HEPH: 3-BIT GRID-ALIGNED TRELLIS QUANTIZATION OF MXFP4
MIXTURE-OF-EXPERTS FOR ACCURATE, HADAMARD-FREE SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

A growing class of mixture-of-experts models — `gpt-oss`, Kimi K3, DeepSeek-V4 — ships its routed experts natively in the MXFP4 hardware microscaling format, produced by quantization-aware post-training rather than by a projection from high precision. HEPH is a trellis-coded quantization codec for those weights. Existing trellis quantizers (QTIP, QuIP#) attain excellent rate–distortion but decode to arbitrary `fp16` values in a randomized-Hadamard-rotated basis: at serving time they require a Hadamard transform on *every* forward pass, cannot use the native FP4 tensor-core kernel, and — lacking a fused low-bit grouped-expert kernel — must run a custom per-expert decode+MMA at batch rather than the single fused vendor kernel. HEPH instead makes the hardware grid the target: it reuses the checkpoint's own UE8M0 microscaling exponent as an *exact* per-block scale and constrains its trellis codebook to the 15-atom E2M1 alphabet, so every decoded weight is a valid MXFP4 element. Two things follow. First, because reusing the exponent makes the quantization targets land exactly on the grid and the search is activation-sensitivity-aware, HEPH attains the best accuracy in its rate class: on `gpt-oss-20b` it scores **80.03** averaged over five zero-shot benchmarks at 3.04 bits/weight, ahead of a rate-matched QTIP (77.72) and within 0.76 of the 4.25-bpw MXFP4 model it re-quantizes. Second, because the decoded tensor *is* on-grid MXFP4, serving needs *no* randomized Hadamard and no off-grid dequantization — a win in two complementary regimes. At *equal memory*, HEPH and a rate-matched QTIP both keep a $\sim$3-bpw stream resident and decode 3-bit weights in-kernel every forward; here HEPH serves **96.1** tokens/s at batch-1 decode on a Grace-Blackwell GPU inside SGLang versus **74.2** for QTIP ($1.30\times$), and 1.1–$1.5\times$ across batch 1–32 even against QTIP's real multi-batch trellis kernel. The reason is structural: HEPH's decode is a Hadamard-free on-grid gather, whereas QTIP runs a per-expert decode+MMA plus a Hadamard every forward. Alternatively, being valid MXFP4, HEPH can keep the same $\sim$3-bpw *storage* footprint yet expand to the full 4.25-bpw MXFP4 tensor *resident in memory*, trading the resident-memory saving — not the storage saving — for throughput on the *unmodified* production fused MXFP4 grouped-MoE kernel. That path runs all expert work in a single vendor kernel with no Hadamard, reaching **382** tokens/s at batch-1 versus **72.8** for QTIP ($5.25\times$) and 5–$29\times$ across batch 1–32. Our thesis: for mixture-of-experts serving on native-FP4 hardware, aligning the codec to the grid is a two-for-one — it is simultaneously the more accurate and the faster choice.

1 INTRODUCTION

Weight-only post-training quantization is usually posed as a projection from a high-precision tensor to a smaller numerical format. This premise is no longer universal. The routed experts of recent mixture-of-experts (MoE) models are increasingly shipped natively in a hardware microscaling format such as MXFP4 (Rouhani et al., 2023): a group of 32 weights shares a power-of-two UE8M0 scale and each weight is an E2M1 atom, and accelerators such as NVIDIA Blackwell execute this

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

format directly in their tensor cores. This is a convention rather than a one-off. `gpt-oss` (OpenAI et al., 2025) ships its experts this way, and both Kimi K3 (Team et al., 2026) and DeepSeek-V4 (DeepSeek-AI et al., 2026) carry MXFP4 quantization-aware training *through post-training* (K3 across SFT and RL; V4 during its post-training stage), so the routed experts they release are natively MXFP4 while their non-expert modules stay in higher precision. For such checkpoints the quantizer's input is not a high-precision tensor at all: the deployed tensor is already discrete, and its support is selected by the accelerator instruction set.

This regime opens a gap between the quantization literature and the serving contract. GPTQ and AWQ (Frantar et al., 2023; Lin et al., 2026), additive/vector quantizers including the lattice-based QuIP# (Egiazarian et al., 2024; Liu et al., 2024; Tseng et al., 2024), and the trellis method QTIP (Tseng et al., 2025) are formulated around continuous or rotated reconstruction values. In QTIP, the randomized Hadamard transform (RHT) is part of the inference computation, and the decoded weights are not required to lie in the native E2M1 $\times$ UE8M0 set: a production MXFP4 grouped-MoE kernel cannot consume such a tensor without conversion to legal operands or a different execution path. Conversely, simply dequantizing MXFP4 to `bf16` does not recover information — it only embeds the same discrete values in a larger container. Yet rate–distortion is not the metric that decides whether a quantized model is deployable: *serving throughput* is, and the best-compressing codec is not necessarily the fastest to serve.

We call the resulting problem *hardware-embedded quantization*: the codec must decode *inside* the checkpoint's own MXFP4 reproduction set — the finite family of tensors expressible as E2M1 atoms scaled by the stored per-32 UE8M0 exponents (defined formally in §3). We ask:

> Can a high-dimensional trellis keep its coding gain while decoding directly inside this native MXFP4 family, at a physical rate that remains tile-addressable?

HEPH answers this question affirmatively, making two changes to trellis-coded quantization (Marcellin & Fischer, 1990; Tseng et al., 2025). First, it *reuses the MXFP4 checkpoint's own UE8M0 exponent* as a free per-block scale; because that scale is a power of two, dividing it out is exact, so the values the trellis must encode land *exactly* on the 15-atom E2M1 alphabet with no rounding. Second, it *constrains the trellis codebook to that E2M1 grid*, so every codeword the decoder emits is a valid MXFP4 element. The re-quantized weight is therefore *on-grid* native MXFP4, served by the exact production fused MXFP4 kernel with no Hadamard.

