E8-QAT: EFFICIENT 2-BIT QUANTIZATION-AWARE TRAINING WITH IMPLICIT LATTICE CODEBOOKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Low-bit quantization reduces the memory footprint of large language models, but preserving accuracy with efficient training remains challenging. Scalar quantization encodes weights independently, while explicit-codebook vector quantization introduces costly search during training. We propose E8-QAT, a 2-bit quantization-aware training framework that uses the E_8 lattice as an implicit eight-dimensional codebook. It jointly quantizes weight groups through arithmetic projection without storing or searching an explicit codebook during the training forward pass. A 2-bit core encoding and sparse extension preserve the selected lattice points exactly, adding less than 0.1% to core index storage. On Qwen3.6-35B-A3B, E8-QAT matches the performance of the full-precision model in terms of the average score across the four tasks. E8-QAT improves over the state-of-the-art baseline by 2.03% on IFEval and by 1.51% on LiveCodeBench, under the same data, schedule, and bit width. The 2-bit representation reduces peak memory by 80% and supports a $4\times$ larger batch. At each representation's largest feasible batch, E8-QAT reaches $1.97\times$ the decoding throughput of BF16.

1 INTRODUCTION

Large language models (LLMs) have made substantial progress across language tasks (Wu et al., 2023), but growing demands for privacy-preserving, offline, and cost-efficient deployment have made running them on resource-constrained platforms a pressing need (Yao et al., 2024; Liagkou et al., 2024). Weight quantization addresses this by reducing model size (Lin et al., 2024). Post-training quantization (PTQ) incurs low computational cost but can suffer accuracy degradation at ultra-low bit widths (Frantar et al., 2023). Quantization-aware training (QAT) recovers accuracy by allowing the model to adapt to quantization errors through weight updates, but training requires substantial data, compute, and memory (Jacob et al., 2018; Chen et al., 2025). This motivates quantization schemes that achieve strong accuracy at ultra-low bit widths without prohibitive training overhead.

Existing QAT approaches primarily rely on scalar quantization, which assigns quantization levels to each weight coordinate independently (Liu et al., 2025). While scalar quantization supports efficient per-coordinate arithmetic, it limits the flexibility of codeword arrangements across dimensions. Vector quantization (VQ) addresses this by jointly encoding groups of weights through a shared codebook (Gray, 1984), allowing richer codeword arrangements that better capture cross-dimensional weight structure and reduce quantization distortion at a given bit budget (Egiazarian et al., 2024; Tseng et al., 2024). Recent work has therefore explored integrating vector quantization into quantization-aware training (VQ-QAT) (Malinovskii et al., 2024; Wang et al., 2026a).

However, conventional VQ-QAT requires an explicit codebook that must be **stored and searched at every training step**, posing two challenges at scale: (1) nearest-codeword search during the training forward pass introduces substantial overhead, and (2) codebook size grows exponentially with vector dimension at a fixed bit width, making high-dimensional VQ impractical for both training and inference.

To address these challenges, we propose **E8-QAT**, a QAT framework that uses the E_8 lattice (Conway & Sloane, 1982) as an implicit eight-dimensional codebook, as illustrated in Figure 1. During the training forward pass, each eight-dimensional weight group is projected onto the E_8 lattice by rounding, parity correction, and a constant number of candidate comparisons, reducing the per-group computational cost from $O(v\,2^{kv})$ for exhaustive codebook search to $O(v)$, independent of the

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

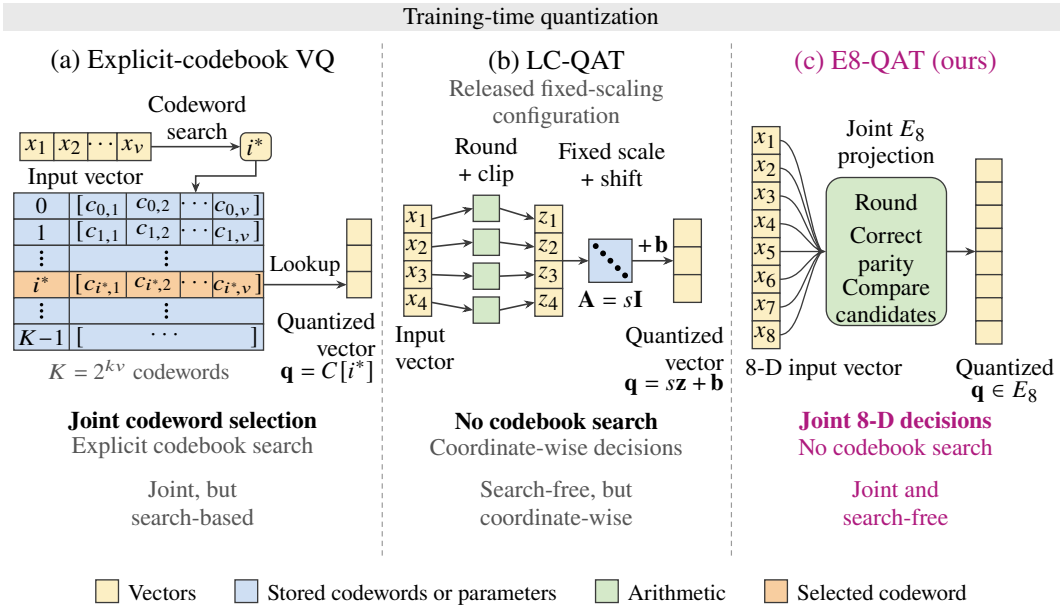


Figure 1: Training-time quantization. (a) Explicit-codebook vector quantization searches a stored codebook of $K = 2^{kv}$ entries. (b) LC-QAT rounds each coordinate and applies a fixed scale and shift, $\mathbf{q} = s\mathbf{z} + \mathbf{b}$. (c) E8-QAT projects eight normalized coordinates onto E_8 by arithmetic, without a stored codebook.

nominal codebook size, where k and v denote the bit width and vector dimension, respectively. Exploiting the observed concentration of quantized weight groups within the finite core, E8-QAT combines a 2-bit core encoding with a sparse extension that, at very low additional memory cost, exactly preserves the lattice points selected during training with an identity straight-through estimator (STE) (Bengio et al., 2013).

Our main contributions are as follows:

- **Search-free joint quantization for QAT.** We jointly quantize each group of eight weights by closed-form projection onto the infinite E_8 lattice in the QAT forward pass. The projection requires $O(v)$ operations per group, independent of the nominal codebook size, without codebook storage, search, or learning.
- **Exact export beyond the finite core.** Lattice projections during QAT can fall outside the finite core, and clipping these points at export would alter the trained quantized weights. We combine a dense core index with sparse extension records to preserve every trained lattice point exactly. At the end of training, the sparse records add less than 0.1% to the 2-bit index payload.
- **Matched-budget 2-bit evaluation on agentic and long-form tasks.** We evaluate the performance and efficiency of E8-QAT on Qwen3.6-35B-A3B (Qwen Team, 2026) and Llama-3.1-8B (Grattafiori et al., 2024). On Qwen3.6-35B-A3B, E8-QAT improves over LC-QAT by 2.03% on IFEval and by 1.51% on LiveCodeBench, under the same training corpus, schedule, and bit budget. Its four-task average is 73.93, equal to the reported BF16 average. Relative to BF16, the exported model reduces peak memory usage by 80%, enabling a $4\times$ larger batch size. Decoding throughput reaches $1.97\times$ that of BF16 when each representation is evaluated at its largest feasible batch size.

