VODKA: VECTOR QUANTIZATION-AWARE TRAINING VIA DIFFERENTIABLE KERNEL APPROXIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The low bits quantization of Large Language Models (LLMs) is essential for lowering deployment costs and enabling inference with limited resources. As vector quantization (VQ) offers higher expressivity than scalar quantization (SQ) and quantization-aware training (QAT) demonstrates a higher upper-bound than post-training quantization (PTQ), combining VQ with QAT shows better potential for low-bit quantization. However, in face of the optimization and efficiency problems, current VQ-QAT methods either constrain the expressivity, or require extensive search, which motivates us to explore the tradeoff between expressivity and efficiency. Therefore, we propose **Vodka**, a novel VQ-QAT algorithm via differentiable kernel approximation. We identify the key expressivity gap as the mapping function: current VQ-QAT restricts this map to be affine, whereas VQ allows it to be arbitrary. To bridge the gap, Vodka constructs the **Codebook Spectrum** with kernel by ordering the curvature, with two endpoint functions provably the affine and arbitrary, which stays search-free and admits an exact codebook gradient at fixed indices, serving as implicit codebook to approach the VQ controllably. In contrast to prior work with a fixed initialized codebook, we optimize the implicit codebook jointly with the weights, so that both adapt to each other. We also propose **Shakernel**, a fused GPU kernel that implements Vodka without heavy intermediate tensors, reducing training cost and memory. Empirical evaluations on Qwen3.6-35B-A3B and Llama3.1-8B on 7 benchmarks demonstrate that Vodka outperforms state-of-the-art (SOTA) low-bit methods while maintaining efficiency. Notably, Vodka improves over the strongest baseline by 1.31% on IFEval and 0.51% on BFCL, demonstrating the excellent performance. Our implementation is available via an anonymized link.[1]

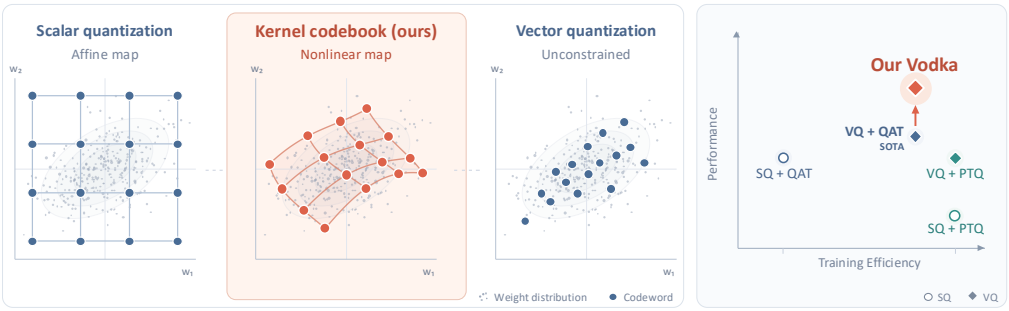
1 INTRODUCTION

Large language models (LLMs) have demonstrated excellent performance in a wide range of tasks (Touvron et al., 2023; Singh et al., 2025; DeepSeek-AI et al., 2026), but their large number of parameters impose substantial demands on memory capacity and bandwidth (Gholami et al., 2024; Pope et al., 2023). Quantizing weights to low bit widths, such as 2 bits per weight, reduces storage requirements, offering a promising approach to lower deployment costs and enabling inference on devices with limited resources (Gholami et al., 2022; Chee et al., 2023). However, reducing the bit restricts the expressive capacity of quantized weights, which can degrade the model performance. This tradeoff between efficiency and performance is a central challenge in low-bit quantization.

Scalar quantization (SQ), which represents each weight by a value from a finite set of scalar levels, can be applied through post-training quantization (PTQ) (Frantar et al., 2022; Lin et al., 2024) or QAT (Liu et al., 2024; Chen et al., 2025). However, even with nonuniform levels (Kim et al., 2023), SQ has limited flexibility in representing weight, which can impair accuracy at low bit widths (Van Baalen et al., 2024). In contrast, vector quantization (VQ) represents weight groups by a codeword (Gersho & Gray, 2012) whose weights are no longer confined to a uniform grid and the codebook can adapt to correlations within a group. Existing VQ methods exploit its capability in PTQ (Chee et al., 2023), and results motivate us to study the VQ-QAT to effectively improve model accuracy under low bit and improve training data efficiency.