Grid alignment turns out to help *both* axes at once. On the accuracy axis, re-quantizing `gpt-oss-20b`'s experts to 3.04 bpw, HEPH scores **80.03** averaged over five zero-shot benchmarks — the best in its rate class, ahead of a rate-matched QTIP (77.72) and QuIP# (75.25), and within 0.76 of the 4.25-bpw MXFP4 model it re-encodes. On the serving axis, HEPH *always* keeps the storage win — its on-disk footprint is a $\sim$3-bpw stream no matter how it is served — and this on-grid property pays off in two regimes. At *equal memory*, where HEPH and QTIP both keep that $\sim$3-bpw stream resident and decode 3-bit weights in-kernel every forward, HEPH serves at **96.1 tok/s** at batch-1 decode on a Grace-Blackwell (sm100) GPU inside SGLang versus QTIP's **74.2** ($1.30\times$), and stays 1.1–$1.5\times$ ahead across batch 1–32 — even against QTIP's real ported 3-bit multi-batch trellis kernel — because HEPH's decode is a Hadamard-free *on-grid* gather while QTIP runs a per-expert decode+MMA plus a randomized Hadamard every forward. When the resident-memory saving is not needed, HEPH can instead expand its stream to the full 4.25-bpw MXFP4 tensor and ride the *unmodified* vendor fused kernel for a further large throughput gain (§4) — an option an off-grid codec does not have. Our contributions:

- **A grid-constrained trellis codec.** We reuse the microscaling exponent as an exact scale and restrict the TCQ codebook to the E2M1 alphabet, so that (i) quantization targets land on the grid with no scale rounding and (ii) every decoded weight is valid MXFP4 (§3).

- **Best-in-class accuracy *and* serving at 3 bits.** We show HEPH attains the top five-benchmark accuracy among $\sim$3-bpw codecs on `gpt-oss-20b`, and — because it decodes to the grid — give the first, to our knowledge, *measured* whole-model serving comparison of a trellis codec against a native-FP4 path on Blackwell, where at equal memory (both decoding 3-bit weights in-kernel every forward) it serves 1.1–$1.5\times$ faster than a rate-matched QTIP (§4).

- **A decomposition of both wins.** We attribute the accuracy gain to exact exponent reuse and a functional Viterbi metric (§3), and decompose the serving advantage — a Hadamard-free on-grid decode and no unfused per-expert epilogue — with a controlled breakdown in §4 and §A.2.

We are careful about scope. HEPH needs a microscaling exponent to reuse: it re-quantizes MXFP4, not `fp16`. That is a real restriction but no longer a narrow one — `gpt-oss`, Kimi K3 and DeepSeek-V4 all ship their routed experts on this grid, and it is precisely the class that the vendor fused FP4 grouped-MoE kernels exist to serve. The construction depends only on the MXFP4 block format (a per-32 UE8M0 scale over E2M1 atoms), not on any property of the model, though we evaluate it on `gpt-oss-20b` alone. The thesis of this paper is that *for MoE serving on native-FP4 hardware, aligning the codec to the grid is a two-for-one: it is at once the more accurate and the faster choice.*

2 Related Work

Post-training weight quantization. Weight-only post-training quantization (PTQ) compresses a frozen model without retraining. GPTQ (Frantar et al., 2023) performs Hessian-aware error-feedback rounding; AWQ (Lin et al., 2026) scales salient channels by activation statistics. To push below 4 bits, vector and additive codebook methods such as AQLM (Egiazarian et al., 2024) and VPTQ (Liu et al., 2024) quantize groups of weights jointly. These methods store weights in bespoke codebook formats and, at inference, either dequantize to `fp16` or rely on custom kernels, leaving the vendor low-bit kernel behind — which we instead preserve.

Incoherence processing and trellis codes. QuIP (Chee et al., 2024) showed that quantization benefits from *incoherence*: multiplying weights and their Hessian by random orthogonal (Hadamard) rotations spreads magnitude and removes outliers. QuIP# (Tseng et al., 2024) combined a randomized Hadamard transform (RHT) with an E_8 lattice ("E8P") codebook, and QTIP (Tseng et al., 2025) replaced the lattice with *trellis-coded quantization* (TCQ) (Marcellin & Fischer, 1990): a stateful "bitshift" decoder attains ultra-high effective dimension at a fixed bitrate and is the current rate–distortion state of the art for LLM weights. Crucially, these methods rely on the RHT at *every* forward pass—activations must be Hadamard-transformed to match the rotated weight basis—and their decoded weights are arbitrary `fp16` values in the rotated basis, off any low-bit hardware grid. QuaRot (Ashkboos et al., 2024) similarly rotates to enable 4-bit inference. Our method keeps the trellis machinery of QTIP but removes both the Hadamard and the off-grid decode.

Hardware microscaling formats. The OCP Microscaling (MX) standard (Rouhani et al., 2023) defines MXFP4: E2M1 4-bit elements sharing a UE8M0 (power-of-two) scale per block of 32, executed natively by Blackwell tensor cores; NVFP4 (NVIDIA et al., 2026) uses 16-element blocks with an `fp8` scale. `gpt-oss` (OpenAI et al., 2025) ships its mixture-of-experts (MoE) weights *natively* in MXFP4, and production stacks such as SGLang (Zheng et al., 2024) serve it with a fused MXFP4 grouped-MoE kernel. Kimi K3 (Team et al., 2026) and DeepSeek-V4 (DeepSeek-AI et al., 2026) reach the same format from the other direction, carrying MXFP4 quantization-aware training through post-training (K3 across SFT and RL, with MXFP8 activations; V4 during its post-training stage) so that the routed experts they release are already MXFP4. All three put the routed experts on the grid and keep the non-expert modules — attention, routers, any shared or dense experts — in higher precision, which is the split we adopt. Prior quantizers treat the low-bit grid as a target to approximate. For these checkpoints there is no higher-precision release to approximate, so we instead treat the vendor MXFP4 grid as a *fixed alphabet* and re-encode it with a trellis at a lower rate, keeping serving on the production kernel.