2 Related Work

2.1 Low-bit weight quantization

Representative PTQ methods include GPTQ (Frantar et al., 2023) and AWQ (Lin et al., 2024), which use calibration data to guide scalar quantization of individual weight coordinates. Other PTQ methods

use VQ to jointly encode groups of weights. QuIP# (Tseng et al., 2024) uses predefined codebooks, while AQLM (Egiazarian et al., 2024) represents each weight group as a sum of codewords from learned codebooks. QAT methods such as EfficientQAT (Chen et al., 2025) and ParetoQ (Liu et al., 2025) incorporate scalar quantization into training, allowing model weights to adapt to quantization errors.

2.2 Vector quantization-aware training

Recent work has also explored VQ-QAT, with PV-Tuning and LC-QAT adopting different strategies for updating quantized weights.

PV-Tuning optimizes both continuous parameters and discrete codeword assignments. For learned VQ codebooks, it combines gradient-based updates of codeword values with discrete assignment updates for selected weight groups. These assignment updates require nearest-codeword search, or beam search when multiple additive codebooks are used, introducing additional computational cost during training (Malinovskii et al., 2024).

LC-QAT avoids explicit codebook lookup in the training forward pass through an affine mapping of discrete vectors (Wang et al., 2026a). However, it rounds and clips each coordinate independently before applying a fixed scale and shift (AI9Stars, 2026). The resulting quantizer retains the per-coordinate structure of scalar quantization, limiting the flexibility of codeword arrangements across dimensions.

Consequently, these VQ-QAT designs face a trade-off between training efficiency and quantization accuracy. They either rely on costly codeword search to determine the discrete assignments, or give up the joint encoding that makes VQ accurate. E8-QAT addresses this challenge by using the E_8 lattice as an implicit eight-dimensional codebook, enabling joint weight quantization through arithmetic projection (Conway & Sloane, 1982) without storing or searching an explicit codebook during the training forward pass.

3 Preliminaries

At a fixed bit budget, quantization error depends on both the distribution of the weights and the geometry of the codebook. We first explain how Hadamard preprocessing redistributes weight magnitudes, then introduce vector quantization as a way to represent the transformed weights jointly. These considerations motivate the use of the E_8 lattice, whose geometric properties and efficient nearest-point projection provide the mathematical basis for E8-QAT.

3.1 Hadamard Preprocessing

Randomized Hadamard preprocessing mixes weight coordinates before quantization (Tseng et al., 2024). Within a transform block, it maps a weight vector $\mathbf{w}$ to

$$\widetilde{\mathbf{w}} = \mathbf{HDw}, \qquad \mathbf{H}^\top \mathbf{H} = \mathbf{I}, \tag{1}$$

where $\mathbf{D}$ applies independent random sign flips and $\mathbf{H}$ is a normalized Hadamard matrix. Each output coordinate is a normalized sum of signed weights from the same block, while the transform preserves the sum of squared weights.

The central limit theorem (Durrett, 2019) shows that when many independent signed weights contribute to each output coordinate and no single weight dominates, the resulting marginal distribution approximates a Gaussian distribution. To verify this, we sampled 14 weight tensors from a pre-quantized BF16 model and observed that the transformed marginal kurtosis is about 3.02, close to the Gaussian value 3.

Under the zero-mean isotropic Gaussian approximation, the probability density depends only on the distance from the origin and decreases with that distance. We observed that the radial probability distribution of the rotated weights is geometrically similar to a finite E_8 codebook (Tseng et al., 2024), which can improve lattice utilization and representation accuracy in vector quantization.

3.2 VECTOR QUANTIZATION

After Hadamard preprocessing and scaling, we partition each row of a weight matrix into non-overlapping groups of d consecutive weights along the input dimension. Let $\mathbf{x} \in \mathbb{R}^d$ denote one such group. Given a finite codebook $\mathcal{C} \subset \mathbb{R}^d$ with K codewords, vector quantization assigns the entire group to its nearest codeword:

$$Q_{\mathcal{C}}(\mathbf{x}) = \arg\min_{\mathbf{c} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|_2^2. \tag{2}$$

With $K = 2^{kd}$ codewords, each group requires a kd-bit index.

Unlike scalar quantization, VQ is not restricted to a rectangular grid and can better match the distribution of weight groups. This flexibility can reduce quantization error at the same index budget (Gray & Neuhoff, 1998).

However, exhaustive nearest-codeword search costs $O(dK)$ operations per group, with $K = 2^{kd}$ growing exponentially in the group dimension at a fixed per-weight bit budget. This motivates structured point sets whose nearest points can be computed directly rather than found by exhaustive search.

3.3 E_8 LATTICE QUANTIZATION

The E_8 lattice is a classical eight-dimensional construction from sphere-packing and lattice-quantization theory (Conway & Sloane, 1999), not a codebook learned from model weights. Its points have either all integer or all half-integer coordinates, with an even coordinate sum:

$$E_8 = \left\{ \mathbf{z} \in \mathbb{Z}^8 \cup \left(\mathbb{Z}^8 + \tfrac{1}{2}\mathbf{1}\right) : \mathbf{1}^\top \mathbf{z} \in 2\mathbb{Z} \right\}, \tag{3}$$

where $\mathbf{1}$ is the all-ones vector. Its geometry and algebraic structure offer three useful properties.

E_8 combines favorable covering geometry with an algebraic structure that enables efficient exact quantization. At equal cell volume, its smaller covering radius than $\mathbb{Z}^8$ provides a tighter worst-case error bound throughout the infinite lattice (Conway & Sloane, 1999), while its structure permits exact nearest-point projection without exhaustive codebook search (Conway & Sloane, 1982). This combination makes E_8 attractive for QAT, where weights are repeatedly projected as they evolve during training. We exploit these properties by using an approximately ball-shaped core $\mathcal{C}_{\text{core}} \subset E_8$ of 2^{16} low-norm points for compact encoding, while retaining the full lattice as the training-time projection space. This design pairs 2-bit-per-weight indices for core points with the flexibility to project beyond the core boundary; the following section describes how these core-external projections are encoded and their additional cost is accounted for.

4 METHOD

E8-QAT separates training-time quantization from post-training encoding, combining efficient joint quantization with compact deployment. As illustrated in Figure 2, the framework starts from finite-core PTQ codewords, performs lattice QAT with an identity straight-through estimator, and preserves the final projections using core indices and sparse extensions.

4.1 LATTICE QAT FROM PTQ INITIALIZATION

Let $\mathbf{L} \in \mathbb{R}^{m \times n}$ denote the trainable latent weights of a layer with n input and m output channels. Assuming n is divisible by $d = 8$, we partition each row into contiguous groups $\boldsymbol{\ell}_g \in \mathbb{R}^d$. The quantization mapping is

$$\mathbf{x}_g = w_s \boldsymbol{\ell}_g, \qquad \mathbf{q}_g = Q_{E_8}(\mathbf{x}_g), \qquad \overline{\mathbf{w}}_g = c_s \mathbf{q}_g + c_z \mathbf{1}_d, \tag{4}$$

where group g contains $d = 8$ latent weights $\boldsymbol{\ell}_g$. The fixed per-tensor scale w_s maps them to lattice coordinates $\mathbf{x}_g$, and Q_{E_8} returns the nearest lattice point $\mathbf{q}_g$. The fixed decode scale c_s and zero-point c_z map that point back to weights; $\mathbf{1}_d$ adds c_z to every coordinate.

