost codepoints. For E2M5 the rule gives

a scale of 2: the outermost codepoint of the shipped table sits at 3.88σ (the conditional mean of an edge cell that holds only a few states; the ideal triangular quantile of Section 3.5 is 3.67σ), and 3.88×2 fills the type's span of 7.75 to within half a step.

Three observations. First, *the rule of thumb is $b \approx K + 2$ integer bits*: INT4 saturates beyond K=2, INT5 beyond K=3, INT6 beyond K=4, each failing as the index-law analysis predicts (Appendix B.1): with too few levels the grid cannot both cover the tails and resolve the center. Second, *level count sets where a type saturates and level placement sets the plateau height*: FP4 and INT4 (15 levels each) both cap beyond K=2, but FP4's exponent grid plateaus at 3.12 against INT4's 2.70 at the no-clipping scale; FP6 and INT6 both carry $K \leq 4$, with FP6 ahead by 0.07 at K=5 and 0.34 at K=6 (5.11 vs. 4.77); E2M5, whose 253 values keep all 127 composed codepoints distinct, stays within 0.06 bits of FP16 everywhere, and at equal level count E4M3's coarse mantissa caps it near FP6 (5.19 at K=6) while E2M5 reaches 5.76, so the codepoints need center resolution, not range. Third, the scalar columns (each type used as a plain scalar quantizer of $\mathcal{N}(0, 1)$) locate every plateau: wherever a type saturates, the CPTC rows flatten at the type's own scalar quality (1.60, 2.70, 3.70, ...), i.e. the coder degenerates to the format it stores once the table can no longer supply distinct codepoints. As scalar formats, E2M5 delivers $\sim$7.0 effective bits, BF16 $\sim$9.2 and FP16 $\sim$12.2, all well above the $K \leq 6$ rates the table serves, which is why rounding the 127 codepoints into any of them is nearly lossless. The design guidance: E2M5 is the right default for the full $K \in [1, 6]$ range; INT6 or FP6 are valid cost reductions for $K \leq 4$, where they give up at most 0.01 bits. (This table's E2M5 row uses the deterministic no-clipping scale; Table 3 uses the per-β calibrated scale, which at β=2 coincides to the displayed precision.)

A minimal codebook. The sweep above holds the table-size parameter at Q=6 and varies only the storage type; the two knobs can be driven down together. Setting $Q = K+1$ composes a codebook of $2^{K+2}-1$ codepoints, and INT(K+2) storage offers exactly $2^{K+2}-1$ symmetric levels, one per codepoint, so both the entry count and the entry width are at their matched minimum. Two bits further down on either knob is a provable floor, formalizing the table-sizing argument of Section 3.3:

Proposition 4 (Alphabet floor)**.** *If the decoded-state codebook takes at most M distinct values, then for every trellis depth L, block size T, and encoder, the per-sample MSE is at least that of the M-point Lloyd-Max scalar quantizer.*

The proof is two lines: every decoded value lies in the codebook, so each sample's squared error is at least its squared distance to the nearest codepoint, and the M-point Lloyd-Max codebook minimizes the expectation of exactly that quantity. At $Q = K-1$ the composed codebook has 2^K-1 distinct codepoints, and a symmetric INT(K) table has 2^K-1 representable levels whatever Q is, in either case one value short of the 2^K a rate-K scalar quantizer uses, so by the proposition the decoder is strictly worse than the scalar optimum no matter how deep the trellis. Any trellis gain therefore requires $Q \geq K$ and strictly more than K storage bits, and the matched configuration sits two doublings above each floor; Table 15 shows those two bits buy back essentially everything. By the dominance rule of Section 3.1, every row with $K \leq 5$ runs bit-exactly on the fixed ($L_{\max}$=16, Q=6) datapath via the shift $s = 5 - K$; the K=6 row has Q=7 and needs the wider table of CPTC$_{Q7}$.

Table 15: The smallest matched codebook, $Q = K+1$ stored in INT(K+2), and its effective bits (↑; protocol of Table 2): "Scale" is the no-clipping storage scale (Appendix C.3), "Table bits" the stored half (2^{K+1} entries of K+2 bits), and the last two columns the same-size and the Q=6 codebooks in FP16.

					Table	K_{eff}		
K	Q	Codepoints	Type	Scale	bits	INT(K+2)	same Q, FP16	Q=6, FP16
1	2	7	INT3	1.75	12	0.95	0.94	0.94
2	3	15	INT4	2.75	32	1.93	1.94	1.95
3	4	31	INT5	5	80	2.90	2.93	2.94
4	5	63	INT6	9	192	3.90	3.92	3.93
5	6	127	INT7	16	448	4.88	4.90	4.90
6	7	255	INT8	30	1024	5.86	5.89	5.82

Table 16: Effective bits (↑) by table data type at L=16 (T=256, 2^{22} samples, 3 seeds). Each unit-variance codepoint v is stored as $\mathrm{round}(c\,v)$ in the type's grid, with the fixed scale c of the Scale column, and decoded as the stored value$/c$. FP4 is E2M1 (15 values), FP6 is E2M3 (63); E4M3 and E2M5 have 253 values each (all-ones reserved for NaN). The last two columns use the type's grid as a plain scalar quantizer, at the same scale and at the MSE-optimal scale with clipping; the 16-bit rows are native round-to-nearest.

		CPTC (L=16)						Scalar (type alone)	
Table type	Scale	K=1	K=2	K=3	K=4	K=5	K=6	at c	opt. c
INT3	0.875	0.92	1.60	1.60	1.60	1.60	1.60	1.60	2.21
INT4	1.875	0.93	1.95	2.68	2.70	2.70	2.70	2.70	3.14
FP4	1.75	0.93	1.94	2.84	3.12	3.12	3.12	3.12	3.15
INT5	3.75	0.94	1.95	2.94	3.68	3.70	3.70	3.70	4.04
INT6	8	0.94	1.95	2.94	3.92	4.73	4.77	4.79	4.93
FP6	1.875	0.94	1.95	2.94	3.92	4.80	5.11	5.11	5.12
E4M3	112	0.94	1.95	2.94	3.92	4.80	5.19	5.23	5.24
INT7	16	0.94	1.95	2.94	3.93	4.88	5.64	5.77	5.83
INT8	32	0.94	1.95	2.94	3.93	4.89	5.74	6.73	6.73
E2M5	2	0.94	1.95	2.94	3.93	4.89	5.76	6.99	7.08
BF16	1	0.94	1.95	2.94	3.93	4.90	5.82	9.23	9.23
FP16	1	0.94	1.95	2.94	3.93	4.90	5.82	12.23	12.23

C.4 LLM WEIGHT-ONLY QUANTIZATION DETAILS

Every LLM result in this section uses the YAQA Sketch-B pipeline of Tseng et al. (2026) without fine-tuning: all quantized rows share its RHT preprocessing, two-sided Kronecker-factored Hessians, per-block scales, and the pipeline's published loading factor (`scale_override` 0.9, from the authors' released recipe), with only the trellis codebook swapped, so the comparison isolates the format from the rounding algorithm. The LLaMA 3.1 Instruct results use the authors' released Hessians; the Qwen3.8-27B results at the end of this section use Hessians we collected ourselves, since none are published for that model. All of them use the authors' evaluation protocol (OPTQ Wikitext-2 and C4 splits, sequence length 8,192). D_{KL} is the divergence to the unquantized model's next-token distribution, the metric YAQA optimizes and the most sensitive of the three.