3 Method: HEPH

HEPH re-quantizes an MXFP4 checkpoint with a trellis code whose *every* output is a legal MXFP4 element. We first establish why this on-grid property is unavailable to the rotation-based route that state-of-the-art trellis coders take (§3.1); we then build a codebook that lives on the grid (§3.2, the *MagStrat* codebook) and give the encoder and decoder (§3.3, *GAV*), whose Viterbi metric — unlike QTIP's — carries activation sensitivity directly.

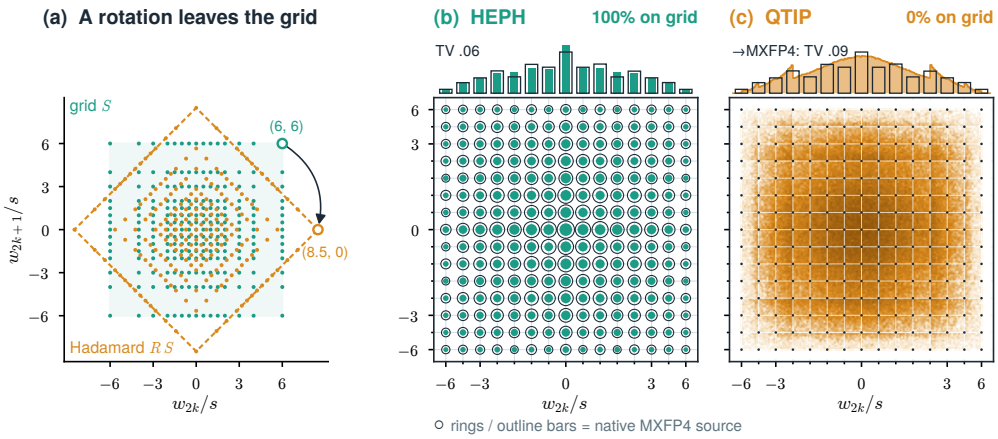


Figure 1: On-grid closure, on one real `gpt-oss-20b` expert (see §3.1). **(a)** A 2×2 Hadamard rotates the atom grid $\mathcal{S}(E)$ (teal) off itself (gold); e.g. the corner $(6, 6)$ maps to $(8.5, 0)$, outside the grid's own box. **(b, c)** Adjacent decoded weight pairs $(\hat{w}_{2k}, \hat{w}_{2k+1})/s_g$, axes in E2M1 code order: HEPH stays on the atom grid (100% on-grid, atom-marginal TV $= 0.06$ to the source; rings = native source), whereas QTIP is a continuous off-grid cloud (0%, TV $= 0.09$).

3.1 The hardware grid is closed only under signed permutations

An MXFP4 tensor stores each weight as $w_i = s_g\, a_i$, where $a_i \in \mathcal{A} = \{0, \pm 0.5, \pm 1, \pm 1.5, \pm 2, \pm 3, \pm 4, \pm 6\}$ is one of the 15 E2M1 atoms and $s_g = 2^{E_g - 127}$ is a power-of-two scale shared by a block of 32 weights through the stored UE8M0 byte E_g. `gpt-oss` ships its experts in exactly this format. For a fixed scale assignment E, the set of tensors the hardware can represent at those scales is the finite Cartesian product

$$\mathcal{S}(E) = \prod_g (s_g \mathcal{A})^{32}. \tag{1}$$

A codec is *on-grid* if its reconstruction lies in $\mathcal{S}(E)$ (or in the wider family obtained by also rewriting the UE8M0 bytes); only then can the production fused MXFP4 kernel consume it without conversion.

Proposition 1 (grid closure). For a non-degenerate scale assignment E, every orthogonal matrix R with $R\mathcal{S}(E) = \mathcal{S}(E)$ is a signed coordinate permutation that maps each coordinate to one of equal scale. In particular, a randomized Hadamard or generic random-orthogonal transform is *not* grid-preserving.

Proof sketch. The convex hull of $\mathcal{S}(E)$ is the axis-aligned box $\prod_i [-6s_i, 6s_i]$ (since $\max_{a \in \mathcal{A}} |a| = 6$); an orthogonal R fixing $\mathcal{S}(E)$ fixes that box, so it is one of its symmetries — a signed coordinate permutation sending each coordinate to one of equal scale, and $\mathcal{A}$'s sign-symmetry makes every such permutation grid-preserving. Full proof in §A.1.

The consequence is structural, not incidental. QTIP (Tseng et al., 2025) and QuIP# (Tseng et al., 2024) reconstruct in a randomized-Hadamard-rotated basis — their decoded weights lie in $R\mathcal{S}(\cdot)$ for a Hadamard R — so by Proposition 1 they fall off the native family, and a fused MXFP4 kernel cannot consume them without a conversion or a different execution path. HEPH takes the only other route the proposition leaves open: it never rotates, and reconstructs inside $\mathcal{S}(E)$ itself.

Figure 1 shows this on one real `gpt-oss-20b` expert — the very weights QTIP quantizes. Panel (a) is Proposition 1 as geometry: a 2×2 Hadamard sends the atom grid to a rotated diamond whose vertices fall *between* the grid points, so no rotation preserves it. Panels (b, c) plot adjacent decoded weight pairs on axes in E2M1 *code order*: HEPH places every weight exactly on an atom, reproducing the native source (100% on-grid), whereas QTIP — rescaled by the same per-32 UE8M0 power-of-two but left unsnapped — spreads into a continuous off-grid cloud (0%).