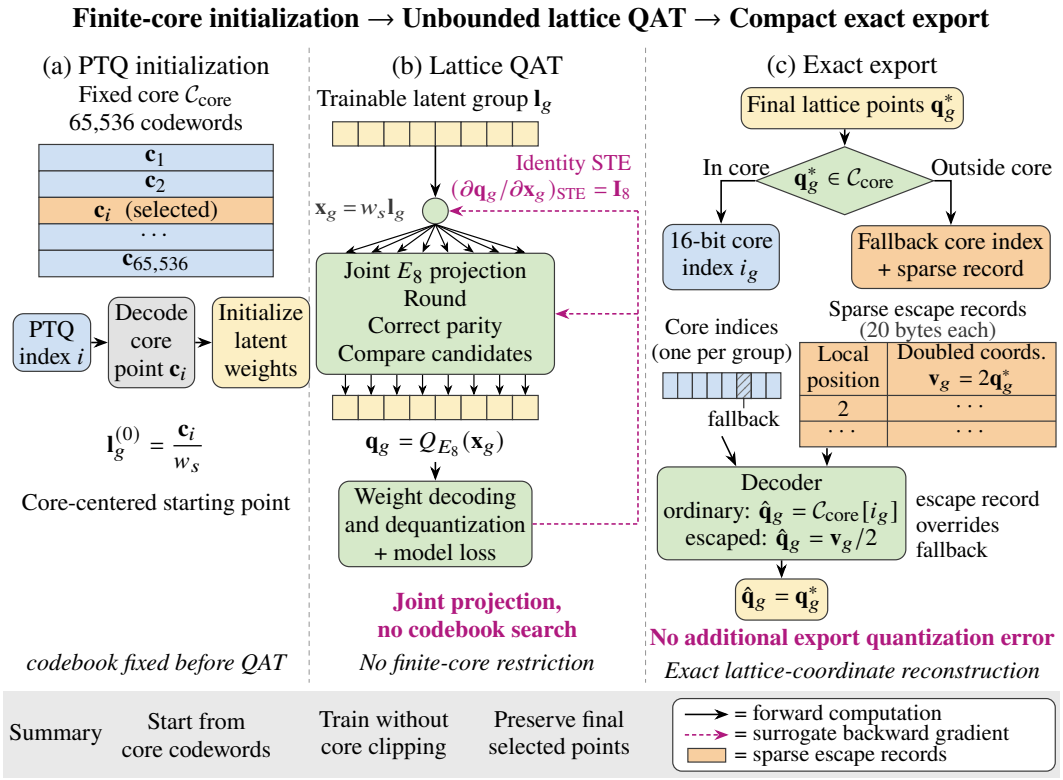


Figure 2: Overview of E8-QAT. (a) PTQ initializes latent weights from a fixed E_8 core. (b) Arithmetic lattice projection with an identity STE enables joint QAT without explicit codebook search or finite-core clipping. (c) Core indices and sparse escape records preserve the final lattice points. Hadamard transforms and reconstruction scales are omitted for clarity.

Weight reconstruction. The decoded groups in Equation 4 are assembled into $\overline{\mathbf{W}} \in \mathbb{R}^{m \times n}$. The effective weight matrix is reconstructed as

$$\widehat{\mathbf{W}} = \operatorname{diag}(\mathbf{s}_V)\, H_{\text{out}}\, \overline{\mathbf{W}}\, H_{\text{in}}\, \operatorname{diag}(\mathbf{s}_U), \tag{5}$$

where $\mathbf{s}_U \in \mathbb{R}^n$ and $\mathbf{s}_V \in \mathbb{R}^m$ are trainable input- and output-channel scaling vectors, respectively. The matrices H_{in} and H_{out} are fixed orthogonal block-Hadamard transforms with block size 128. For an input $\mathbf{a} \in \mathbb{R}^n$, the layer computes $\mathbf{y} = \widehat{\mathbf{W}}\mathbf{a}$.

Finite-core construction. The finite core $\mathcal{C}_{\text{core}}$ consists of the first 2^{16} points of E_8 ordered by increasing squared norm, with ties resolved lexicographically. It includes all 56,881 points with squared norm at most 10 and the first 8,655 points in lexicographic order from the shell at squared norm 12. Initialization, export, and decoding use the same point set and index ordering. Since the outermost shell is only partially retained, a norm threshold does not suffice for escape detection; a projected point is classified as an escape if and only if it lies outside $\mathcal{C}_{\text{core}}$.

Numerical precision. Latent weights are stored in BF16, while projection candidates and squared distances are evaluated in FP32. Gradients through the hard projection use an identity straight-through estimator. Hadamard transforms, affine decoding, and channel scaling lie outside the projector and use ordinary backpropagation.

PTQ initialization. We initialize from an E_8-based PTQ checkpoint obtained using Hadamard preprocessing and LDLQ (Tseng et al., 2024; Chee et al., 2023). PTQ uses a fixed core $\mathcal{C}_{\text{core}} = \{\mathbf{c}_i\}_{i=0}^{2^{16}-1} \subset E_8$, shared with the export representation. Given the PTQ index $i_g^{(0)}$, we initialize

$$\boldsymbol{\ell}_g^{(0)} = \frac{\mathbf{c}_{i_g^{(0)}}}{w_s}. \tag{6}$$

Thus, $\mathbf{x}_g^{(0)} = \mathbf{c}_{i_g^{(0)}}$ in exact arithmetic. This places each group at a quantization-cell center, providing a local stability margin against small perturbations (Conway & Sloane, 1999). The finite-precision condition is given in Appendix A; subsequent optimization is not constrained to remain within the core.

Lattice training. Each forward pass applies the arithmetic projector in Equation 4. Rounding, parity correction, and comparison of two candidates require $O(d)$ operations per group (Conway & Sloane, 1982), without explicit codebook search or restriction to the finite core. Since hard projection is piecewise constant, we use an identity STE (Bengio et al., 2013; van den Oord et al., 2017). For the autoregressive language-modeling loss $\mathcal{L}$, the forward computation and surrogate latent gradient are

$$\mathbf{q}_g = \mathbf{x}_g + \mathrm{sg}(Q_{E_8}(\mathbf{x}_g) - \mathbf{x}_g)\,, \frac{\partial \mathcal{L}}{\partial \boldsymbol{\ell}_g} \quad \approx w_s c_s \frac{\partial \mathcal{L}}{\partial \overline{\mathbf{w}}_g}, \tag{7}$$

where sg is the identity in the forward pass and has zero derivative in the backward pass. This substitutes I_d for the projection Jacobian while retaining the hard forward assignment.

The assignment therefore remains jointly constrained by E_8, while latent updates can move groups across Voronoi boundaries. We optimize the latent weights, per-channel scales, and other trainable parameters without learning the lattice or its core codewords. The STE supplies a surrogate gradient rather than the true derivative of hard projection.