KL tracks effective bits (Figure 1b). The i.i.d. effective bits of Table 2 predict the quantized model's quality. For CPTC, QTIP 1MAD and QTIP HYB at $K \in \{2, 3, 4\}$, nine cells per model (the CPTC$_{Q5}$ arm lies within 0.005 KL of CPTC throughout and is omitted for legibility), a quadratic least-squares fit of $\log_2 D_{\mathrm{KL}}$ against K_{eff} has $r^2 = 0.999$ at 8B and 0.997 at 70B. Because the codebook points cluster near 1.9, 2.9 and 3.9 bits, each point is weighted by the inverse count of its 0.1-bit K_{eff} bin, so the fit is uniform over the covered range rather than a curve through three clusters, and the 8B fit also includes the depth-sweep points below, 44 distinct cells in all. A slope of -2 is what a KL proportional to weight MSE predicts, since one effective bit is a $4\times$ factor in distortion (Equation (1)); 70B follows it at every rate (fitted slope -2.0 at 2 bits to -1.9 at 4), while at 8B the slope runs from -3.0 at 2 bits through -2.3 at 3 to -1.7 at 4: the KL gain per effective bit shrinks with rate. Rank order is exact at both scales: in every same-rate pair the codebook with more effective bits has the lower KL ($9/9$ pairs each; Spearman $\rho = -0.999$ and -1.000). Within one rate the local slope is shallower at K=4 (-1.4 and -1.3): HYB, whose loading factor the pipeline's authors tuned, does better in the model than its i.i.d. score predicts. At 8B the K=2 points are the steep end (local slope -3.8 across codebooks; -3.9 along depth): at 2 bits the weight error is about 7% of the weight variance, and the smaller model leaves the regime where KL is proportional to MSE while 70B does not yet. K_{eff} values are the L=16 Gaussian measurements; KL values are the seed-0 runs of Table 4.

Trellis-depth sweep. Varying L at fixed K moves a single codebook along the same law. We quantized LLaMA 3.1 8B with CPTC at every L from K to 16 for $K \in \{2, 3, 4\}$ (the per-(L, K) constants of Table 5, RHT seed 0, same recipe), and measured each configuration's i.i.d. effective bits under the protocol of Table 2. Within each rate, KL is monotone in L up to one near-tie at the top (K=4, L=15 vs. 16: 0.00954 vs. 0.00955, inside the seed spread): at K=4 it rises from 0.0095

(L=16, 3.93 effective bits) to 0.047 (L=4, 2.80); at K=3 from 0.038 to 0.169 (L=3, 2.16); at K=2 from 0.218 to 0.78 (L=2, 1.48). The L=K cells keep the standard table fill of Section 3.3 rather than a Lloyd-Max table, which is why K=4, L=4 sits below INT4; the exact scalar reproduction of Section 3.4 needs the scalar codebook loaded. Figure 1b plots the 38 cells with $K_{\text{eff}} \geq 1.75$ (all but $L \leq 5$ at K=2): $\log_2 D_{\text{KL}}$ against K_{eff} has Spearman $\rho = -0.999$ and a uniformly weighted slope of -2.4 ($r^2 = 0.99$); per rate the slope over L is -2.0 at K=4, -2.7 at K=3 and -3.9 at K=2, the steepening described above. (K=1 was also swept and destroys the model at every depth, KL 3.4–4.4 and perplexity above 190.) The depth points land on the codebook comparison of the previous paragraph without any change of fit, which is the transfer claim in its strongest form: what Table 2 measures on Gaussian samples is what the model sees. Scripts: `kl_vs_keff.py`, `keff_LK_grid.py`; run logs in `results/llm/`.

Five bits. As a check beyond Table 4, we ran the codebooks at K=5 (single runs, seed 0) and swept CPTC's depth (Table 17). The ordering holds: at 8B CPTC 0.0031, CPTC$_{Q5}$ 0.0032, 1MAD 0.0032 and HYB 0.0075; at 70B CPTC 0.0068, CPTC$_{Q5}$ 0.0069 and 1MAD 0.0070. HYB's cell is the alphabet ceiling of Theorem 1 showing up in the model: its 5-bit table has 3.94 effective bits, and its KL lands beside the K=4 cluster (0.0095–0.0131) rather than with the other 5-bit codebooks. These cells are kept off Figure 1b: from K=4 to 5 the 8B KL falls 3.1× over 0.97 effective bits, and the shallow K=5 depth cells sit 10–20% below the $K \leq 4$ trend at equal K_{eff}, so at 8B the map from effective bits to KL flattens with rate and depends mildly on the raw rate below $K_{\text{eff}} \approx 4.5$, while the 70B 5-bit cell stays on the -2.0 line.

Table 17: CPTC at K=5 on LLaMA 3.1 8B at every trellis depth (single runs): i.i.d. effective bits and KL to BF16 (↓).

L	5	6	7	8	9	10	11	12	13	14	15	16
K_{eff}	3.45	3.87	4.19	4.43	4.58	4.69	4.77	4.81	4.85	4.87	4.88	4.89
D_{KL}	0.0148	0.0078	0.0055	0.0042	0.0038	0.0035	0.0033	0.0033	0.0032	0.0031	0.0031	0.0031

Harness and protocol validation. Our reproduction of the YAQA pipeline matches its authors' published numbers wherever the same artifacts can be evaluated, at both scales. Our BF16 baselines match theirs to displayed precision (6.502/8.022 vs. 6.50/8.02 at 8B); evaluating their *released* checkpoints in our harness reproduces their published numbers on every metric (8B `QTIP-4Bit`: D_{KL} 0.0159 vs. 0.016, ppl 6.608/8.129 vs. 6.61/8.13; 70B YAQA-QTIP 4/3/2-bit: 0.0294/0.0910/0.2672 vs. 0.029/0.091/0.266); and running the QTIP quantizer ourselves under their recipe (seed 0 of the QTIP HYB rows of Table 4) reproduces the corresponding row of YAQA's Table 1 to all displayed digits at K=4 (0.0130, 6.608/8.118 vs. 0.013, 6.61/8.12), with K=3/2 regenerations at 0.0452/0.2403 vs. their 0.044/0.241 and the 70B regenerations at 0.0298/0.0967/0.3281 vs. their 0.030/0.094/0.335 at K=4/3/2. The released 70B checkpoints quoted in Section 4.2 are the authors' own YAQA-quantized models, produced with their full recipe including its fine-tuning stage. Two controls localize everything to the codebook: re-collecting the full 64K-sequence Sketch-B Hessians from scratch reproduces our results to three decimals, and six independent draws of HYB's k-means table span 0.006 effective bits. The loading factor matters: at unit scale the QTIP quantizer degrades to 0.0172 at K=4, so we adopt the authors' released 0.9; it was tuned by them for HYB, and CPTC is nearly insensitive to it (0.0097 at unit loading vs. 0.0096 at 0.9, seed 0, Table 19), so the shared setting does not favor CPTC. Every cell of Table 4 but 70B HYB is the mean of three runs that differ only in the Random Hadamard Transform's sign vectors: the pipeline is deterministic given the Hessians and this seed (re-collecting the Hessians moved results by under 0.001 KL), so we exposed the per-layer RHT seed and reran seeds 1 and 2 (seed 0 is the original run; Table 18). The spread across seeds is 0.3–4.1% of the value, comparable across codebooks, and CPTC has a lower KL than 1MAD on every individual seed at every rate and scale (the closest call is 70B K=3, seed 1: 0.0871 vs. 0.0872), so the ordering of Table 4 does not rest on a seed. HYB has three seeds at 8B and one at 70B because QTIP's encoder is about six times slower per run.