[1] Implementation: `https://anonymous.4open.science/r/Vodka-29DB`.

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



(a) Motivation: Bridge the Gap between SQ and VQ (b) Motivation: Tradeoff

Figure 1: **Motivation** *(left)* Vodka aims to bridge the gap between SQ and VQ through a nonlinear map. *(right)* Conceptual illustration on expressivity and efficiency: Vodka aims to achieve better performance, while approaching PTQ-level efficiency.

However, applying QAT with VQ remains challenging: the mapping function in VQ is an argmin operation with no derivative. Existing work resolves this in two ways: **Constrained Mapping** restrict the codebook to a fixed and uniform lattice, making the forward search-free but restricting expressivity (Wang et al., 2026). **Alternating optimization** keeps expressivity, but updating codes by discrete search with extensive computes (Malinovskii et al., 2024). This raises a central question:

Can a codebook mapping in VQ be expressive enough, yet still admit efficient forward and end-to-end gradients?

Therefore, we propose **Vodka**, a novel **V**ect**o**r quantization-aware training via **d**ifferentiable **k**ernel **a**pproximation. To address the expressivity and efficiency limitations, we identify the mapping function as the source of the expressivity gap. We therefore place the affine map of current VQ-QAT and the arbitrary map of VQ at the two ends of one function space, so it is natural that a smooth space in between would trade expressivity against efficiency. Such spaces are actually infinite, as those containing the affine end form a manifold of positive dimension, so we need a criterion. We define curvature, the squared second differences over the index grid as the criterion, since its null space is exactly the affine design and no other order of difference has this property. Ordering the remaining directions by curvature with kernel functions gives the **Codebook Spectrum**, whose two ends are provably exact, and whose levels cost the same bits, as shown in Fig. 1.

Additionally, we jointly optimize the parameters of our proposed implicit codebook and model weights, which allows the codebook and the weight distribution to adapt to each other throughout training. We further observe that the index gradient needs a $d \times d$ Jacobian per group, four times the size of the weights, yet only K of them are distinct, since a group enters the backward pass only through its index. For efficiency, we therefore implement **Shakernel**, a fused GPU kernel that computes the K blocks and the codebook on chip instead of storing them per group.

To show the effectiveness of Vodka, we evaluate on 7 benchmarks with Qwen3.6-35B-A3B (Yang et al., 2025) and Llama3.1-8B (Grattafiori et al., 2024). Vodka outperforms state-of-the-art QAT baselines while retaining training and inference cost. Specifically, over the best baseline, Vodka improves by 1.31% on IFEval and 0.51% on BFCL at the same 2 bits on Qwen, and by 1.90% on BFCL on Llama. Shakernel removes 70% of the step time and 64% of the peak memory of an unfused implementation, holding training within $1.21\times$ of affine VQ-QAT. To this end, these results demonstrate that Vodka is promising to low-bit quantization for future LLM deployment.

2 Preliminary and Challenge

2.1 Quantization-aware Training

Weight quantization replaces a full-precision weight matrix $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ by a low-bit representation. A quantizer Q maps W to a dequantized $\hat{W} = Q(W)$ whose entries come from a finite set of representable values, keeping at inference only the integer index of each, together with a full-

precision index-to-value map D, parameterized by θ, that returns the value of a given index. A layer stores $b\, d_{\text{in}} d_{\text{out}}$ index bits, the bit width b being the average per weight. Each entry of θ is shared by many weights, so its amortized cost is small, but at low bit width it is no longer negligible relative to b (Nagel et al., 2021).