4.2 Exact Encoding with Sparse Extensions

Training on the infinite lattice permits projections outside the finite core. Let $\mathbf{q}_g^\star = Q_{E_8}(\mathbf{x}_g^{(T)})$ denote the final projection after T training steps. For G quantized groups, define

$$\mathcal{E} = \left\{ g : \mathbf{q}_g^\star \notin \mathcal{C}_{\mathrm{core}} \right\}, \qquad \rho = \frac{|\mathcal{E}|}{G}. \tag{8}$$

An escape is determined by core membership, not solely by a norm threshold. Reprojecting escaped points onto the core would alter the quantized weights selected during training.

Instead, we store a 16-bit core index i_g for every group. For an in-core group, it represents $\mathbf{q}_g^\star$ directly; for an escaped group, it identifies a legal fallback. Each escape additionally stores its position and doubled coordinates $\mathbf{v}_g = 2\mathbf{q}_g^\star \in \mathbb{Z}^d$. Decoding gives

$$\widehat{\mathbf{q}}_g = \begin{cases} \mathbf{c}_{i_g}, & g \notin \mathcal{E}, \\ \mathbf{v}_g/2, & g \in \mathcal{E}. \end{cases} \tag{9}$$

The sparse record overrides the fallback rather than adding absolute coordinates to it. When positions and doubled coordinates fit the representation, $\widehat{\mathbf{q}}_g = \mathbf{q}_g^\star$ for every group. Export therefore introduces no additional lattice-coordinate quantization error.

With B additional bytes per escape, the payload cost is

$$b_{\mathrm{payload}} = \frac{16G + 8B|\mathcal{E}|}{dG} = 2 + \frac{8B}{d}\rho \quad \text{bits per quantized weight.} \tag{10}$$

Our coordinate format uses $B = 20$, giving $b_{\mathrm{payload}} = 2 + 20\rho$ for $d = 8$. This accounts for core indices and escape records, excluding shared codebooks, scales, and format metadata. Appendix B details the representation, range checks, and optional root packing. Section 5.2 measures the encoding overhead empirically.

After export, latent weights are discarded, leaving core indices, sparse records, and reconstruction parameters. Inference may use a shared decoding table, whereas training-time projection remains lookup-free.

5 Experiments

5.1 Experimental Setup

We summarize the experimental configuration below and provide further details in Appendix C.1.

Table 1: Results at 2 bits on Qwen3.6-35B-A3B. Bold marks the best quantized result in each column. Full results, including ARC-Easy, BoolQ, are given in Appendix C.3.

			Base (lm-eval)					Agent		IF
Type	Method	Bits	ARC-C	HelS	PIQA	WinG	Avg.	LCB	BFCL	IFEval
N/A	BF16	16	55.20	83.49	82.75	74.27	73.93	81.42	75.99	85.40
SQ	RTN	2	49.23	79.44	79.87	69.85	69.60	20.38	53.91	77.82
	GPTQ	2	52.78	79.17	80.20	69.76	70.48	20.38	55.87	78.18
	UniSVQ	2	56.83	78.31	80.58	71.59	71.83	31.28	72.32	80.41
VQ	QuIP#	2	56.66	77.25	80.20	72.77	71.72	47.77	67.31	81.70
	QTIP	2	51.71	79.36	81.12	72.69	71.22	26.45	61.77	80.78
	AQLM	2	54.95	79.59	81.50	73.32	72.34	52.32	75.14	81.52
	E8-PTQ	2	56.74	79.38	80.47	72.61	72.30	55.73	**76.14**	81.89
VQ-QAT	PV-Tuning	2	56.57	80.89	**82.15**	73.48	73.27	75.73	74.83	81.89
	LC-QAT	2	56.57	81.28	80.74	73.88	73.12	78.58	75.22	80.96
	E8-QAT	2	**57.85**	**82.29**	81.56	**74.03**	**73.93**	**80.09**	74.16	**82.99**

		Base (lm-eval)				
Type	Method	ARC-C	HelS	PIQA	WinG	Avg.
N/A	BF16	55.55	79.55	81.28	73.80	72.54
SQ	RTN	25.34	26.54	51.90	51.70	38.87
	GPTQ	24.40	26.38	50.98	50.04	37.95
	UniSVQ	39.51	63.55	72.63	66.46	60.54
VQ	E8-PTQ	43.94	66.16	74.76	68.90	63.44
VQ-QAT	LC-QAT	40.10	67.71	73.29	67.40	62.12
	E8-QAT	**44.62**	**70.03**	**76.17**	**69.61**	**65.11**

Table 2: Results at 2 bits on Llama-3.1-8B. Bold marks the best quantized result in each column. Full results, including ARC-Easy, BoolQ, and WikiText perplexity, are given in Appendix C.4.

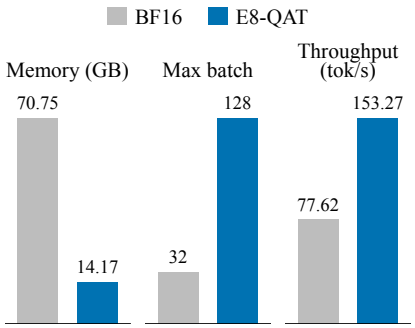


Figure 3: Serving capacity. Memory is measured at batch 1 and throughput at each model's largest feasible batch.

Baselines. We compare E8-QAT against two categories of 2-bit methods: (1) post-training quantization, including round-to-nearest (RTN) and GPTQ;UniSVQ (Wang et al., 2026b; AI9Stars, 2026), the scalar PTQ used to initialize LC-QAT; AQLM with a single learned 2^{16}-entry codebook shared across all feed-forward matrices of a layer, which matches our bit width; and E8-PTQ, the first stage of our method; and (2) quantization-aware training, including PV-Tuning, initialized from the same AQLM model, and LC-QAT.

Models, data, and evaluation. We use Qwen3.6-35B-A3B and Llama-3.1-8B as our base models. Both stages draw on the UltraData collection (OpenBMB, 2026): stage-1 calibration uses 1,680 sequences of 16,384 tokens constructed from 2,000 agentic conversations in UltraData-SFT-Agent, shared by all PTQ methods, and stage-2 training uses 2,000 long sequences from the same collection, totaling 50.8M tokens per pass, with a 375-step schedule for Llama-3.1-8B. We evaluate model performance with lm-evaluation-harness (Gao et al., 2024) on four zero-shot benchmarks, ARC-Challenge (ARC-C) (Clark et al., 2018), HellaSwag (HelS) (Zellers et al., 2019), PIQA (Bisk et al., 2020), and WinoGrande (WinG) (Sakaguchi et al., 2020), together with LiveCodeBench (LCB) (Jain et al., 2025) pass@1, BFCL (Patil et al., 2025) v3 overall accuracy, IFEval prompt-level strict accuracy (Zhou et al., 2023), and GPQA (Rein et al., 2024) accuracy. All models are evaluated with the same harness and settings; details are given in Appendix C.1.