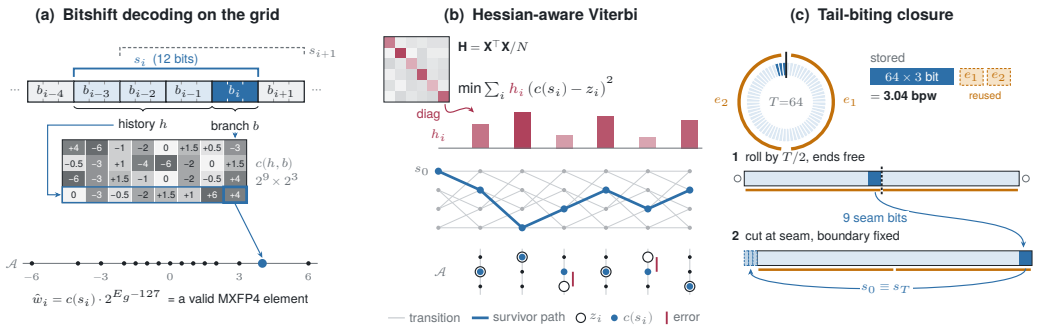


Figure 2: GAV: grid-aligned encoding and decoding (§3.3). **(a)** Bitshift decoding maps each state through the MagStrat table $c(s)$ onto the E2M1 atom grid. **(b)** A Hessian-aware Viterbi weights each branch by the sensitivity h_i. **(c)** Tail-biting closure in two passes; the stored stream is 64×3 label bits plus the reused UE8M0 exponents (3.04 bpw).

3.2 The MagStrat codebook

Two design choices place both the encoding targets and the decoder codewords on the grid of equation 1.

Targets: reuse the microscaling exponent. For a weight w_i in a block with byte E_g, we reuse the checkpoint's own scale $s_g = 2^{E_g - 127}$ and encode $z_i = w_i / s_g$. Because s_g is a power of two, this division is *exact* in floating point, so the values the trellis must encode are *exactly* the 15 E2M1 atoms, with no scale rounding. The scale is already stored in the checkpoint, so reusing it costs nothing. On a real expert matrix, reusing the source exponent keeps 100% of targets on the atom grid, versus 70.3% when a re-derived RMS scale is used (Figure 3).

Codewords: a magnitude-stratified atom table. We fill the 2^L decoder codewords from $\mathcal{A}$ alone. E2M1 has exactly eight magnitudes $\mathcal{M} = \{0, 0.5, 1, 1.5, 2, 3, 4, 6\}$, and K=3 payload bits give 2^K=8 branches per step, so each history can be assigned a full permutation of $\mathcal{M}$ with an independent random sign. Concretely, for a state s (the L-bit trellis state of §3.3) with history $h = \lfloor s/2^K \rfloor$ and branch $b = s \bmod 2^K$, the emitted atom is

$$c(s) = \varsigma_{h,b}\, \mu_{\pi_h(b)}, \qquad \mu \in \mathcal{M},\ \varsigma_{h,b} \in \{-1, +1\}, \tag{2}$$

where π_h is a per-history permutation of the eight magnitudes and $\varsigma_{h,b}$ a sign, both drawn from a fixed seed. We call this the *MagStrat* (magnitude-stratified) codebook. Every codeword $c(s)$ is an exact atom, and the whole table ($2^L \times$ `fp16`, e.g. 8 KB at L=12) is regenerated from its seed at load time, so it does not count toward the stored rate.

Consequence: on-grid MXFP4. Combining the two choices, every decoded weight is $\hat{w}_i = c(s_i)\, s_g$ with $c(s_i) \in \mathcal{A}$ — a valid MXFP4 element. The re-quantized tensor is therefore itself a valid MXFP4 tensor, served by the unmodified production fused MXFP4 kernel with no Hadamard. Because targets and codewords both live on the grid, ≈76% of weights reconstruct *bit-exactly* to their source MXFP4 value (Figure 3) and the residual is sparse; HEPH is a *lossy* 3.04-bpw re-quantization of the 4.25-bpw checkpoint whose small residual costs only 0.76 accuracy points (§4). The CPU reference and the fused CUDA decoder produce bit-identical reconstructions over the full packed stream (§4). Because the alphabet is fixed by the hardware rather than chosen by a rotation, HEPH keeps "the grid is already the right alphabet" as free structure that a randomized-Hadamard codec (QTIP/QuIP#) discards for generic Gaussianization; the Hessian-aware Viterbi (§3.3) then spends the trellis's coding gain where activations are most sensitive. This is the mechanism behind HEPH's accuracy lead in §4.

3.3 GAV: grid-aligned Viterbi encoding and decoding

Trellis. Following QTIP (Tseng et al., 2025; Marcellin & Fischer, 1990), the codewords are driven by a stateful "bitshift" decoder (Figure 2a). A sequence of K-bit labels $b_1, \ldots, b_T$ advances an L-bit

state

$$s_i = \big((s_{i-1} \ll K) \mid b_i\big) \bmod 2^L, \tag{3}$$

so each state has 2^K successors and shares $L-K$ bits with its predecessor; the state's memory makes the effective codebook dimension far larger than 2^K, which is what gives the trellis its coding gain. We use $K{=}3$ and $L{=}12$. Decoding label i is the single lookup $c(s_i)$ of equation 2; once labels are unpacked the decode is a pure gather with no sequential dependency. We set the blocklength to $T{=}64$ by concatenating two adjacent 32-weight groups of one output row, so shaping memory crosses a scale boundary while each half keeps its own UE8M0 exponent and the targets remain exactly the atoms; only the random-access granularity grows from 32 to 64.

A functional (Hessian-aware) branch metric. Labels are chosen by a Viterbi search (Figure 2b) that minimizes $\sum_i d_i$ with the *sensitivity-weighted* branch cost

$$d_i = h_i \big(c(s_i) - z_i\big)^2, \tag{4}$$

where h_i is the diagonal of the routed-calibration input second moment $\mathbf{H} = \mathbf{X}^\top\mathbf{X}/N$ (for the gate/up projections $\mathbf{X}$ is the expert input; for the down projection it is the SwiGLU intermediate). This is where GAV departs from QTIP. QTIP's Viterbi minimizes the *unweighted* error $\|c - z\|^2$: its proxy objective $\mathrm{tr}\big((\hat{W} - W)\mathbf{H}(\hat{W} - W)^\top\big)$ is not additive along the sequence, so it cannot be placed inside the Viterbi recursion, and QTIP recovers Hessian-awareness only through an *outer* BlockLDLQ error-feedback loop (Tseng et al., 2025). We observe that the *diagonal* of $\mathbf{H}$ is a per-coordinate reweighting, which keeps the metric additive; sensitivity can therefore enter the branch cost directly and the Viterbi stays exactly optimal, with no outer loop.