Post-training quantization (PTQ) constructs the quantizer from a small calibration set and leaves the pretrained weights otherwise untouched (Frantar et al., 2022; Lin et al., 2024). Quantization-aware training (QAT) instead keeps W trainable and places $\hat{W}$ in the training loss L, so the weights adapt to the quantization grid and compensate each other's error (Jacob et al., 2018; Liu et al., 2024; Chen et al., 2025), which yields higher accuracy at low bit width at the cost of orders of magnitude more data and compute than the few calibration sequences PTQ requires. For an input $X \in \mathbb{R}^{n \times d_{\text{in}}}$ of n tokens, a linear layer computes $Y = X\hat{W}$, and back-propagation gives

$$\frac{\partial L}{\partial \hat{W}} \;=\; X^{\top} \frac{\partial L}{\partial Y}, \qquad \frac{\partial L}{\partial W} \overset{\text{STE}}{\approx} \frac{\partial L}{\partial \hat{W}}, \tag{1}$$

where the first equality is exact and the second is the straight-through estimator (STE) (Bengio et al., 2013), which treats Q as the identity in the backward pass, since Q is piecewise constant and its derivative is zero wherever it is defined. The parameters θ are often fixed in advance, either from calibration statistics or by a preceding PTQ stage, but they can also be trained alongside W. Here W is a latent weight that only accumulates gradients (Helwegen et al., 2019). It is discarded after training, and inference keeps the indices together with θ.

2.2 Vector Quantization and Current Limitations

SQ maps each weight to a level of a grid shared by its group, uniform SQ to a regular grid fixed by a step size and a zero point and nonuniform SQ to a learned set of levels, so either way a weight is confined to the 2^b levels its group shares. VQ instead stores one index for every d neighboring weights and quantizes the group jointly: given a codebook of K codewords $c_1, \ldots, c_K \in \mathbb{R}^d$, the g-th group $w_g \in \mathbb{R}^d$ of W is replaced by its nearest codeword:

$$\hat{w}_g = c_{k_g^\star}, \qquad k_g^\star = \arg\min_k \|w_g - c_k\|_2^2. \tag{2}$$

Integrating Eq. 2 into QAT presents two challenges. First, hard assignments do not support direct gradient-based optimization. For hard nearest-neighbor quantization, back-propagation yields

$$\frac{\partial L}{\partial c_k} = \sum_{g:\, k_g^\star = k} \frac{\partial L}{\partial \hat{w}_g}, \qquad \frac{\partial \hat{w}_g}{\partial w_g} = \mathbf{0} \quad \text{a.e.} \tag{3}$$

Each codeword therefore receives gradients only from its currently assigned groups, while the map is piecewise constant in the input and non-differentiable at assignment boundaries, so ordinary back-propagation through the hard assignment cannot directly guide the optimization. Second, recomputing assignments is costly: an exhaustive nearest-neighbor search costs $O(Kd)$ per group, where $K = 2^{bd}$ reaches $65{,}536$ at $b = 2$ and the common group size $d = 8$, and the weights change during QAT so the assignments require repeated searches.

2.3 Challenge in VQ-QAT

Two representative approaches address these challenges differently. LC-QAT avoids explicit codebook search in the training forward pass through a learned affine mapping of discrete vectors, but couples all codewords through that shared affine structure (Wang et al., 2026), whereas PV-Tuning alternates continuous-parameter optimization with gradient-informed discrete reassignment, but incurs repeated discrete-search overhead (Malinovskii et al., 2024). Both integrate VQ into QAT at the expense of either codebook expressivity or optimization efficiency, whereas our implicit codebook provides explicit control over this trade-off.

Codebooks computed rather than stored are not new, but have been developed within PTQ and motivated by the inference routine (Tseng et al., 2024a;b), and those conclusions do not transfer: in QAT the codebook is itself an optimization variable that must receive the gradient in Eq. 3, and assignments drift as the latent weights move, so the map is exercised at every step rather than only at deployment. We therefore revisit the parameterization question in the QAT regime, where the index-to-value map must be trainable and cheap to reassign at the same time.