Implementation details. All Qwen3.6-35B-A3B training runs use eight GPUs. E8-QAT and LC-QAT use 750 steps, a warmup–stable–decay schedule annealed to zero, and a peak learning rate of 10^{-4}. The Llama-3.1-8B second stage uses 375 steps, with the peak learning rate scaled by the

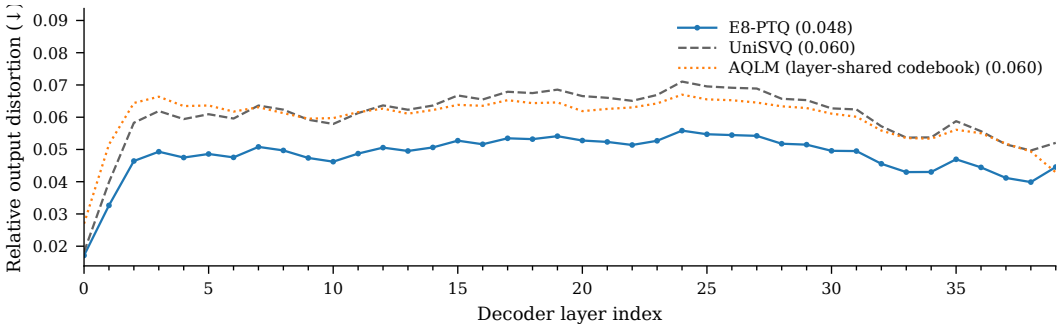


Figure 4: Layer-wise stage-1 output distortion at matched bit width (lower is better).

inverse weight scale. PV-Tuning uses its released recipe. In paired 375-step runs, the E8-QAT stage completes 4.5% faster than LC-QAT. Inference is measured with fused decode kernels and 16-bit indices, launching a separate process for each configuration.

5.2 Main Results

Comparison with 2-bit methods. To evaluate whether the lattice codebook yields the strongest 2-bit model at a matched bit budget, we compare E8-QAT against the PTQ and QAT baselines on Qwen3.6-35B-A3B. As shown in Table 1, E8-QAT achieves the best quantized score on ARC-Challenge, HellaSwag, and WinoGrande, and on LiveCodeBench and IFEval. Compared with LC-QAT, which shares the same data, schedule, and bit width, E8-QAT improves accuracy by 1.51 percentage points on LiveCodeBench and by 2.03 percentage points on IFEval.

On the six-task average, E8-QAT scores 76.95, against 75.97 for LC-QAT. On Llama-3.1-8B, its WikiText perplexity is 13.77, against 16.73 for LC-QAT.Appendix C.4 gives ARC-Easy, BoolQ, and the perplexity for every Llama row. These results demonstrate that replacing the scalar grid with the E_8 lattice improves both base-model quality and agentic capability at 2 bits.

Generalization to dense models. To examine whether this advantage transfers across architectures, we further evaluate E8-QAT on the dense Llama-3.1-8B. As shown in Table 5.1, E8-QAT again achieves the best quantized score on all four tasks, outperforming LC-QAT by 2.99% on average. These results confirm that the lattice advantage transfers across architectures once the learning rate is matched in lattice space.

Inference efficiency. We compare the serving capacity of E8-QAT and BF16 to validate that 2-bit storage allows a single GPU to serve larger batches As shown in Figure 3, E8-QAT reduces peak memory by 80% relative to BF16, supports a $4\times$ larger batch, and achieves 97% higher throughput at each model's largest feasible batch. These results indicate that the lattice model nearly doubles serving throughput on the same hardware.

Most accurate initialization. At equal bits, the lattice should reconstruct weights more accurately than a scalar grid or a learned codebook. We measure the layer-wise output distortion , which weighted by routed-token frequency, with the same Hessians for all methods. As shown in Figure 4, E8-PTQ has the lowest distortion in 39 of the 40 layers. Averaged over layers, it is 20.4% below UniSVQ, which runs the same LDLQ pipeline with a scalar grid, and 19.4% below AQLM. These results confirm that the lattice provides the most accurate starting point at matched bits.

Gains from second-stage QAT. Reconstruction-based PTQ is expected to preserve short-form ability but not long-form generation, which QAT should recover. To verify this, Figure 5 compares each pipeline before and after the 750-step second stage. QAT improves LiveCodeBench by 24.36% and GPQA by 17.67% for E8-QAT. E8-QAT is ahead of the LC-QAT pipeline both before and after this stage. These results demonstrate that second-stage QAT is necessary for long-form generation at 2 bits and that the lattice advantage persists through training.

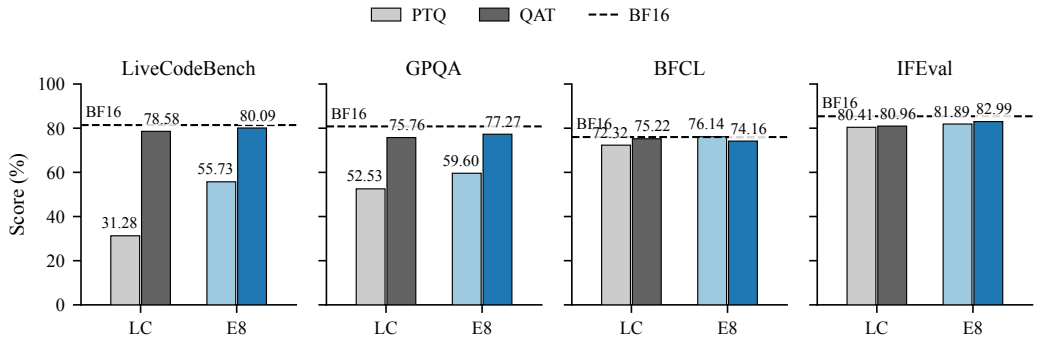


Figure 5: Performance of PTQ and QAT model with the 750-step second stage. Dashed lines mark BF16.

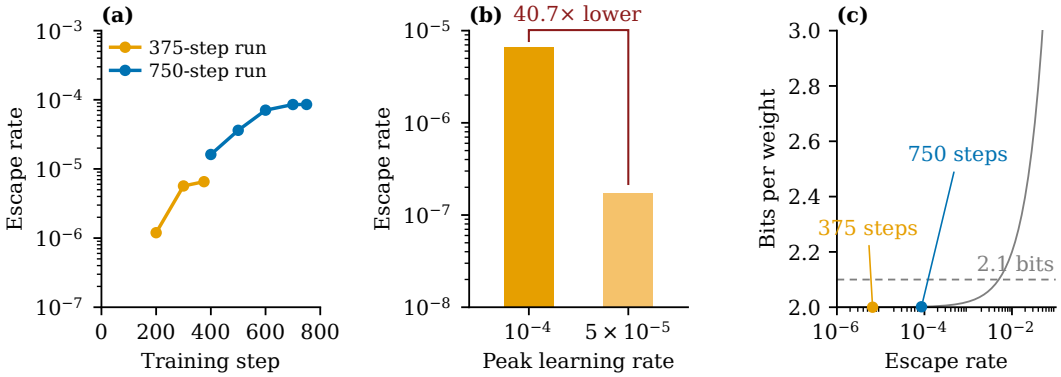


Figure 6: Escape trajectories (a), peak-learning-rate control at 375 steps (b), and the index-and-extension payload $b = 2 + 20\rho$ (c).

Exact export at two bits. Allowing QAT to move groups outside the finite core is expected to incur little additional storage. To quantify this overhead, Figure 6 tracks the escape rate during training, its sensitivity to the peak learning rate, and the resulting payload. Escape growth slows as the learning rate anneals, with the final 50 steps of the 750-step run raising the escape rate by only 0.25%; halving the peak learning rate reduces the final escape rate by $40.7\times$; and at the 750-step endpoint, the sparse records add only 0.086% to the 2-bit index payload. These results confirm that a finite core with a sparse extension keeps exact export within 0.1% of two bits, preserving the trained weights exactly with only a small number of exceptions.