Tail-biting closure. The sequence is *tail-biting* — the terminal state wraps to the initial one — which removes the overhead of storing a boundary state (Figure 2c). We solve it with two Viterbi passes: a free-boundary pass on the sequence rolled by $T/2$ estimates the $L-K$ shared history bits at the seam, and a constrained pass then fixes that boundary. This is QTIP's approximate tail-biting solver, not a proof of the globally optimal cyclic path. The stored stream is the 3-bit label path plus the reused UE8M0 bytes — already present in the checkpoint — for the 3.04-bpw footprint evaluated in §4.

4 Experiments

4.1 Setup

We serve `gpt-oss-20b` (24 layers, 32 experts/layer, top-4 routing, hidden size 2880; three 2880×2880 projections per expert) on a single Grace-Blackwell GPU (sm100) inside SGLang (Zheng et al., 2024) with tensor parallelism 1. The native MXFP4 checkpoint loads to **14.66 GB** resident and is served by the fused FlashInfer MXFP4 grouped-MoE kernel. The QTIP baseline is integrated into the *same* stack at a rate-matched $K{=}3$ format (3.03 bpw), each method given its own best kernel per batch and verified against an independent reference before timing (§A). We report median decode throughput with CUDA graphs on for both.

4.2 Accuracy

Table 1 reports zero-shot accuracy on five benchmarks (ARC-Challenge, ARC-Easy, HellaSwag, PiQA, MMLU) under an identical protocol for all rows (0-shot next-token-choice log-likelihood in the Harmony `final` channel; item counts equal across methods). All 3-bit codecs re-quantize the same 768 routed experts; the source row is the unquantized 4.25-bpw MXFP4 model. HEPH attains the highest five-benchmark mean, **80.03**, at 3.04 bpw — $+2.31$ over a rate-matched QTIP (`3inst`, 77.72), ahead of QTIP `quantlut_sym` (77.30) and QuIP# (75.25), and within **0.76** of the MXFP4 ceiling (80.79). The gain traces to exact exponent reuse and the Hessian-aware Viterbi (§3.2).

Pushing the codec to $K{=}2$ ($\sim$2 bpw) sharpens the separation from off-grid trellis codecs. On the three benchmarks scored at this rate (Table 2; ARC-Easy and PiQA were not run at 2 bits, so its mean is not comparable to Table 1), HEPH keeps the top mean — $+4.9$ over QTIP `3inst` and $+4.0$ over `quantlut_sym` — while QuIP# collapses toward chance. This is an *ordering* statement

Table 1: Zero-shot accuracy (%) on `gpt-oss-20b` at ∼3 bpw, same protocol for all rows. HEPH has the highest five-benchmark mean, within 0.76 of the 4.25-bpw MXFP4 source it re-quantizes.

method	**bpw**	ARC-C	ARC-E	HellaSw.	PiQA	MMLU	**mean**
source (MXFP4, ceiling)	4.25	88.05	96.00	68.29	82.64	68.96	80.79
HEPH (ours)	3.04	88.23	95.66	66.93	81.99	67.33	**80.03**
QTIP `3inst`	3.02	87.54	95.41	60.96	77.75	66.93	77.72
QTIP `quantlut_sym`	3.03	86.01	95.03	60.64	79.00	65.82	77.30
QuIP# E8P12RVQ3B	3.01	83.11	93.86	62.11	73.78	63.37	75.25

Table 2: Zero-shot accuracy (%) on `gpt-oss-20b` at ∼2 bpw (K=2), same protocol as Table 1. ARC-Easy and PiQA were not scored at 2 bits, so this is a three-benchmark comparison and its mean is *not* comparable to the five-benchmark means of Table 1.

method	**bpw**	ARC-C	HellaSw.	MMLU	**mean (3-bench)**
HEPH (ours)	2.04	**80.80**	31.56	**59.63**	**57.33**
QTIP `3inst`	2.02	71.16	31.94	54.09	52.40
QTIP `quantlut_sym`	2.03	72.27	**35.70**	52.03	53.33
QuIP# E8P12	2.01	34.04	25.58	31.86	30.50

carried by ARC-C and MMLU; on HellaSwag every trellis codec, HEPH included, degrades sharply and HEPH does not lead.

4.3 Inference speed

HEPH keeps the same ∼3-bpw *storage* footprint however it is served; because its decoded tensor is valid on-grid MXFP4, that stream can be served two ways, which trade the resident-memory saving against throughput.

Equal memory. HEPH decodes its 3.04-bpw stream *in-kernel* on every forward (a pure gather, no Hadamard; §3); QTIP decodes its own 3.03-bpw trellis. At equal memory (Table 3) HEPH serves **96.1 tok/s** at batch-1 decode versus QTIP's 74.2, and stays 1.1–1.5× ahead across batch 1–32 even against QTIP's real multi-batch trellis kernel.

Storage only. When the resident-memory saving is not needed, HEPH expands its stream to the full 4.25-bpw MXFP4 tensor in memory — the on-disk footprint stays ∼3 bpw — and serves it on the *unmodified* vendor fused MXFP4 grouped-MoE kernel: one kernel, still no Hadamard. This reaches **382 tok/s** at batch-1 decode versus 72.8 for QTIP (5.25×), widening to 27–29× at batch 8/32 (Table 4); an off-grid codec cannot expand to a native MXFP4 tensor at all.

HEPH wins the equal-memory comparison even though its in-kernel decode *operator* is 2.2× slower than QTIP's in isolation: it carries no per-forward randomized Hadamard and no unfused per-expert gather/SwiGLU/accumulate, and once expanded it rides a single fused kernel. We decompose the batch-1 cost, the Hadamard's small share, the batch-scaling, and the isolated-operator microbenchmark in §A.2.