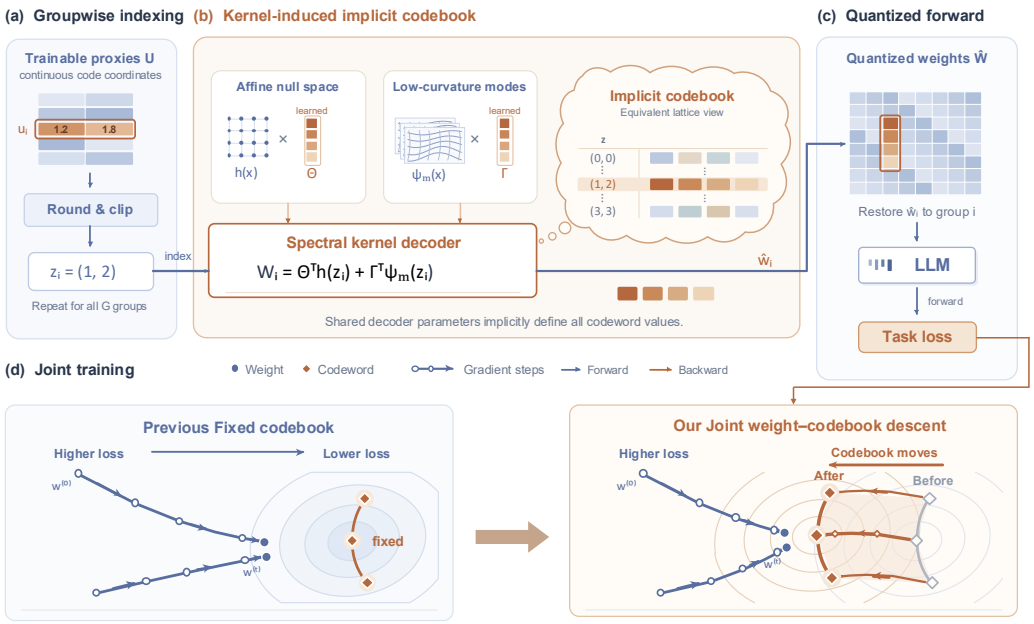


Figure 2: **Overview of Vodka.** *(a) Vector Indexing.* Each group of d weights is rounded coordinate-wise to an integer index. *(b) Implicit codebook.* The kernel mapping dequantizes the index into the values for matmul. Only the shared coefficients Θ, Γ are stored, and m moves from affine to arbitrary function. *(c) Quantized forward.* Values are gathered back into $\hat{W}$, which the LLM uses to compute the task loss. *(d) Joint optimization.* A fixed codebook leaves the codewords as frozen targets (left), whereas Vodka optimizes them with the weights (right).

3 Method

Our motivation is the limitation raised above: in QAT the index-to-value map D must be trainable and cheap to reassign at the same time, while existing designs achieve one at the expense of the other. Sec. 3.1 therefore models the affine map of current VQ-QAT and the unconstrained codebook of VQ in one space and derives a family of codebooks; Sec. 3.2 places it in a pipeline whose forward pass needs no search; Sec. 3.3 presents our novel joint optimization of the codebook and the weights; and Sec. 3.4 shows that the parameterization lets one kernel carry the forward and the backward.

3.1 A Functional View of Vector Quantization

Regarding the map D as a function, we first model the two designs it takes in practice, the affine map to which current VQ-QAT restricts the codebook (Esser et al., 2019; Wang et al., 2026) and the codebook of VQ, which returns a stored vector:

$$D_{\text{aff}}(z) = Az + t, \qquad D_{\text{VQ}}(k) = c_k, \tag{4}$$

where the index z of a group runs over the integer grid $\mathcal{Z} = \{0, \dots, q-1\}^d$ of $q = 2^b$ levels per coordinate, so $K = q^d$, and the $d(d+1)$ entries of $A \in \mathbb{R}^{d\times d}$ and $t \in \mathbb{R}^d$ make D_{aff} affine, of which uniform SQ is the case $A = sI$, whereas the Kd free entries of the codebook make D_{VQ} arbitrary. One reads a grid point z carrying the coordinate structure of $\mathcal{Z}$ and the other a label k carrying none, so the two have no common parameterization. We therefore rewrite D_{VQ} onto the domain of D_{aff}, which their equal size permits, defining $\iota : \mathcal{Z} \to \{1, \dots, K\}$ as the bijection reading a grid point as the base-q digits of a label and storing each codeword at the grid point carrying its own,

$$\widetilde{D}_{\text{aff}}(z) = Az + t, \qquad \widetilde{D}_{\text{VQ}}(z) = C_z := c_{\iota(z)}, \tag{5}$$

where the matrix $C \in \mathbb{R}^{K\times d}$ collects the K codewords, so that a map on $\mathcal{Z}$ and the matrix of its values are identified by $D(z) = C_z^\top$. Relabeling leaves the quantizer itself untouched, so the two ends of Eq. 5 are the two designs of Eq. 4 and not approximations of them.