A full 8B quantization run takes about six minutes on one 8-GPU node (fused Viterbi encoder; a banded reformulation of the rounding solver with identical results), which is what makes recipe-uniform grids and these controls affordable. We did not run LC or RPTC in the LLM pipeline; their

Table 18: KL to BF16 (↓) on Wikitext-2 for each RHT seed behind the means of Table 4: seed 0 is the original run, seeds 1 and 2 change only the RHT sign vectors; range is $(\max - \min)/\text{mean}$. At every rate and scale both CPTC arms are below 1MAD on each seed; HYB was rerun at 8B only.

Model	Codebook	K	seed 0	seed 1	seed 2	mean	range
8B	QTIP HYB	4	0.01301	0.01294	0.01324	0.01306	2.3%
8B	QTIP 1MAD	4	0.01019	0.00978	0.01004	0.01000	4.1%
8B	CPTC$_{Q5}$	4	0.00975	0.00958	0.00951	0.00961	2.5%
8B	CPTC	4	0.00955	0.00947	0.00950	0.00951	0.8%
8B	QTIP HYB	3	0.04520	0.04509	0.04486	0.04505	0.8%
8B	QTIP 1MAD	3	0.04047	0.04092	0.04010	0.04050	2.0%
8B	CPTC$_{Q5}$	3	0.03953	0.03996	0.03920	0.03956	1.9%
8B	CPTC	3	0.03822	0.03912	0.03863	0.03866	2.3%
8B	QTIP HYB	2	0.24026	0.24487	0.24209	0.24241	1.9%
8B	QTIP 1MAD	2	0.23204	0.22568	0.22348	0.22707	3.8%
8B	CPTC$_{Q5}$	2	0.22073	0.22044	0.22252	0.22123	0.9%
8B	CPTC	2	0.21820	0.22043	0.22191	0.22018	1.7%
70B	QTIP 1MAD	4	0.02352	0.02404	0.02387	0.02381	2.2%
70B	CPTC$_{Q5}$	4	0.02264	0.02255	0.02224	0.02248	1.8%
70B	CPTC	4	0.02265	0.02272	0.02265	0.02267	0.3%
70B	QTIP 1MAD	3	0.08818	0.08719	0.08744	0.08760	1.1%
70B	CPTC$_{Q5}$	3	0.08449	0.08706	0.08518	0.08558	3.0%
70B	CPTC	3	0.08381	0.08496	0.08481	0.08453	1.4%
70B	QTIP 1MAD	2	0.32468	0.32101	0.32053	0.32207	1.3%
70B	CPTC$_{Q5}$	2	0.31241	0.31374	0.31727	0.31447	1.5%
70B	CPTC	2	0.31936	0.31884	0.31453	0.31758	1.5%

4–128 KiB tables are outside the deployment regime this section targets, so Table 2 remains the locus of that comparison.

Results (Table 4). Against HYB in the same pipeline, CPTC's D_{KL} margin at 8B grows with rate (9% at K=2, 14% at K=3, 27% at K=4; 8.6/12.1/26.9% against the published numbers), exactly as Table 2 predicts: HYB's fixed 512-entry alphabet saturates as K grows (Theorem 1). Against 1MAD, the table-free codebook, the margin is 5/4/3% at 8B and 5/4/1% at 70B for K=4/3/2 (full-table arm; 6/2/2% for CPTC$_{Q5}$), matching its i.i.d. ordering in Table 2; the large HYB margin is thus specifically the alphabet effect, and the codebook-level advantage over the best QTIP variant is the smaller, steady one. At K=5 quantization is nearly free (D_{KL} 0.0031, 6.516 vs. 6.502). HYB is the natural reference for the table-flexibility claim: like CPTC, its table can target any distribution, while 1MAD and 3INST are locked to their approximately Gaussian output (Section 2). The Q=5 arm, whose 63 FP16 codepoints make the GPU kernel's table gather bank-conflict-free (Appendix D.2), costs at most 0.001 D_{KL} at 8B (0.0001 at K=4), is within 0.002 at 70B K=3, and *beats* the full table at 70B $K \in \{2, 4\}$ (0.3145 vs. 0.3176, 0.0225 vs. 0.0227): the deployment table is not a quality compromise. Zero-shot accuracy over ARC-{c,e}, BoolQ, HellaSwag, PIQA sits within 1.1 points of BF16 (71.38 at 8B, 77.09 at 70B) for every codebook at $K \geq 3$ and drops together at K=2 (4.0–4.7 points at 8B, 2.4–2.8 at 70B), so it separates rates but not formats and we do not rank by it. Absolute 0-shot numbers do not transfer across harness versions (our harness scores the identical BF16 model 1.56 points higher on average than their published row), so every 0-shot number we report is self-measured.

Codebook or rounding? (Table 19). Replacing YAQA's adaptive rounding with round-to-nearest (RTN; both codebooks at unit loading) isolates the two effects. CPTC's advantage survives without adaptive rounding (37% lower D_{KL} than HYB under RTN, 44% under YAQA at the same loading), so it is a property of the codebook, not an interaction with the rounding algorithm; conversely YAQA rounding is worth 3.1–3.5× for either codebook.

A preliminary result on a different architecture: Qwen3.8-27B (Table 20). Everything above is dense-attention LLaMA. Qwen3.8-27B interleaves 48 gated-DeltaNet linear-attention layers with 16 gated-attention layers in a 3:1 pattern and carries a vision tower, so it tests whether the codebook advantage is architecture-specific. No Sketch-B Hessians are published for it, so we collected the full 65,536-sequence set ourselves (about 6.5 days on one 8-GPU node, text only with the vision tower

Table 19: The codebook effect with and without adaptive rounding: KL to BF16 (↓) at $K{=}4$ on 8B, same pipeline at unit loading, single runs, with YAQA's adaptive rounding and with plain round-to-nearest. The codebook advantage survives removing adaptive rounding.

Rounding	CPTC	HYB	CPTC advantage
RTN	0.03363	0.05334	-37%
YAQA-B	**0.00967**	0.01720	-44%
YAQA gain	$3.5\times$	$3.1\times$	

detached). We quantize 400 linears (six per DeltaNet layer, seven per attention layer); the small decay-gate projections, the short convolutions and all norms stay in BF16, and the 17,408-wide MLP dimension, which admits no power-of-two Hadamard, uses an order-68 Paley construction. This is a single run on our own Hessians, and we report it as preliminary. CPTC lowers D_{KL} at every rate ($0.0563/0.1060/0.2630$ vs. $0.0719/0.1175/0.3063$ at $K{=}4/3/2$) and C4 perplexity at $K{=}3$ and $K{=}2$, tying at $K{=}4$ (6.8171 vs. 6.8165); at $K{=}5$ (CPTC only; HYB's 512-pair table caps at 3.94 effective bits there, Table 2) C4 perplexity is within 0.005 of BF16 and D_{KL} halves again to 0.0295. Zero-shot accuracy behaves as at 8B and 70B: every quantized model sits within a point of BF16 (74.1), $K{=}2$ costs 0.7–1.0 points for both codebooks, and the two formats are not separated.