6 Conclusion

In this work, we introduce E8-QAT, a QAT framework designed to address the trade-off between training efficiency and quantization accuracy at ultra-low bit widths. By using the E_8 lattice as an implicit eight-dimensional codebook, E8-QAT enables joint weight quantization through arithmetic projection without storing or searching an explicit codebook during the training forward pass. An identity STE supports weight updates during training. To preserve the lattice points selected during training, we combine a 2-bit core encoding with a sparse extension, enabling exact encoding at very low additional memory cost in our experiments.

Experimental results on Qwen3.6-35B-A3B and Llama-3.1-8B show that E8-QAT achieves higher average accuracy than LC-QATon the four zero-shot benchmarks. On Qwen3.6-35B-A3B, E8-QAT also reduces peak memory by 80% relative to BF16 at batch size 1 and achieves $1.97\times$ the BF16 serving throughput on the same hardware, with throughput measured at each model's largest feasible batch.

AI USE STATEMENT

Generative AI tools were used to aid and polish the writing. This included language editing, translation, LaTeX formatting, and drafting the presentation of figures and tables. The authors reviewed all AI-assisted text and independently checked the mathematical derivations, experimental claims, citations, and generated artifacts. The authors take responsibility for the final content.

REPRODUCIBILITY STATEMENT

The model, data processing, quantization pipeline, training configurations, export format, kernel validation, and evaluation protocols are described in the main text and appendix. We plan to provide anonymized code and the artifacts needed to reproduce the reported tables and figures during review.

REFERENCES

AI9Stars. UniSVQ/LC-QAT: Official implementation. GitHub repository, 2026. URL https://github.com/AI9Stars/UniSVQ/tree/449e8e4.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. doi: 10.48550/arXiv.1308.3432. URL https://arxiv.org/abs/1308.3432.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020. doi: 10.1609/aaai.v34i05.6239. URL https://doi.org/10.1609/aaai.v34i05.6239.

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*, volume 36, pp. 4396–4429, 2023. URL https://arxiv.org/abs/2307.13304.

Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. EfficientQAT: Efficient quantization-aware training for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10081–10100. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.498. URL https://aclanthology.org/2025.acl-long.498/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

John H. Conway and Neil J. A. Sloane. Fast quantizing and decoding algorithms for lattice quantizers and codes. *IEEE Transactions on Information Theory*, 28(2):227–232, 1982. doi: 10.1109/TIT.1982.1056484. URL https://doi.org/10.1109/TIT.1982.1056484.

John H. Conway and Neil J. A. Sloane. *Sphere Packings, Lattices and Groups*. Springer, 3rd edition, 1999. doi: 10.1007/978-1-4757-6568-7.

Rick Durrett. *Probability: Theory and Examples*, volume 49 of *Cambridge Series in Statistical and Probabilistic Mathematics*. Cambridge University Press, 5 edition, 2019. doi: 10.1017/9781108591034. URL https://doi.org/10.1017/9781108591034.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 12284–12303. PMLR, 2024. URL https://proceedings.mlr.press/v235/egiazarian24a.html.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.17323.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 2024. URL https://zenodo.org/records/12608602.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/arXiv.2407.21783. URL https://arxiv.org/abs/2407.21783.

Robert M. Gray. Vector quantization. *IEEE ASSP Magazine*, 1(2):4–29, 1984. doi: 10.1109/MASSP.1984.1162229. URL https://doi.org/10.1109/MASSP.1984.1162229.

Robert M. Gray and David L. Neuhoff. Quantization. *IEEE Transactions on Information Theory*, 44 (6):2325–2383, 1998. doi: 10.1109/18.720541.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2704–2713, 2018. URL https://openaccess.thecvf.com/content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Vasiliki Liagkou, Evangelia Filiopoulou, George Fragiadakis, Mara Nikolaidou, and Christos Michalakelis. The cost perspective of adopting large language model-as-a-service. In *2024 IEEE International Conference on Joint Cloud Computing (JCC)*, pp. 80–83. IEEE, 2024. doi: 10.1109/JCC62314.2024.00020. URL https://doi.org/10.1109/JCC62314.2024.00020.

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 on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 87–100, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html.

Zechun Liu, Changsheng Zhao, Hanxian Huang, Sijia Chen, Jing Zhang, Jiawei Zhao, Scott Roy, Lisa Jin, Yunyang Xiong, Yangyang Shi, Lin Xiao, Yuandong Tian, Bilge Soran, Raghuraman Krishnamoorthi, Tijmen Blankevoort, and Vikas Chandra. ParetoQ: Improving scaling laws in extremely low-bit LLM quantization. In *Advances in Neural Information Processing Systems*, volume 38, pp. 91311–91336. Curran Associates, Inc., 2025. doi: 10.52202/085713-3055. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/83b17fb3369b1effa97ca5409526b02e-Abstract-Conference.html.

Vladimir Malinovskii, Denis Mazur, Ivan Ilin, Denis Kuznedelev, Konstantin Burlachenko, Kai Yi, Dan Alistarh, and Peter Richtárik. PV-Tuning: Beyond straight-through estimation for extreme LLM compression. In *Advances in Neural Information Processing Systems*, volume 37, pp. 5074–5121. Curran Associates, Inc., 2024. doi: 10.52202/079017-0165. URL https://papers.nips.cc/paper_files/paper/2024/hash/091166620a04a289c555f411d8899049-Abstract-Conference.html.

OpenBMB. Data science and technology towards AGI part I: Tiered data management. *arXiv preprint arXiv:2602.09003*, 2026.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392. PMLR, 2025. URL https://proceedings.mlr.press/v267/patil25a.html.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level Google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8732–8740, 2020. doi: 10.1609/aaai.v34i05.6399. URL https://doi.org/10.1609/aaai.v34i05.6399.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. QuIP#: Even better LLM quantization with hadamard incoherence and lattice codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 48630–48656. PMLR, 2024. URL https://proceedings.mlr.press/v235/tseng24a.html.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://arxiv.org/abs/1711.00937.

Haoyu Wang, Xingyu Yu, Haiyan Zhao, Fengxiang Wang, and Xu Han. LC-QAT: Data-efficient 2-bit QAT for LLMs via linear-constrained vector quantization. *arXiv preprint arXiv:2606.10531*, 2026a. doi: 10.48550/arXiv.2606.10531. URL https://arxiv.org/abs/2606.10531.

Haoyu Wang, Haiyan Zhao, Xingyu Yu, Zhangyang Yao, Xu Han, Zhiyuan Liu, and Maosong Sun. UniSVQ: 2-bit unified scalar-vector quantization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026b. URL https://arxiv.org/abs/2606.10520.

Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. A brief overview of ChatGPT: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica*, 10(5):1122–1136, 2023. doi: 10.1109/JAS.2023.123618. URL https://doi.org/10.1109/JAS.2023.123618.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211, 2024. doi: 10.1016/j.hcc.2024.100211. URL https://doi.org/10.1016/j.hcc.2024.100211.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472/.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. doi: 10.48550/arXiv.2311.07911. URL https://arxiv.org/abs/2311.07911.