To build a space between these two ends, we consider the dimension. Both are subspaces of one $\mathbb{R}^{K\times d}$: VQ occupies all Kd dimensions, whereas the affine end writes every value as a combination of the same $d+1$ terms with only the $d(d+1)$ entries of A and t free, so it occupies $\mathrm{col}(F)$, where F has row $[1,\ z^\top]$ at the grid point z. A design in between generates C from a subspace containing $\mathrm{col}(F)$, and the candidates of a given dimension form a Grassmann manifold of positive dimension (Absil et al., 2008), so infinitely many exist and selecting one needs a criterion.

Our criterion ranks the directions of $\mathbb{R}^{K\times d}$ by how far each departs from the affine end. An affine map has constant slope, since a unit step along coordinate j adds the j-th column of A at every z, so such a departure is a change of slope, measured by second differences:

$$\mathcal{J}(C) = \sum_{j,k,z} \big\|\Delta_{jk} D(z)\big\|_2^2 = \mathrm{tr}\big(C^\top \Omega C\big), \tag{6}$$

where e_j is the j-th coordinate step, $\Delta_{jk}D(z) = D(z+e_j+e_k) - D(z+e_j) - D(z+e_k) + D(z)$ is evaluated wherever its arguments lie in $\mathcal{Z}$, and $\Omega \succeq 0$ depends only on q and d. Neither the order of the difference nor the mixed terms $j \neq k$ is free to choose. First differences vanish only on constants and higher ones vanish on quadratics too, while dropping the mixed terms would put products such as $z_1 z_2$ at zero, and a direction at zero is never ranked (Appendix B.3).

Theorem 1 (Curvature splits the space)**.** *Let $q \geq 2$ and $d \geq 1$. Since Ω acts on each output column of C separately, both parts are statements in $\mathbb{R}^K$.* (a) *$\mathcal{J}(C) = 0$ if and only if D is affine, that is,* $\ker\Omega = \mathrm{col}(F)$. (b) *Consequently* $\mathrm{rank}\,\Omega = K - d - 1 =: M$*, and the eigenvectors $v_1, \ldots, v_M$ of the positive eigenvalues $0 < \rho_1 \leq \cdots \leq \rho_M$ form an orthonormal basis of* $\mathrm{col}(F)^\perp$.

Applied to every output column, the theorem splits $\mathbb{R}^{K\times d}$ into the two parts the space needs: by (a) the part at zero curvature is the affine end itself, and by (b) the rest is spanned by M orthogonal directions of increasing curvature, so the m least curved are a well-defined choice that adds to rather than duplicates it, and taking all M reaches the other end.

Definition 1 (Curvature filtration)**.** For $m \leq M$ write $V_m = [v_1, \ldots, v_m]$ and

$$\mathcal{C}_m = \big\{F\Theta + V_m\Gamma\big\}, \qquad \dim \mathcal{C}_m = d(d+1+m), \tag{7}$$

where $\Theta \in \mathbb{R}^{(d+1)\times d}$ and $\Gamma \in \mathbb{R}^{m\times d}$ are the coefficients of the affine and of the admitted directions, and the spaces are strictly nested, $\mathcal{C}_0 \subsetneq \cdots \subsetneq \mathcal{C}_M$.