Why we do not rank this model by Wikitext-2 perplexity. On this split every quantized model scores *below* BF16, and the score falls as the bit rate falls, contradicting D_{KL} and C4, both of which degrade monotonically. The cause is miscalibration, not fidelity: softening the untouched BF16 model's logits by a temperature τ takes its Wikitext-2 perplexity from 7.685 at $\tau{=}1$ to a minimum of 7.440 at $\tau{=}1.15$, so temperature alone, with no quantization, reproduces CPTC's $K{=}4$ score of 7.447; on a 64-sequence C4 subset the same sweep moves perplexity only from 6.648 to 6.633, an improvement $16\times$ smaller, which is why the C4 ordering stays informative. CPTC at $K{=}4/3$ ($7.447/7.475$) sits at the temperature floor and $K{=}5$ (7.563) just above it, so at these rates quantization noise acts as a temperature and nothing more; at $K{=}2$ both codebooks (7.088 CPTC, 6.802 HYB) fall below anything a global temperature achieves, largest on the sequences where the model was most overconfident. The same harness shows no inversion on LLaMA (Table 4), so the effect belongs to this heavily post-trained model, overconfident on raw 2016 encyclopedia text while calibrated on C4, close to its training mixture. We report the column for completeness and rank by D_{KL}, which is also the argument of Tseng et al. (2026) for the metric. The caution generalizes: Wikitext-2 perplexity, still the default metric in most quantization papers, can reward damage, and a method reported on it alone would "beat BF16" at 2 bits here. In sum: in one validated pipeline CPTC has the lowest D_{KL} at every rate on LLaMA 3.1 8B and 70B (and beats HYB on Qwen3.8-27B in one preliminary run), the advantage is a property of the codebook rather than the rounding algorithm, and D_{KL} is the metric that orders codebooks reliably across models.

C.5 MATMUL-AWARE EXPERIMENT: A BIMODAL TARGET

This experiment demonstrates that CPTC's table absorbs an arbitrary target density (here the ρ-dependent, past-the-transition *bimodal* $\lambda^\star$ of Ang et al. (2026)) with no change to the decode path, coefficients, or encoder: only the 64 table bytes differ (Equation (4)). This is the table-design flexibility that RPTC pays 2^L entries for and QTIP's hash-based variants give up; it is a flexibility demonstration, not a deployment claim. We adopt the synthetic correlated-Gaussian setting of Ang et al. (2026) as-is (both operands quantized, unlike the weight-only rest of the paper; see that work for the significance of operand correlation): A and B are drawn so each output entry's multiplicative pairs are i.i.d. bivariate standard normal with correlation ρ, and we report relative Frobenius error of $\widehat{A}\widehat{B}$ over 10 trials at $m{=}n{=}128$, $k{=}256$.

Figure 5 shows CPTC-MMAWARE (CPTC with the $\lambda^\star$-composed table) achieves the lowest error at every ρ tested: its edge over the best standard scalar format (NF4) widens from $1.5\times$ at $\rho{=}0$ to $2.3\times$ at $\rho = 0.9$, where it is also $1.8\times$ better than Gaussian-table CPTC (trellis coding alone cannot capture the matmul-tail importance that $\lambda^\star$ encodes), and $1.2\times$ better than the scalar MATMUL-OPT of Ang et al. (2026), because codebook design and trellis coding compose: one is the table, the other the coder.

Table 20: Qwen3.8-27B (hybrid gated-DeltaNet/attention) in YAQA's pipeline with our own 65,536-sequence Hessians, no fine-tuning, single run: KL and perplexity (↓), Acc (↑, mean 0-shot accuracy as in Table 4). CPTC has the lower KL at every rate and the lower or equal C4 perplexity. †Wikitext-2 perplexity is not informative on this model (temperature alone reproduces the quantized scores from the BF16 model, see text), so that column is unbolded.

K	Codebook	D_{KL} ↓	Perplexity ↓		Acc ↑
		W2	W2†	C4	0-shot
16	BF16	0	7.685	6.800	74.1
5	CPTC	0.0295	7.563	6.805	73.9
4	HYB	0.0719	7.263	**6.817**	73.7
	CPTC	**0.0563**	7.447	**6.817**	74.0
3	HYB	0.1175	7.270	6.879	74.1
	CPTC	**0.1060**	7.475	**6.870**	73.8
2	HYB	0.3063	6.802	7.222	73.1
	CPTC	**0.2630**	7.088	**7.192**	73.4

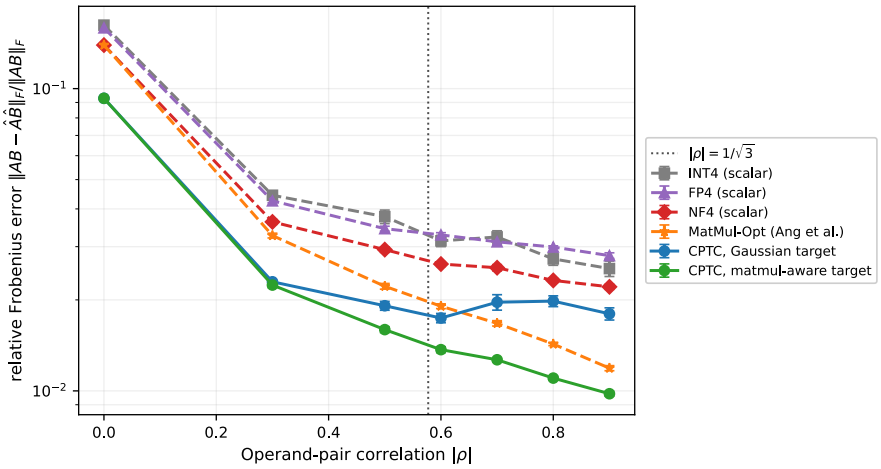


Figure 5: Matmul error against operand correlation: relative Frobenius error at 4 bits per value on synthetic correlated Gaussians. The vertical line is the predicted phase transition $|\rho| = 1/\sqrt{3}$; error bars are the standard error of the mean (SEM) over 10 trials.

Table design. The table contents encode the *reproduction distribution*, not the decode path. Ang et al. (2026) show that for a multiplicative pair with operand correlation ρ, the asymptotically optimal scalar quantizer has point density

$$\lambda^\star(u) \;\propto\; \exp(-u^2/6)\cdot\left((1-\rho^2)+\rho^2u^2\right)^{1/3}, \quad u = x/\sigma, \tag{3}$$

exhibiting a unimodal-to-bimodal phase transition at $|\rho| = 1/\sqrt{3}$. Their work and ours are orthogonal: they improve the *codebook* for a given K-level scalar quantizer; we improve the *coder* for a given codebook. The two compose cleanly: we build the table by the rule of Equation (2) with the target distribution taken to be the normalization of $\lambda^\star$,

$$\mathcal{T}[i] = \mathbb{E}_{U\sim\lambda^\star}\big[U \,\big|\, F^\star(U) \in [G(i{-}1),\, G(i)]\big], \tag{4}$$

where $F^\star$ is the CDF of the normalized $\lambda^\star$ and G is the empirical index CDF of Section 3.3. Decode primitive, (a_0, a_1), and Viterbi encoder are unchanged; only the 64 table bytes differ. The deployment recipe is to estimate the operand-pair ρ once per layer or attention head and look up the precomputed table.