A Initialization and Implementation Details

Geometric margin over the integer lattice. At equal cell volume, E_8 has covering radius smaller than $\mathbb{Z}^8$ by a factor of $\sqrt{2}$, and its normalized second moment is 0.0716821 against $1/12$ for $\mathbb{Z}^8$ (Conway & Sloane, 1999). This is a granular gain of $0.654\,\text{dB}$, or a 14.0% reduction in mean squared error at matched rate under a high-resolution model.

Local stability of initialization. In the standard normalization of E_8, the minimum distance between distinct E_8 points is $\sqrt{2}$. Let

$$\epsilon_g^{(0)} = \left\| \mathbf{x}_g^{(0)} - \mathbf{c}_{i_g^{(0)}} \right\|_2$$

denote the finite-precision initialization error of group g. The initial assignment is preserved at step t whenever

$$\epsilon_g^{(0)} + \left\| \mathbf{x}_g^{(t)} - \mathbf{x}_g^{(0)} \right\|_2 < \frac{1}{\sqrt{2}} \quad \Longrightarrow \quad \mathbf{q}_g^{(t)} = \mathbf{c}_{i_g^{(0)}}. \tag{11}$$

To see this, write $\mathbf{c} = \mathbf{c}_{i_g^{(0)}}$. The triangle inequality gives $\delta = \|\mathbf{x}_g^{(t)} - \mathbf{c}\|_2 < 1/\sqrt{2}$. Every other lattice point $\mathbf{c}'$ then satisfies

$$\|\mathbf{x}_g^{(t)} - \mathbf{c}'\|_2 \geq \sqrt{2} - \delta > \delta,$$

so $\mathbf{c}$ remains the unique nearest point.

B Sparse Encoding and Decoding Verification

Coordinate records. Each eight-weight group retains a 16-bit core index. An escaped group additionally stores its 32-bit tensor-local position and doubled lattice coordinates,

$$\mathbf{v}_g = 2\mathbf{q}_g^\star \in \mathbb{Z}^8. \tag{12}$$

Every E_8 point has all-integer or all-half-integer coordinates, so $\mathbf{v}_g$ is integral. Eight signed 16-bit coordinates and one 32-bit position occupy 20 bytes per record. During decoding, the record supplies the absolute lattice coordinates and overrides the fallback codeword, so the exported point is the lattice point selected in training.

Export checks that every position fits the selected index representation and that the stored coordinates satisfy

$$-2^{15} \leq (\mathbf{v}_g)_j \leq 2^{15} - 1, \qquad j = 1, \ldots, 8.$$

Conditional root packing. A smaller record is available when an escaped point differs from a selected core fallback by one E_8 root. Define

$$\mathcal{R} = \left\{ \mathbf{r} \in E_8 : \|\mathbf{r}\|_2^2 = 2 \right\}, \qquad |\mathcal{R}| = 240. \tag{13}$$

When the residual belongs to $\mathcal{R}$, decoding becomes

$$\mathbf{q}_g^\star = \mathbf{c}_{i_g} + \mathbf{r}_{j_g}, \qquad \mathbf{r}_{j_g} \in \mathcal{R}. \tag{14}$$

An 8-bit root identifier and a 24-bit position within a chunk form a 4-byte record, excluding chunk metadata. All 26,435 escaped groups in the 375-step artifact satisfy this condition. Payloads reported in the paper use the 20-byte coordinate record, including the 750-step endpoint.

Fused decoding. For the corresponding group of input-side transformed activations $\mathbf{h}_g$, linearity gives

$$\left\langle \mathbf{h}_g, \mathbf{q}_g^\star \right\rangle = \left\langle \mathbf{h}_g, \mathbf{c}_{i_g} \right\rangle + \left\langle \mathbf{h}_g, \mathbf{r}_{j_g} \right\rangle. \tag{15}$$

The fused matrix-vector path adds the sparse correction to the core-codeword contribution before the output-side Hadamard transform. The decoding scale c_s applies to both lattice terms, and the offset c_z is applied once. The corrected weight matrix is never materialized.

Encoding checks. Table 3 records two checks. Core decoding agrees on all 65,536 codewords, and every escaped group in the 375-step artifact lies one root from its fallback. The second count is consistent with $\rho = 6.540 \times 10^{-6}$ over the 4.04×10^9 quantized groups of the model.

C Additional Experimental Results

C.1 Training and Evaluation Protocol

Table 4 records the Qwen3.6-35B-A3B settings. The quality comparison uses the 750-step endpoints. The paired timing runs use an independently annealed 375-step schedule. The Llama-3.1-8B second stage uses that same 375-step shape.

Table 3: Encoding checks. Root packing holds for every escaped group in the 375-step artifact.

Check	Coverage	Result
Core decoding agreement	All 65,536 core codewords	Identical lattice coordinates across the tested decoding paths
Single-root membership	All 26,435 escaped groups in the 375-step artifact	Every residual belongs to $\mathcal{R}$

Table 4: Recorded settings for the Qwen3.6-35B-A3B experiments.

Setting	Value
Model	Qwen3.6-35B-A3B
Total parameters	35,107,181,936
Quantized parameters	32,338,083,840 expert FFN weights (92.1%)
Unquantized weights	2,769,098,096 in BF16: attention projections, router, embeddings, norms
Rate, quantized weights	2.03 bits, including scales and codebooks
Latent weights	BF16
Projection candidates and distances	FP32
PTQ preprocessing	LDLQ with block-Hadamard transforms of size 128
Finite core	2^{16} codewords; eight coordinates per group
Training corpus	2,000 long sequences from UltraData (OpenBMB, 2026)
Tokens per corpus pass	Approximately 50.8 million
Maximum sequence length	61,959 tokens
Configured context window	65,536 tokens
Main quality schedule	750 steps over six epochs
Peak learning rate	10^{-4}, warmup–stable–decay to zero
Paired timing configuration	375 steps, eight GPUs

Comparison controls. E8-QAT and our reproduction of LC-QAT from Wang et al. (2026a) share the base model, the quantized tensors, the training corpus, and the step budget. Timing measurements cover the QAT stage.

Baselines and stages. E8-PTQ is the lattice checkpoint that initializes E8-QAT, and UniSVQ is the scalar checkpoint that initializes LC-QAT. The matched-bit AQLM baseline uses one learned 2^{16}-entry codebook shared by all feed-forward matrices of a layer. QuIP# and QTIP are included as additional vector PTQ baselines. The per-matrix AQLM configuration, at 7.36 bits, is reported as a higher-bit reference.

Downstream evaluation. We evaluate function calling with BFCL (Patil et al., 2025), scientific question answering with GPQA (Rein et al., 2024), code generation with LiveCodeBench (Jain et al., 2025), and instruction following with IFEval (Zhou et al., 2023). The reported metrics are BFCL v3 overall accuracy, GPQA accuracy, LiveCodeBench pass@1, and IFEval prompt-level strict accuracy. All models use the same harness and the same generation settings.