5 Conclusion and limitations

We presented HEPH, a trellis-coded weight codec that reuses a microscaling checkpoint's own UE8M0 exponent as an exact per-block scale and constrains its codebook to the E2M1 alphabet, so that every decoded weight is on-grid native MXFP4. This single design choice pays off on two axes at once. On accuracy, exact exponent reuse and a functional Viterbi give the best five-benchmark score in the 3-bit class on `gpt-oss-20b` (80.03, within 0.76 of the 4.25-bpw MXFP4 source

Table 3: Equal-memory decode throughput for `gpt-oss-20b` on one Grace-Blackwell GPU (SGLang, TP1, CUDA graphs on), median decode tok/s. Both methods keep a ∼3-bpw stream resident and decode 3-bit weights every forward with their own real kernels (HEPH's F4 in-kernel decode; QTIP's `decompress_matvec` at batch 1 and its multi-batch tile-decode+MMA kernel, ported to sm100 and 3-bit, at batch 8/32). QTIP here is the same-host re-run.

batch	HEPH (3.04 bpw, tok/s)	QTIP (3.03 bpw, tok/s)	speedup
1	**96.1**	74.2	1.30×
8	**111.0**	97.2	1.14×
32	**448.6**	291.3	1.54×

Table 4: Storage-only decode throughput: HEPH's 3.04-bpw stream is expanded to a full 4.25-bpw MXFP4 tensor resident in memory and served by the *unmodified* vendor fused MXFP4 grouped-MoE kernel (no Hadamard), vs. QTIP's real 3-bit multi-batch trellis kernel. One Grace-Blackwell GPU, SGLang TP1, CUDA graphs on; median decode tok/s. QTIP here is the original-host measurement of the same kernel as Table 3 (hence 72.8 vs. 74.2).

batch	HEPH fused-MXFP4 (tok/s)	QTIP (tok/s)	speedup
1	**382**	72.8	5.25×
8	**2649**	96.4	27.5×
32	**8392**	290	28.9×

and +2.31 over a rate-matched QTIP). On serving, at *equal memory* — both methods keeping a ∼3-bpw stream resident and decoding 3-bit weights in-kernel every forward — HEPH serves 96.1 tok/s at batch-1 decode versus 74.2 for a rate-matched QTIP integration, and 1.1–1.5× across batch 1–32 even against QTIP's real multi-batch trellis kernel, because HEPH's decode is a Hadamard-free on-grid gather while QTIP runs a per-expert decode+MMA plus a Hadamard every forward; and because the tensor is valid MXFP4, HEPH can additionally ride the unmodified production fused kernel at the 4.25-bpw footprint. The broader lesson is that rate–distortion, the field's dominant quantization metric, does not by itself predict serving behavior on modern low-bit hardware — and that aligning a codec to the vendor grid can be simultaneously the more accurate and the faster choice. We hope grid-aligned coding becomes a first-class objective for quantization methods targeting native-FP4 accelerators.

Limitations. HEPH applies only to checkpoints shipped on a hardware microscaling grid: an `fp16` model has no UE8M0 exponent to reuse, and within that class we evaluate a single model, `gpt-oss-20b`. Its in-kernel decode operator is $\approx 2.2\times$ slower than QTIP's in isolation (§A.2); the equal-memory serving advantage comes from carrying no per-forward Hadamard and no unfused per-expert epilogue, not a faster kernel, and closing that operator gap with a tensor-core FP4 decode pipeline is future work.

Reproducibility statement

All serving numbers are measured end-to-end inside SGLang on a single Grace-Blackwell (sm100) GPU with tensor parallelism 1 and CUDA graphs enabled, using the public `gpt-oss-20b` MXFP4 checkpoint. Section 3 specifies the codec (exponent reuse, the E2M1-constrained trellis codebook, the K=3 tail-biting structure with state transition $s_i = ((s_{i-1} \ll 3) \,|\, b_i) \bmod 2^{12}$, blocklength T=64, and the Hessian-aware Viterbi metric) in full. The on-grid property is that every decoded value is a valid MXFP4 element; separately, the fused CUDA decode reproduces the evaluated reconstruction word-for-word (0 of 8.29 M `fp16` words differ on a real expert matrix), so the weights we time are exactly those we score. For the QTIP baseline we state the exact re-quantization configuration (decode mode `quantlut_sym`, L=16, tlut_bits=9, V=2, K=3, mma-swizzled trellis) and verify the integration against an independent decode reference to relative error $\leq 2.7 \times 10^{-3}$

before timing it. Appendix A lists the kernel build, the per-batch benchmark commands, and the correctness checks.

THE USE OF LARGE LANGUAGE MODELS

We used a large language model as a coding and writing assistant: to help implement the CUDA kernel build and the SGLang integration used for the serving measurements, to draft and edit the prose of this paper, and to help format tables and figures. All experimental results — accuracy numbers, throughput measurements, and correctness checks — were produced by executing code on the hardware described in Section 4, not generated by the model, and were verified by the authors against the logged outputs. The authors take full responsibility for the content, claims, and artifacts of this work.

REFERENCES

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms, 2024. URL https://arxiv.org/abs/2404.00456.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher De Sa. Quip: 2-bit quantization of large language models with guarantees, 2024. URL https://arxiv.org/abs/2307.13304.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao

Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization, 2024. URL https://arxiv.org/abs/2401.06118.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

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, 2026. URL https://arxiv.org/abs/2306.00978.

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, 2024. URL https://arxiv.org/abs/2409.17066.

M.W. Marcellin and T.R. Fischer. Trellis coded quantization of memoryless and gauss-markov sources. *IEEE Transactions on Communications*, 38(1):82–93, 1990. doi: 10.1109/26.46532.