Eq. 7 is the space constructed above, and m is the single parameter that moves an implicit codebook across it. At $m = 0$ it is the affine design and at $m = M$ all of $\mathbb{R}^{K\times d}$, the unconstrained codebook of VQ, both *exactly* rather than approximately, while an intermediate m keeps the affine design and spends m further coefficients per output column on the smoothest corrections to it. The admissible m are those closing a complete eigenspace, since a level cutting inside a repeated eigenvalue would depend on the basis an eigensolver returns; these depend only on q and d, and form the capacity axis used throughout. The construction is a kernel expansion, as Eq. 6 is the discrete second-order Beppo-Levi seminorm and $\Omega^\dagger$ therefore a kernel modulo affine maps with leading eigenfunctions $v_1, \ldots, v_m$, so truncating at m orders the corrections by complexity rather than degree (Appendix B.6).

Raising m multiplies the decoder coefficients by up to $K/(d+1)$, raising the question of whether the capacity is paid for in bits. It is not: the coefficients are side information shared by every group the decoder serves, whereas the bit width is set by the index alone.

Proposition 1 (Rate invariance)**.** *Let one decoder serve G groups, each coefficient stored in β_c bits. A layer quantized at any level m costs*

$$\underbrace{\log_2(K)/d}_{=b} + (d+1+m)\,\beta_c/G \qquad \textit{bits per weight,} \tag{8}$$

whose first term is the nominal bit width, independent of m.

Eq. 8 separates capacity from rate: m enters only the second term, which at $m = 27$ is 2.5×10^{-5} bits per weight averaged over a layer, below the last digit of the nominal 2.000 (Appendix B.4). What it does change is the distortion: on a calibrated Gaussian source under nearest-neighbor encoding, the affine end loses 0.096 bits to uneven occupancy of the cells and a further 0.46 to the affine constraint, of which traversing the family closes 51.6% at $m = 27$, a figure Eq. 10 meets at the affine end and bounds beyond it (Appendix B.5).

3.2 Search-Free Vector Quantization with an Implicit Codebook

We now place the family above inside a quantizer, which has to reproduce the full-precision $Y = XW$ from a low-bit representation. Vodka quantizes W conjugated by a fixed pair of transforms, which spread its distribution before rounding, and the layer restores the product,

$$Y = \big(X \operatorname{diag}(\sigma_{\text{in}})\, H_{\text{in}}\big)\, \hat{W}\, H_{\text{out}} \operatorname{diag}(\sigma_{\text{out}}), \tag{9}$$

where $H_{\text{in}}, H_{\text{out}}$ are block Hadamard matrices with random signs, $\sigma_{\text{in}}, \sigma_{\text{out}}$ trainable diagonal scales, and $\hat{W}$ the quantization of $H_{\text{in}}^{\top} \operatorname{diag}(\sigma_{\text{in}})^{-1} W \operatorname{diag}(\sigma_{\text{out}})^{-1} H_{\text{out}}^{\top}$, exactly invertible so that the only approximation is the rounding below. As shown in Fig. 2, $\hat{W}$ follows in two steps: a group is rounded coordinate-wise to an index z_g, which an implicit codebook drawn from $\mathcal{C}_m$ then decodes. For the first, groups are formed from d consecutive weights along the input dimension, and the latent weight $u_g \in \mathbb{R}^d$ of group g is rounded and clipped to

$$z_g = \operatorname{clip}\big(\operatorname{round}(u_g), 0, q-1\big) \in \mathcal{Z}. \tag{10}$$

Being coordinate-wise, this needs no nearest-neighbor search over K codewords. For the second, the parameters θ of Sec. 2.1 become the coefficient pair (Θ, Γ) of Eq. 7, and z_g selects the rows of F and V_m that decode it,

$$\hat{w}_g^{\top} = [\,1,\ z_g^{\top}\,]\,\Theta + (V_m)_{z_g}\Gamma, \qquad C = F\Theta + V_m\Gamma \in \mathbb{R}^{K\times d}, \tag{11}$$

where $(V_m)_{z_g}$ is row $\iota(z_g)$ of V_m, and the $\hat{W}$ enters Eq. 9 so the matmul proceeds unchanged.