C.2 Stage-1 Reconstruction Quality

We evaluate E8-PTQ, UniSVQ, and layer-shared AQLM with the same calibration Hessians, collected once on the original model and reused for every method. Per-expert Hessians are shrunk toward the layer mean with a pseudo-count equal to the input width. For layer ℓ, with feed-forward matrices $\mathcal{M}_\ell$ and routed-token weights $w_{\ell m}$,

$$D_\ell = \sum_{m \in \mathcal{M}_\ell} w_{\ell m} \frac{\operatorname{tr}\big(\Delta \mathbf{W}_{\ell m} H_{\ell m} \Delta \mathbf{W}_{\ell m}^\top\big)}{\operatorname{tr}\big(\mathbf{W}_{\ell m} H_{\ell m} \mathbf{W}_{\ell m}^\top\big)}. \tag{16}$$

Shared-expert matrices use unit weight. The reported mean averages D_ℓ over the 40 decoder layers.

Table 5: Stage-1 reconstruction on Qwen3.6-35B-A3B. Bold marks the lowest distortion and the largest best-layer count.

Method	Mean distortion ↓	Best layers ↑
UniSVQ	0.0605	0/40
AQLM, layer-shared	0.0598	1/40
E8-PTQ	**0.0482**	**39/40**

Table 6: Six-task `lm_eval` results on Qwen3.6-35B-A3B. Bold marks the best 2-bit score in each column.

Stage	Method	Bits	ARC-C	ARC-E	HellaSwag	PIQA	WinoGrande	BoolQ	Avg_6
Ref.	BF16	16	55.20	72.22	83.49	82.75	74.27	85.02	75.49
PTQ	UniSVQ	2.03	56.83	75.84	78.31	80.58	71.59	**89.33**	75.41
PTQ	AQLM, layer-shared	2.03	54.95	73.06	79.59	81.50	73.32	88.38	75.13
PTQ	E8-PTQ	2.03	56.74	78.41	79.38	80.47	72.61	88.78	76.06
QAT	LC-QAT	2.03	56.57	75.72	81.28	80.74	73.88	87.61	75.97
QAT	E8-QAT	2.03	**57.85**	**79.34**	**82.29**	**81.56**	**74.03**	86.64	**76.95**
Higher-bit reference									
PTQ	AQLM, per-matrix	7.36	56.91	74.87	82.18	81.77	75.14	87.61	76.42

Table 7: AIME25 pass@3. Bold marks the highest score, including ties.

Method	AIME25 pass@3 ↑
BF16	**90.00**
AQLM	83.33
LC-QAT	**90.00**
E8-QAT	**90.00**

E8-PTQ has the lowest distortion in 39 of the 40 layers. Its average, 0.0482, is 20.4% below UniSVQ and 19.4% below layer-shared AQLM.

C.3 Additional Quality Results

Complete task breakdown. Table 6 adds ARC-Easy and BoolQ to the four tasks in Table 1. ARC-Challenge, ARC-Easy, PIQA, and HellaSwag use normalized accuracy; WinoGrande and BoolQ use accuracy. The six-task average is

$$\text{Avg}_6 = \frac{1}{6}\sum_{j=1}^{6} a_j. \tag{17}$$

E8-QAT obtains the highest six-task average among the 2-bit configurations and leads on five of the six tasks. On BoolQ, UniSVQ scores 89.33.

Additional reasoning evaluation. On AIME25 pass@3, E8-QAT matches BF16 and LC-QAT at 90.00. AQLM scores 83.33.

C.4 Llama-3.1-8B Results

Table 8 adds ARC-Easy, BoolQ, and WikiText perplexity to Table 5.1. E8-QAT has the highest six-task average and the lowest perplexity among the quantized models: 13.77, against 16.73 for LC-QAT and 9.01 for BF16.

The last row uses the Qwen peak learning rate, 10^{-4}, on Llama. The Llama feed-forward scale w_s is $2.68\times$ larger, so this rate pushes 0.607% of groups outside the finite core and raises the index-and-

Table 8: Complete `lm_eval` results on Llama-3.1-8B. WikiText is perplexity (lower is better). Bold marks the best quantized result; the last row is excluded.

Stage	Method	ARC-C	ARC-E	HelS	PIQA	WinG	BoolQ	Avg_6	WikiText
Ref.	BF16	55.55	79.80	79.55	81.28	73.80	85.47	75.91	9.01
PTQ	UniSVQ	39.51	65.36	63.55	72.63	66.46	79.27	64.46	22.38
PTQ	E8-PTQ	43.94	**70.50**	66.16	74.76	68.90	81.47	67.62	20.79
QAT	LC-QAT	40.10	64.14	67.71	73.29	67.40	**82.57**	65.87	16.73
QAT	E8-QAT	**44.62**	69.49	**70.03**	**76.17**	**69.61**	77.43	**67.89**	**13.77**
QAT	E8-QAT, LR 10^{-4}	35.84	52.36	58.39	69.48	59.04	72.29	57.90	19.46

Table 9: Escape fractions under the two annealing schedules. A zero is an exact zero.

Training budget	Checkpoint step	Escape fraction ρ
375	100	0
375	200	1.196×10^{-6}
375	300	5.672×10^{-6}
375	375	6.540×10^{-6}
750	400	1.618×10^{-5}
750	500	3.618×10^{-5}
750	600	7.077×10^{-5}
750	700	8.535×10^{-5}
750	750	8.556×10^{-5}

Table 10: Endpoint storage with 20-byte coordinate records, in bits per quantized weight.

Endpoint	ρ	Δb	b_{payload}
375 steps, peak LR 10^{-4}	6.540×10^{-6}	1.308×10^{-4}	2.000131
375 steps, peak LR 5×10^{-5}	1.608×10^{-7}	3.216×10^{-6}	2.000003
750 steps	8.556×10^{-5}	1.711×10^{-3}	2.001711

extension payload to 2.12 bits per quantized weight. Scaling the rate by the inverse weight scale, to 3.73×10^{-5}, is the E8-QAT result in Table 5.1.

C.5 Escape Measurements and Learning-Rate Sensitivity

Escape across training budgets. Table 9 reports the fraction ρ of groups outside $\mathcal{C}_{\text{core}}$ for the 375-step and 750-step schedules. Escapes stay sparse, and growth slows near the endpoints: the final 50 steps of the 750-step run raise ρ by 0.25% of its own value.

Learning-rate sensitivity. Two 375-step runs share the schedule shape $11 + 252 + 112$ and differ only in the peak learning rate. Halving it from 10^{-4} to 5×10^{-5} lowers the endpoint escape fraction by $40.7\times$, from 6.540×10^{-6} to 1.608×10^{-7}. The reduced-rate run has zero escapes at steps 100 and 200.

Storage consequences. Each group keeps a 16-bit core index, and each escaped group adds a 20-byte coordinate record:

$$\Delta b = 20\rho, \qquad b_{\text{payload}} = 2 + \Delta b. \tag{18}$$

At the 750-step endpoint the sparse records add 1.711×10^{-3} bits per quantized weight, 0.086% of the 2-bit index payload. These figures cover indices and escape records.

D LIMITATIONS

Evaluation scope. Our experiments cover Qwen3.6-35B-A3B and Llama-3.1-8B, with long-form generation and agentic evaluations focused on the MoE model. Extending the evaluation to additional architectures, model scales, and application domains would further characterize the applicability of the method.

Weight-only quantization. This work quantizes weights. Activations and the KV cache remain in higher precision. Combining the lattice weights with activation and KV-cache quantization is a direct way to reduce memory further in long-context inference.

Deployment configurations. The inference results use the fused kernels and batch sizes reported in the main text. The memory reduction is what admits the larger batch. Further kernel work and other hardware platforms are natural next measurements.