NVIDIA, Felix Abecassis, Anjulie Agrusa, Dong Ahn, Jonah Alben, Stefania Alborghetti, Michael Andersch, Sivakumar Arayandi, Alexis Bjorlin, Aaron Blakeman, Evan Briones, Ian Buck, Bryan Catanzaro, Muya Chang, Jinhang Choi, Mike Chrzanowski, Eric Chung, Victor Cui, Steve Dai, Bita Darvish Rouhani, Carlo del Mundo, Deena Donia, Burc Eryilmaz, Henry Estela, Abhinav Goel, Oleg Goncharov, Yugi Guvvala, Robert Hesse, Russell Hewett, Herbert Hum, Ujval Kapasi, Brucek Khailany, Mikail Khona, Nick Knight, Alex Kondratenko, Ronny Krashinsky, Ben Lanir, Simon Layton, Michael Lightstone, Daniel Lo, Paulius Micikevicius, Asit Mishra, Tim Moon, Deepak Narayanan, Chao Ni, Abhijit Paithankar, Satish Pasumarthi, Ankit Patel, Mostofa Patwary, Ashwin Poojary, Gargi Prasad, Sweta Priyadarshi, Yigong Qin, Xiaowei Ren, Oleg Rybakov, Charbel Sakr, Sanjeev Satheesh, Stas Sergienko, Pasha Shamis, Kirthi Shankar, Nishant Sharma, Mohammad Shoeybi, Michael Siu, Misha Smelyanskiy, Darko Stosic, Dusan Stosic, Bor-Yiing Su, Frank Sun, Nima Tajbakhsh, Shelby Thomas, Przemek Tredak, Evgeny Tsykunov, Gandhi Vaithilingam, Aditya Vavre, Rangharajan Venkatesan, Roger Waleffe, Qiyu Wan, Hexin Wang, Mengdi Wang, Lizzie Wei, Hao Wu, Evan Wu, Keith Wyss, Ning Xu, Jinze Xue, Charlene Yang, Yujia Zhai, Ruoxi Zhang, Jingyang Zhu, and Zhongbo Zhu. Pretraining large language models with nvfp4, 2026. URL https://arxiv.org/abs/2509.25149.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, Kai Chen, Mark Chen, Enoch Cheung, Aidan Clark, Dan Cook, Marat Dukhan, Casey Dvorak, Kevin Fives, Vlad Fomenko, Timur Garipov, Kristian Georgiev, Mia Glaese, Tarun Gogineni, Adam Goucher, Lukas Gross, Katia Gil Guzman, John Hallman, Jackie Hehir, Johannes Heidecke, Alec Helyar, Haitang Hu, Romain Huet, Jacob Huh, Saachi Jain, Zach Johnson, Chris Koch, Irina Kofman, Dominik Kundel, Jason Kwon, Volodymyr Kyrylov, Elaine Ya Le, Guillaume Leclerc, James Park Lennon, Scott Lessans, Mario Lezcano-Casado, Yuanzhi Li, Zhuohan Li, Ji Lin, Jordan Liss, Lily, Liu, Jiancheng Liu, Kevin Lu, Chris Lu, Zoran Martinovic, Lindsay McCallum, Josh McGrath, Scott McKinney, Aidan McLaughlin, Song Mei, Steve Mostovoy, Tong Mu, Gideon Myles, Alexander Neitz, Alex Nichol, Jakub Pachocki, Alex Paino, Dana Palmie, Ashley Pantuliano, Giambattista Parascandolo, Jongsoo Park, Leher Pathak, Carolina Paz, Ludovic Peran, Dmitry Pimenov, Michelle Pokrass, Elizabeth Proehl, Huida Qiu, Gaby Raila, Filippo

Raso, Hongyu Ren, Kimmy Richardson, David Robinson, Bob Rotsted, Hadi Salman, Suvansh Sanjeev, Max Schwarzer, D. Sculley, Harshit Sikchi, Kendal Simon, Karan Singhal, Yang Song, Dane Stuckey, Zhiqing Sun, Philippe Tillet, Sam Toizer, Foivos Tsimpourlas, Nikhil Vyas, Eric Wallace, Xin Wang, Miles Wang, Olivia Watkins, Kevin Weil, Amy Wendling, Kevin Whinnery, Cedric Whitney, Hannah Wong, Lin Yang, Yu Yang, Michihiro Yasunaga, Kristen Ying, Wojciech Zaremba, Wenting Zhan, Cyril Zhang, Brian Zhang, Eddie Zhang, and Shengjia Zhao. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

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

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang,

Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

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

Albert Tseng, Qingyao Sun, David Hou, and Christopher De Sa. Qtip: Quantization with trellises and incoherence processing, 2025. URL https://arxiv.org/abs/2406.11235.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

A Additional details

Hardware and software. All measurements use a single Grace-Blackwell GPU (compute capability sm100, aarch64 host) running SGLang with the FlashInfer MXFP4 grouped-MoE kernel, tensor parallelism 1, and CUDA graphs enabled (with the SM100 first-capture autotune disabled, held constant across methods). The model is the public `gpt-oss-20b` MXFP4 checkpoint (24 decoder layers, 32 experts per layer, top-4 routing, hidden size 2880; expert projections $w_1, w_3 : 2880 \to 2880$ gate/up and $w_2 : 2880 \to 2880$ down), resident at 14.66 GB.

QTIP baseline construction. We re-quantize all $24 \times 32 \times 3 = 2304$ expert matrices to the kernel-servable format (decode mode `quantlut_sym`, L=16, tlut_bits=9, V=2, K=3; aggregate 3.03 bpw), applying the mma-swizzle required by the `decompress_matvec` kernel, which we build for the 2880×2880 shape. Two experts with zero routed calibration tokens are packed with a layer-aggregate fallback. The QTIP mixture-of-experts is integrated behind the same SGLang quantization seam as the native path; the per-forward randomized Hadamard uses the real `fast_hadamard_transform` kernel. We validate the integration against an independent `torch` reference (per-linear relative ℓ_2 error 2.9×10^{-4}; full MoE block 2.7×10^{-3}; CUDA-graph replay bit-exact) before timing.