3.3 Joint Optimization of the Codebook and the Weights

Existing VQ-QAT holds the codebook fixed and trains the weights alone (Wang et al., 2026), leaving the codewords as targets the weights must approach, whereas Eq. 11 makes both the indices and the codebook trainable, so Eq. 12 optimizes them against the task loss,

$$\min_{\Theta,\Gamma,\{u_g\},\sigma} L\big(\hat{W}(\Theta,\Gamma,\{u_g\});\sigma\big), \tag{12}$$

where σ collects the diagonal scales of Eq. 9 with the normalization weights of the surrounding blocks, shown in Fig. 2(d). The implicit codebook is differentiable but the rounding of Eq. 10 is not, so the two admit different gradients. The coefficients (Θ, Γ) act after the rounding and admit a closed-form derivative, since at fixed indices Eq. 11 is linear in them and the gather selects rows, so with $\eta_g = \partial L/\partial \hat{w}_g$ pooled over the groups sharing a codeword,

$$[G_C]_{k,:} = \sum_{g:\,\iota(z_g)=k} \eta_g^{\top}, \qquad \nabla_{\Theta} L = F^{\top} G_C, \qquad \nabla_{\Gamma} L = V_m^{\top} G_C, \tag{13}$$

which is the exact derivative at fixed indices (Remark 5) and, since every coefficient contributes to every row of C, moves the whole codebook in $d(d+1+m)$ coordinates rather than the Kd of a stored table (Remark 6). The latent weights u_g act before the rounding, for which we adopt the differentiable gradient estimator (DGE) of Wang et al. (2025),

$$\widetilde{\nabla}_{u_g} L = S(u_g)\, J(z_g)^{\top} \eta_g, \tag{14}$$

in which the diagonal $S(u_g)$ is the DGE surrogate for the derivative of the rounding and $J(z_g) \in \mathbb{R}^{d\times d}$ is the Jacobian of the index-to-value map. The estimation is confined to S (Remark 7): since each coordinate of an index assumes q values, every column of C is the value table of a unique polynomial of degree at most $q-1$ per coordinate, so J is its analytic derivative rather than a finite difference. Both paths run throughout training and we do not decompose the gains between them, as Appendix D records and Appendix B.8 delimits.

3.4 A Fused Kernel for the Implicit Codebook

Our parameterization also determines how the quantization executes, since the forward pass of Sec. 3.2 decodes a group from row $\iota(z_g)$ of the table C of Eq. 11, of Kd entries, and Eq. 14 contracts η_g against the Jacobian at the same row, whose K blocks form a second table of Kd^2. Both

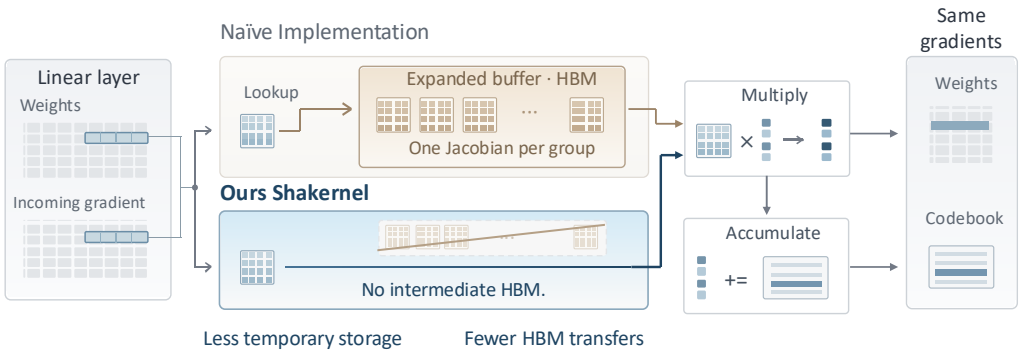


Figure 3: **Shakernel.** The naive implementation stores the index of every group and the Jacobian. Since the codebook is computed from its coefficients rather than stored, it and the Jacobian can stay resident on chip, reducing the step time and the peak memory.

are addressed by the index and sized by q and d alone, so neither grows with the layer: at $d = 4$ they occupy tens of kilobytes and stay on chip for a whole projection slot. As shown in Fig. 3, Shakernel builds both from the $d(d + 1 + m)$ coefficients at launch and traverses the groups once, accumulating the label in one register, decoding $\hat{w}_g$, and on the backward pass accumulating η_g into Eq. 13 and contracting it against the same row of J, so neither the codebook nor the per-group Jacobian is stored over the layer and keeping J resident leaves the index path available for free.