Methods. Scalar baselines INT4, FP4, and NF4 use $\max|c|{=}1$ codebooks with a 20-point grid-searched per-matrix scale. MATMUL-OPT (Ang et al., 2026) uses a 16-level scalar codebook at $\lambda^\star$-quantile midpoints. CPTC-GAUSS is CPTC with the Gaussian-target table; CPTC-MMAWARE uses $\lambda^\star$ at the data-generating ρ (Equation (4)). For matmul-aware methods the codebook is in u-space (variance ≈ 3 at $\rho{=}0$, ≈ 5 at $\rho{=}0.9$); the shared scale calibration is described below.

Per-ρ results. Table 21 gives the numbers plotted in Figure 5. At $\rho = 0$, where $\lambda^\star$ degenerates to a Gaussian wider than the source, CPTC-MMAWARE ties CPTC-GAUSS after shared-scale calibration; the matmul-aware codebook pays off from $\rho = 0.5$ onward, and CPTC-GAUSS plateaus around 0.017–0.020 for $|\rho| \geq 0.5$.

Table 21: Relative Frobenius error (↓) of the quantized product against the exact one at $K{=}4$ ($m{=}n{=}128$, $k{=}256$, mean of 10 trials, SEM ≤ 0.002). Bold: best per column.

	Operand-pair correlation $\lvert\rho\rvert$						
Method	0.0	0.3	0.5	0.6	0.7	0.8	0.9
INT4	0.162	0.044	0.038	0.031	0.032	0.027	0.025
FP4	0.159	0.043	0.034	0.033	0.031	0.030	0.028
NF4	0.139	0.036	0.029	0.026	0.026	0.023	0.022
MATMUL-OPT (Ang et al., 2026)	0.139	0.033	0.022	0.019	0.017	0.014	0.012
CPTC-GAUSS	**0.093**	0.023	0.019	0.017	0.020	0.020	0.018
CPTC-MMAWARE	**0.093**	**0.022**	**0.016**	**0.014**	**0.013**	**0.011**	**0.010**

Calibration objective. Per-entry MSE is the wrong calibration objective for the matmul-aware codebook, which deliberately spreads codepoints into the source tails; it selects too tight a scale. By the symmetry of the data-generating model the optimal scales satisfy $c_A^\star = c_B^\star$, so we calibrate one shared scale per ρ by minimizing matmul Frobenius error over five held-out matrix pairs on a 19-point grid over $[0.4, 1.3]\cdot\hat{\sigma}$ (the calibrated values ship with the scripts).

Comparison with Ang et al. (2026). Ang et al. (2026) present a similar plot (their Figure 2) at higher rate ($K \approx 7$ bits) where the gap between matmul-aware and Gaussian-marginal codebooks is smaller. Our MATMUL-OPT curve reproduces their result at $K{=}4$: it tracks NF4 at $\rho{=}0$ and breaks away above the transition, with $1.85\times$ lower error than NF4 by $\rho{=}0.9$; CPTC-MMAWARE is strictly additive on top.

D SYSTEMS

D.1 VITERBI ENCODER: REFERENCE IMPLEMENTATION AND FUSED GPU KERNEL

Listing 1 is the single-iteration Viterbi update in PyTorch notation; the full T-step encoder applies this sequentially over the block's values, and tail-biting wraps that in an outer pass that fixes the

starting state. The rest of this appendix describes the fused GPU kernel that accelerates this reference by 24–113×. The decoder-side counterpart, a per-value golden reference decoder in plain Python, ships with the source release and is the oracle against which the dominance rule of Section 3.1 and the hardware kernels are validated bit-exactly.

Listing 1: Viterbi iteration for CPTC encoding.

```
# a:              running cost per L-K bit state prefix, shape (2^(L-K),)
# decoded_states: codebook, shape (2^L,); W_t: next value; K: bits/value
# The reference walks the block in reverse, so each new code lands in the
# low K bits of its window in the packed stream of Alg. 2.
def viterbi_iteration(a, decoded_states, W_t, K):
    # squared error of every full state s = (prefix << K) | u
    current_cost = (decoded_states - W_t).square()
    # state s extends prefix s >> K, whose running cost a holds
    total = a.unsqueeze(-1).expand(-1, 2**K).reshape(-1) + current_cost
    # next prefix = low L-K bits of s; minimize over the high K bits;
    # a_new and b_t have shape (2^(L-K),)
    a_new, b_t = total.reshape(2**K, -1).min(dim=0)
    return a_new, b_t
```

What the kernel computes. For each block the encoder solves the tail-biting minimum-distortion path problem of Section 3.2: over all state sequences that follow the bitshift transition and close into a cycle, it minimizes $\sum_t (w_t - \phi(q_t))^2$ with ϕ the decode map of Algorithm 1, and emits the per-value codes u_t. Launched as framework-level tensor operations (Listing 1 under `torch.compile`) the recursion is latency-bound: each of the T trellis steps and T backtrace steps is a separate kernel launch whose state round-trips through HBM. We fuse the whole pipeline (forward pass, $\arg\min$, half backtrace to the tail-biting midpoint state, second forward pass, full backtrace) into one persistent Triton launch, one program per SM slot walking a stripe of independent length-T blocks; the running costs live in registers across all T steps and the backpointer buffer is sized to the resident-program count, so it stays L2-resident between the forward and backtrace phases. Three optimizations matter. *Fused packed argmin:* the per-step minimum over 2^K predecessors packs the FP16 path cost and the predecessor index into one 32-bit word (cost in the high half), so one unsigned minimum returns both. *Backpointer compression:* for $K \leq 4$ the K-bit predecessor indices pack $8/K$ per byte, cutting backpointer write traffic by up to $8\times$ and turning the otherwise write-bound $K \leq 2$ configurations compute-bound; the first $T/2$ forward steps skip the backpointer store entirely, since the half backtrace never reads them. *Occupancy shaping:* at small 2^{L-K} the kernel packs several independent rows per program, with unroll factors and warp counts picked per (L, K) from device-structural quantities.

Table 22 reports the full (L, K) throughput matrix on an NVIDIA L40S at T=128: 24–113× faster than the `torch.compile` reference at matching MSE: the kernel accumulates path costs in FP16 where the reference uses FP32, which flips 2–4% of codes on near-tie paths and moves the MSE by at most 0.01% (both encoding the same FP16 input; $L \in [10, 16]$, $K \in [2, 6]$). At the main design point (L=16, K=4) it sustains ∼59 million values/s, putting a LLaMA 3.1 8B-scale encode at roughly two GPU-minutes; smaller trellises encode proportionally faster (0.92 billion values/s at L=12). The kernel is compute-limited: an NSight Compute profile shows 77% compute vs. 23% memory utilization, with the integer ALU pipe the bottleneck at 77%.