Benchmark protocol. Throughput is the median decode rate reported by the batched single-batch benchmark at 512-token prefill and 64-token decode, for batch sizes 1, 8, and 32, with CUDA graphs on for both methods. HEPH serving uses the unmodified native MXFP4 path (equivalently, decoded HEPH weights); the reported HEPH and QTIP numbers are from the same host and harness. QTIP uses its own kernels at every batch: the single-matvec trellis kernel at batch 1 and its multi-batch (tile-decode+MMA-reuse) kernel at batch 8/32 — upstream ships the latter only for 2-bit, so we ported it to 3 bits and to sm100 and verified its batch-1 output is bit-exact to the single-matvec oracle on our packed weights before timing.

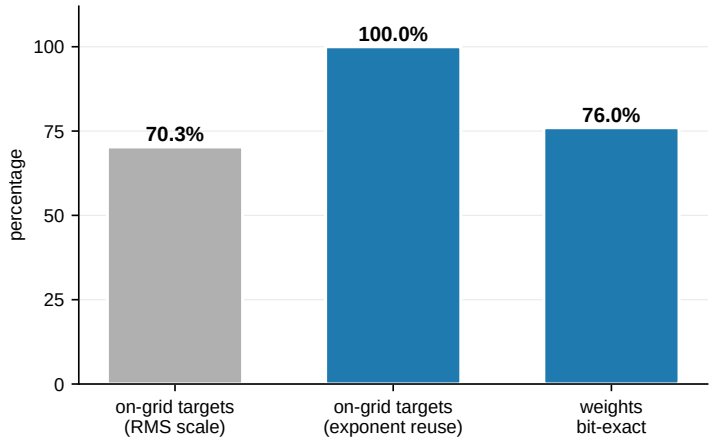


Figure 3: Reusing the checkpoint's UE8M0 exponent keeps 100% of encoding targets on the E2M1 atom grid (vs. 70.3% for a re-derived RMS scale), and the grid-constrained codebook reconstructs $\approx 76\%$ of weights identically to the source MXFP4 value — so the decoded tensor is on-grid MXFP4 at essentially no rate cost over the trellis labels, while the small residual is what separates HEPH's 80.03 accuracy from the 80.79 ceiling.

A.1 Proof of Proposition 1

Since $\max_{a\in\mathcal{A}}|a| = 6$, the convex hull of $\mathcal{S}(E)$ is the axis-aligned box $\mathrm{conv}(\mathcal{S}(E)) = \prod_i[-6s_i, 6s_i]$, writing s_i for the scale of coordinate i. Any orthogonal R fixing $\mathcal{S}(E)$ fixes its convex hull, hence is a symmetry of that box; the orthogonal symmetries of an axis-aligned box are exactly the signed coordinate permutations, and a coordinate may be sent only to one of equal half-width, i.e. equal scale. Because $\mathcal{A}$ is sign-symmetric and identical in every coordinate, these signed permutations do preserve $\mathcal{S}(E)$, so the orthogonal stabilizer of the grid contains nothing else. □

A.2 Serving cost decomposition

This section decomposes the equal-memory serving comparison of §4 (Table 3); all numbers are for the same GPU, harness, and CUDA-graph configuration.

Where the batch-1 time goes. At batch-1 decode the non-expert work (attention, router, norms, KV, sampling; identical `bf16` in both methods) accounts for ≈2.37 ms/token. On that shared base HEPH's in-kernel path costs 10.4 ms/token and QTIP 13.5. Both decode the *same* 4 experts $\times$ 24 layers$\times 3 = 288$ per-expert matrix–vectors every forward, and QTIP's individual decode kernel is in fact $2.2\times$ faster than HEPH's in isolation (below); QTIP is nonetheless slower end to end because of *everything around* those matvecs — a randomized Hadamard on every linear (576/token) and an unfused per-expert gather/SwiGLU/bias/accumulate — which HEPH's Hadamard-free on-grid gather avoids.

The Hadamard is not the bottleneck. One might attribute QTIP's cost to that per-forward Hadamard. It is not the dominant term: with QTIP's real fused `fast_hadamard_transform` batch-1 decode is 72.8 tok/s and with a dense matrix-product Hadamard 68.9 — the same $\sim$70 tok/s band, within run-to-run noise of the 74.2 of Table 3; only a naive pure-`torch` butterfly (36.4) is materially slower, and no serving stack uses it. The dominant cost is *structural*: per token QTIP issues 288 sequential `decompress_matvec` calls plus their per-expert epilogue *and* a Hadamard on every linear, whereas HEPH's decode is a pure on-grid gather.

Why the gap persists as batch grows. QTIP's multi-batch trellis kernel (decode each 16×16 tile once into registers, then reuse it across the batch via tensor-core MMA) amortizes the per-tile

decode, and once ported to our 3-bit rate its throughput scales up ($74.2 \to 97.2 \to 291.3$). HEPH's in-kernel decode scales up too ($96.1 \to 111.0 \to 448.6$) and stays 1.1–$1.5\times$ ahead for two reasons batching does not remove: HEPH decodes with a pure on-grid gather and then a standard dense GEMM, whereas QTIP runs a stateful per-tile decode+MMA plus its own gather/SwiGLU/accumulate; and QTIP still applies a Hadamard every forward.

Operator latency is not the serving metric. Timed alone with CUDA graphs — which remove a $2.05\,\mu$s per-launch floor that otherwise quantizes both methods — HEPH's fused decode+matvec runs at $11.1\,\mu$s on a 2880^2 matrix versus QTIP's $5.1\,\mu$s `decompress_matvec`: QTIP's *kernel* is $2.2\times$ faster, because its 512-entry codebook fits a conflict-free $32\times$-replicated shared table feeding a tensor-core pipeline, which HEPH's 4096-entry codebook cannot. Yet at equal memory HEPH is ahead end to end (Table 3): the fastest operator does not win serving — the per-forward Hadamard and unfused per-expert epilogue decide it, and HEPH pays neither.