Against QTIP's shipped encoder. QTIP's released repository encodes with a batched PyTorch Viterbi (`bitshift_codebook.quantize`, commit `e90c668`), the same framework-level form as our reference. We timed it unmodified on the same L40S at its shipped settings (L=16, T=256, i.e. its 16×16 tiles), 2^{22} standard-normal values, median of 5 runs: 0.76 million values/s for 1MAD, 0.77 for 3INST and 1.5 for HYB, whose V=2 pairs halve the trellis steps. Our fused encoder at T=256 runs 48 million values/s at L=16, 940 at L=12 and 1,990 at L=10. Quality in the same run matches Table 2: CPTC at L=12 scores 3.868 effective bits against 1MAD's 3.864 at L=16 (CPTC L=16: 3.929). Encoding at equal quality is therefore about 1,200× faster than QTIP's shipped encoder (63× at equal depth), of which the 16× from L=12 versus L=16 is method-level and the rest is the fused kernel, which a fused Viterbi for QTIP's codebooks would recover.

Per-token KV-cache cost. The KV-cache use of Section 4.3 is naturally batched: every layer and head appends to its cache in step, so one generated token's new entries encode together in a single fused launch. Concretely, for LLaMA 3.1 8B (32 layers, 8 KV heads of dimension 128 under grouped-query attention), one token adds $2\times8\times128\times32 = 64$Ki cache values, i.e. 512 independent T=128 blocks. Measured directly (CUDA-event median of 50 launches on the L40S, tail-biting's two passes included), this single launch takes 134 μs at L=10 and 171 μs at L=12: 5× and 2.4× below the large-batch rates of Table 22, since 512 blocks do not fill the device, but still only a few percent of a single-GPU 8B-model decode step (several milliseconds, memory-bound), and further amortized whenever requests are batched or cache writes are accumulated over a few tokens. An isolated block, by contrast, is launch-latency-bound, which is why the batched formulation matters.

Table 22: Fused Viterbi encoder throughput (↑, millions of values per second) on one L40S at every trellis depth and bit width (T=128, 2^{22} values, median of 5), with the speedup over the `torch.compile` reference of Listing 1 in parentheses. Work grows as 2^L; the K dependence comes from backpointer traffic and occupancy.

$L\backslash K$	1	2	3	4	5	6
10	2424 (93×)	3062 (52×)	2658 (28×)	2353 (24×)	3425 (36×)	3350 (35×)
11	1374 (106×)	1613 (54×)	1715 (36×)	1633 (36×)	1905 (40×)	1820 (39×)
12	735 (113×)	842 (56×)	866 (36×)	920 (39×)	913 (37×)	906 (37×)
13	356 (109×)	451 (60×)	473 (40×)	499 (42×)	539 (44×)	519 (43×)
14	174 (103×)	206 (54×)	220 (36×)	245 (43×)	260 (45×)	174 (29×)
15	73 (86×)	93 (49×)	107 (36×)	121 (41×)	127 (43×)	86 (28×)
16	22 (52×)	42 (44×)	50 (34×)	59 (40×)	61 (42×)	45 (31×)

D.2 Fused decode + tensor-core GEMM on stock GPUs

This appendix details the decode-side kernel of Section 4.3. It is a separate claim from the accelerator result of Appendix D.4: that one is a post-layout datapath, this one is what a GPU with no CPTC support can do today with general-purpose integer instructions. The two share only the decode primitive of Algorithm 1.

Structure. The kernel targets weights times low-batch activations ($\leq$ 8 columns), the regime weight-only quantization is for. One CTA owns a 16×64 output tile; each lane reads a 128-bit window of the code stream and decodes 32 values, and *the decoded values land directly in the `mma.sync.m16n8k16` operand registers*. This costs nothing at run time because the map from stream position to fragment slot is a fixed permutation, applied to the weights offline before quantization, so no lane shuffles or shared-memory transposes are needed to reach tensor-core layout. Only one word per lane crosses a lane boundary (the window straddling the next lane's data), supplied by a single `shfl`. Activations are staged once per CTA in global $[k][n]$ order, where a k-row is exactly 8 halves = 16 B, and read back with `ldmatrix.trans`, whose lane distribution is verbatim the B-fragment map. The Q=5 table's 63 FP16 entries occupy 32 four-byte words, one per shared-memory bank, so the per-lane table gather is conflict-free without replication. Accumulation is FP32 throughout, including the cross-warp reduction and any block-level split-K atomics; only the final store narrows. The code stream is marked `.L2::evict_first`, since it has zero reuse and would otherwise evict data that does.

The INT4 baseline. Speedup over cuBLAS FP16 conflates two effects: moving 4× fewer weight bytes, and paying to decode them. To separate these we run an INT4 baseline that is the same kernel with the same layout, staging, MMA and epilogue, whose decode is a shift and a mask instead of Algorithm 1. It is verified to be the L=4 CPTC decoder with an identity table, so it is bit-exact-comparable, and it is an *upper bound* on any 4-bit weight-only kernel using this data path. The gap between CPTC and this baseline is the cost of the trellis and nothing else.

Results (Table 23). The rows are four decode schemes over one GEMM pipeline. *Two multiplies* is the full L=16 decoder of Algorithm 1 with the Q=5 table. *Packed state* packs two L=10 states 16 bits apart in one 32-bit register so that one multiply pair and one add serve two values. *Pair table* indexes a 2^{L+K}-entry table by the union window of two consecutive L=10 states, so one 32-bit

Table 23: Decode fused into the tensor-core GEMM at batch 1 (FP16 in/out, FP32 accumulation). "stack" (↓): time per token for the 32-layer LLaMA 3.1 8B linear stack; "vs. cuBLAS" (↑): its speedup over cuBLAS FP16; "saturated" (↓): a $229{,}376 \times 4{,}096$ matrix large enough that cuBLAS itself is bandwidth-bound; "% ceil" (↑): fraction of the measured pure-streaming read ceiling at that size (above 100 is possible because a GEMM's small activation operand comes from L2). The INT4 baseline shares the whole pipeline without any trellis decode, so its gap to the CPTC rows is the decode cost. The state-table scheme needs 128 KiB of shared memory per CTA and does not run on Ada. †QTIP's released HYB kernel at $K{=}4$ on the seven unfused per-projection shapes, timed as its own harness does, kernel time only. ‡CPTC on those same shapes, kernel-only / with the output-zeroing launch; speedup from the kernel-only time.

GPU	Decode	stack (ms)	vs. cuBLAS	saturated (μs)	% ceil
L40S	cuBLAS FP16	19.08	—	2595	95.9
(Ada)	two multiplies	5.24	3.64×	626	99.3
	packed state	5.20	3.67×	626	99.4
	pair table	5.39	3.54×	629	98.9
	INT4 baseline	5.12	3.73×	623	99.8
	QTIP HYB kernel†	5.8	3.3×	—	—
	two multiplies, unfused‡	5.35 / 5.84	3.57×	—	—
H100	cuBLAS FP16	5.51	—	657	101.1
(Hopper)	two multiplies	3.40	1.62×	330	50.4
	packed state	2.93	1.88×	269	61.7
	pair table	3.10	1.78×	282	58.8
	state table	2.67	2.06×	219	75.9
	INT4 baseline	2.11	2.61×	159	104.8
	QTIP HYB kernel†	3.2	1.7×	—	—
	two multiplies, unfused‡	3.54 / 4.31	1.56×	—	—
	state table, unfused‡	2.85	1.93×	—	—

load yields both values. *State table* indexes a 2^L-entry table by the $L{=}10$ state itself. The three $L{=}10$ schemes are the online design point of Section 4.3 (3.79 effective bits at $K{=}4$), not the $L{=}16$ quantizer of Table 2; the table isolates decode cost per weight, the only thing the schemes differ in. Two GPUs give opposite answers. On the L40S every scheme, including the baseline, sits at 98.9–99.8% of a measured pure-streaming ceiling at the saturated size: the memory system binds and the decode hides inside it, the two-multiply form 0.5% behind the baseline (2.4% across the LLaMA shapes). On the H100, with about $4\times$ the bandwidth, the decode ALU becomes the limiter and the same form falls 61% behind; the state-table scheme, which spends Hopper's larger shared memory instead of arithmetic (a 4 KiB state table replicated 32-fold into 128 KiB so the warp-wide gather is bank-conflict-free), reduces decode to a shift, a mask-or and one load and lands 27% behind. That table exceeds Ada's 100 KiB per-CTA cap, hence the empty L40S cell. All measurements are cold-L2 (the code stream is rotated over enough copies to exceed L2), CUDA-event timed, autotuned over CTA width, warps per tile, split-K and pipeline depth, and bit-exact against an FP32 reference matmul on the decoded weights.

What binds each scheme. Profiling attributes the two regimes to different pipes, and both are at method-level limits. The arithmetic schemes are ALU-issue-bound on Hopper: the SM90 ALU is 16 lanes wide, so a warp instruction occupies it for two cycles, and at 82.8% of peak ALU with `math_pipe_throttle` the dominant stall, the decode instruction count is the constraint; five variants spanning 4.5–5.5 ALU operations per value all measured within 0.5% of each other. The state-table scheme instead saturates the shared-memory path at 85.5% of peak L1TEX with one load per decoded value, so its constraint is shared-load issue rate; its lookups cannot be halved without a 2 MiB table or giving up the conflict-free replication, and an oracle that halves them for free bounds the available gain at 25.7%. The two constraints are complementary and neither is reachable by the other's method, which is the concrete sense in which Appendix D.4's decode unit is the remaining lever on bandwidth-rich parts, and the trend sharpens it: the lookups a GPU can afford per streamed byte (its conflict-free shared-memory rate over its HBM bandwidth) has fallen from 4.3 on V100 to 2.5 on H100 and about 1.1 on B200, while a saturating $K{=}4$ stream needs 2 per byte. A decode unit in the weight-load path is what keeps a computed codebook on the bandwidth curve.

Against QTIP's released kernels. QTIP ships CUDA decode kernels for its HYB codebook at $K \in \{2, 3, 4\}$, fixed to LLaMA's per-projection shapes at batch 1. We built them unmodified and timed them the way their own test harness does, inside a CUDA graph with per-launch events, cold-L2 by rotating weight copies past $2.5\times$ L2, and ran our kernel on the same seven shapes (q, k, v, o, gate, up, down; the fused shapes of Table 23 do not exist for their kernels). At $K{=}4$ their kernel takes 5.8 ms per token for the 32-layer LLaMA 3.1 8B stack on the L40S and 3.2 ms on the H100. On the same seven shapes our two-multiply form takes 5.35 and 3.54 ms kernel-only, or 5.84 and 4.31 ms including the output-zeroing launch its split-K atomics currently require (about $4\,\mu$s per launch, avoidable by zeroing in the preceding kernel's epilogue); the fused-stack rows of Table 23 include that launch but run fewer, larger shapes, hence their 5.24 and 3.40 ms. The Hopper state-table scheme takes 2.85 ms kernel-only on the unfused shapes. On the bandwidth-bound L40S the two decoders sit at the same wall; on the H100 they are within 12% either way, HYB ahead of the portable form (3.54 vs. 3.2) and behind the state table (2.85). HYB's is the cheaper GPU decode: its kernel holds the 9-bit pair table 32-fold replicated in shared memory (the conflict-free layout of our state-table scheme) and issues one 32-bit shared load per *two* values with no arithmetic, which is possible only because its output alphabet is 512 pairs, the restriction behind its quality gap in Tables 2 and 4. CPTC keeps the full alphabet and pays with the multiplier, which the accelerator of Appendix D.4 provides natively.

D.3 CPU DECODE KERNELS

The decode primitive also vectorizes on general-purpose CPUs. We implemented it in C++17 for AVX-512 (with the VBMI byte permutes), for AVX2 and for AArch64 NEON, with a portable scalar fallback, and verified every path bit-exact against the reference decoder for $K \in [1, 6]$, for the E2M5 and the INT6 table type and for both table configurations ($Q{=}6$ with the mirror fold, and the $Q{=}5$, 63-entry table stored in full), on x86 (Zen 3, Zen 4) and on Arm (Neoverse V3). The AVX-512 kernel decodes 64 values per step in about 15 instructions: one 64-byte load, one `vpermb` to place each group's chunk in a 64-bit lane, two `vpmultishiftqb` to cut out all 64 sixteen-bit windows, two `vpmullw` and two shifts per 32 windows (the modular products and the top-bit taps), and one `vpermi2w` (FP16 table) or `vpermb` (int8 table) for the lookup; the $Q{=}5$ form needs no compare, subtract or sign flip for the fold. Windows are extracted in a fixed permuted order within each 64-value tile, which the packer undoes offline as the GPU kernel does, so consumers see natural order. The 16-bit lane multiply is the only way to form $q \cdot a_j \bmod 2^{16}$ for 32 windows at once, and the table permute costs the same for 31, 63 or 127 entries, so a smaller table would not speed the CPU up. Table 24 gives single-core throughput; the kernel is issue-bound (removing one multiply in a timing-only experiment gains only 15%). The NEON kernel spends 16 instructions per 16 values on 128-bit vectors; on the Neoverse V3 its multiplies, dot product and table lookup share two of the four vector pipes, which caps it at 8.6 G values/s per core in the GEMM path (the standalone rates of Table 24 also pay a natural-order permute and a store per value) and puts $K{=}2$ and $K{=}3$ out of reach of the memory roof on that core.

Table 24: Single-core decode throughput on CPUs (↑, G values/s) from an L2-resident stream at $K{=}4$ (other K within 3% on x86 and 12% on NEON). FP16 output uses the E2M5 table, INT8 output the INT6 table. Zen 4: AMD EPYC (AVX-512); Zen 3: AMD EPYC (AVX2); Arm: Neoverse V3 (NEON). The last rows unpack INT4 nibbles through a 16-entry table, the cheapest 4-bit decode.

		FP16 out		int8 out	
Kernel	CPU	$Q{=}6$ fold	$Q{=}5$	$Q{=}6$ fold	$Q{=}5$
AVX-512	Zen 4	15.0	24.6	20.0	25.2
AVX2	Zen 4	2.8	3.1	4.1	4.7
AVX2	Zen 3	2.4	2.7	3.6	4.0
NEON	Neoverse V3	3.9	5.0	5.1	6.8
INT4 nibble unpack, AVX-512 / AVX2	Zen 4	47 / 29			
INT4 nibble unpack, NEON	Neoverse V3	23			

End to end. Table 25 times the weight stream of a full LLaMA 3.1 8B forward (all seven linear layers of each of 32 blocks, weights streamed from DRAM, activations resident) on the 64-core Zen 4 server in one parallel region, against FP16 weights, INT8 weights with VNNI dot products, and the INT4 nibble unpack feeding the same VNNI path. CPTC decodes to int8 codes and uses the same VNNI dot product (Q=5 table). The machine sustains about 270 GB/s, which FP16, INT8 and INT4 all reach: they are bandwidth-bound. The comparison of interest is at equal bits: CPTC at K=4 reaches the same bandwidth as INT4 and takes the same time at batch 1, so the trellis decode is hidden under the memory stream. At K=3 and K=2 the stream is 25% and 50% shorter and the time falls to 0.83× and 0.60× the K=4 time, against 0.75× and 0.50× for a purely bandwidth-bound format: the decode is partly exposed at K=3 (233 of the 270 GB/s roof) and more so at K=2 (217 GB/s). At batch 4 the decode is amortized for $K \geq 4$, which stay at or below their batch-1 time, while K=3 moves within noise and K=2 rises 15% as its decode is no longer hidden; at batch 8 the VNNI dot products themselves start to bind for all formats. The INT4 baseline is a pure nibble lookup without scale handling, the strongest form of it; run-to-run spread is about 5%. The lower block repeats the measurement on one 96-core NUMA node of an Arm Neoverse V3 server (NEON with SDOT dot products), whose node roof is about 405 GB/s. At K=4 the picture is the same: CPTC reaches 97% of the INT4 bandwidth and takes the same time at batch 1 (8.9 vs. 8.6 ms). At K=3 and K=2 the 128-bit NEON decoder, capped at 8.6 G values/s per core, cannot feed the shorter streams at the roof (11.1 and 16.7 G values/s per core would be needed), so those rates are decode-bound at 275 and 208 GB/s: K=2 finishes in the INT4 time and K=3 10% above it, rather than below; at batch 4 and 8 the same cap costs CPTC 18–44% over INT4 for $K \leq 4$ and about 85% at K=6. Kernels, tests and logs are in the released harness.

Table 25: Time to stream the weights of a 32-layer LLaMA 3.1 8B forward (ms, ↓) for B tokens sharing the weights, and the bandwidth reached at B=1, on a 64-core Zen 4 server (top) and one 96-core Neoverse V3 NUMA node (bottom). CPTC decodes the INT6 table (Q=5) into the INT8 dot product (VNNI on x86, SDOT on Arm); INT4 unpacks nibbles into the same dot product. Median of 7 runs.

Weights	bits	B=1	B=4	B=8	GB/s at B=1
FP16	16	51.9	51.7	67.4	269
INT8, VNNI	8	25.8	26.6	27.3	271
INT4 unpack, VNNI	4	13.3	14.1	17.3	262
CPTC K=6	6	19.9	19.7	24.1	263
CPTC K=4	4	13.6	12.9	18.2	257
CPTC K=3	3	11.3	11.8	17.9	233
CPTC K=2	2	8.1	9.3	15.9	217
Neoverse V3, one 96-core node (NEON, SDOT)					
FP16	16	36.0	38.1	52.8	388
INT8, SDOT	8	17.5	17.6	19.0	399
INT4 unpack, SDOT	4	8.6	8.6	12.8	405
CPTC K=6	6	13.8	16.2	23.7	379
CPTC K=4	4	8.9	10.7	15.2	391
CPTC K=3	3	9.5	12.4	16.9	275
CPTC K=2	2	8.4	10.6	15.1	208

D.4 Hardware implementation notes

This appendix documents the accelerator figures quoted in Section 4.3. CPTC has been implemented on a commercial systolic-array ML accelerator (name withheld for review), where the decode primitive sits inside the weight-load path that feeds the systolic array. The hardware additions on top of an existing INT4/FP4 dataflow are minimal: the two coefficient products in the shift-accumulate form of Equation (5) (§ *Cheap sequential decode* below), one 8-bit adder, and a 64-byte table held in registers (§ *Integration* below); the rest (state register, sign handling, output formatting) is shared with the integer and floating-point modes already present. We report the cost in node-independent units, gate equivalents (GE, the area of a two-input NAND in the same library), and as a fraction of the systolic array; absolute areas, clock frequency, and the process node are withheld. Sized for throughput parity with the native FP4 dataflow at $K \leq 4$ and with FP8 at $K \leq 6$, the decoder

costs about 2,600 GE per 8-bit value decoded per cycle: 1,300 GE for the two shift-accumulate coefficient products, 830 GE for the index sum and the table lookup (the table held in registers), and about 500 GE for window extraction, pipeline registers and glue. Across all lanes this is about 3% of the systolic array's post-layout area, the strictest denominator we can report: the fraction of the full chip, with its on-chip SRAM and interconnect, is far smaller. The design as a whole, CPTC decode path included, has closed timing and is ready for production, so the parity and cost figures describe a deployable datapath rather than a synthesis estimate (the per-value breakdown above is the one synthesis-level figure). The decoder RTL is formally verified equivalent to the golden reference decoder of Appendix D.1. The two multiplications and the index sum fit within one array clock cycle. The decoded operands are 8-bit (E2M5); on this accelerator the CPTC path runs at the same throughput as the native FP4 path for $K \leq 4$ and as the FP8 path for $K \leq 6$, not only at the same weight-load rate. Two qualifications on the figures: they come from the design's own post-layout and synthesis reports and cannot be independently verified under double-blind review, and we have not implemented a 1MAD or 3INST decoder in the same datapath, so we make no relative silicon-cost claim.

Integration. The packed bitstream is co-located with the layer's weights in the on-chip memory the systolic engine reads from. A dedicated load instruction broadcasts the 64-byte table into the decoder's table registers and sets (a_0, a_1, s) in per-layer configuration registers, a one-time per-layer cost; the hot path reads 32-bit words of packed code into an L-bit state register per lane. Each lane produces 4 or 8 decoded 8-bit values per cycle via Algorithm 1 (the register-held table is read by all lanes' lookup muxes in parallel), forwarded into the systolic array's multiply-accumulate pipeline in the same cycle an INT4 or FP4 value would be.

Throughput and sharing. The CPTC path sustains the chip's native per-cycle weight-load rate (parity with FP4 for $K \leq 4$, FP8 for $K \leq 6$), so a memory-bound decode step runs at the same tokens per second as the native FP4 path while delivering the quality of Table 4, and lower rates stream proportionally fewer weight bytes. The multipliers, adder and table are CPTC-specific; sign handling, casting and output formatting are shared with the existing INT8/FP8 dataflows.

Cheap sequential decode. The accelerator's decoder uses this form. For implementations that decode a block's values in stream order (consecutive states $q' = ((q \ll K) \mid q_i) \bmod 2^{16}$, as in Algorithm 2's windows traversed from the block's last value to its first), the bits shifted out of the window contribute multiples of 2^{16} to the product, so $x_j = (a_j \cdot q) \bmod 2^{16}$ updates incrementally:

$$x_j \leftarrow \big((x_j \ll K) + a_j \cdot q_i\big) \bmod 2^{16}. \tag{5}$$

This replaces each 16×16 multiplier with a K-bit shift and one addition of $a_j \cdot q_i$, a $16 \times K_{\max}$ product ($K_{\max} = 6$) whose partial-product array is about $3/8$ the size of the full one, or, since q_i takes only 2^K values, a 2^K-entry table of precomputed multiples (≤ 128 bytes per coefficient, 2^6 entries of 16 bits) and no multiplier at all.