4 Experiments

4.1 Setting

Datasets We collect a deliberately small corpus of 2,000 sequences from public datasets: DeepMath-103K (He et al., 2026b), rStar-Coder (Liu et al., 2026), UltraData-SFT-2605 (OpenBMB, 2026), UltraData-SFT-Agent-2609 (Team, 2026), spanning math, code, function calling, long-context trajectories, and instruction following.

Training We quantize Qwen3.6-35B-A3B (Yang et al., 2025) and Llama3.1-8B (Grattafiori et al., 2024), which differ in architecture and scale, to 2 bits per weight, the width at which a substantial gap to full precision remains. Every QAT method starts from the same initialization, as is standard for VQ-QAT (Wang et al., 2026), and the data is kept at a maximum context length of 64K, which admits the long agent trajectories while bounding the cost of a single forward pass. All experiments are trained at 1×10^{-4} under a warmup schedule, with an effective batch of 16 on $8\times$ GPUs, and every reported number is a mean over random seeds.

Evaluation We evaluate Vodka on basic and on complex agentic ability. For the basic we use lm-eval (Gao et al., 2024) on ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020) and WinoGrande (Sakaguchi et al., 2021), zero-shot, reporting acc_norm with their unweighted mean as Avg. For the complex we consider three harder benchmarks: LiveCodeBench (Jain et al., 2025) for code, BFCL v3 (Yan et al., 2024) for tool use, and IFEval (Zhou et al., 2023) for instruction following.

4.2 Main Results

We compare Vodka against state-of-the-art (SOTA) quantization methods at 2 bits: RTN, GPTQ (Frantar et al., 2022), QuIP# (Tseng et al., 2024a), QTIP (Tseng et al., 2024b), PV-Tuning (Malinovskii et al., 2024) and LC-QAT (Wang et al., 2026), with FP16 as the reference. Each QAT method runs for 375 steps, the point at which our runs have converged, under the same hyper-parameters and from the same PTQ initialization, and the post-training methods receive the data as their calibration set. Vodka runs at $m = 27$, the smallest level that captures the available

Table 1: **Comprehensive evaluation at 2 bits.** Blue refers to the best performance. Avg. is unweighted mean of four basic benchmarks; Code, Agent and IF are generated free form. Every entry is a mean over random seeds.

		Base (lm-eval)					Code	Agent	IF
Type	**Method**	ARC-C	HeSw	PIQA	WinG	**Avg.**	LiveCodeBench	BFCL	IFEval
Qwen3.6-35B-A3B									
N/A	FP16	62.37	82.43	83.03	75.14	75.74	81.42	75.99	85.40
SQ	RTN	49.23	79.44	79.87	69.85	69.60	20.38	53.91	77.82
	GPTQ	52.78	79.17	80.20	69.76	70.48	20.38	55.87	78.18
VQ	QuIP#	56.66	77.25	80.20	72.77	71.72	47.77	67.31	81.70
	QTIP	51.71	79.36	81.12	72.69	71.22	26.45	61.77	80.78
VQ-QAT	PV-Tuning	57.23	78.93	80.67	73.65	72.62	75.24	72.07	81.87
	LC-QAT	58.11	79.25	81.45	74.90	73.43	74.98	71.66	81.15
	Vodka(m=27)	57.85	79.78	81.50	74.90	73.51	75.55	72.58	83.18
Llama3.1-8B									
N/A	FP16	55.89	79.48	81.34	73.48	72.55	15.92	30.21	74.31
SQ	RTN	25.34	26.54	51.90	51.70	38.87	0.00	0.00	4.99
	GPTQ	24.40	26.38	50.98	50.04	37.95	0.00	0.00	5.04
VQ	QuIP#	36.95	51.19	68.72	59.43	54.07	2.27	3.08	20.27
VQ-QAT	LC-QAT	39.17	60.67	72.37	62.37	58.65	4.29	6.19	28.87
	Vodka(m=27)	40.27	61.28	73.18	60.03	58.69	5.03	8.09	29.11

gain, for the reasons of Appendix E. As shown in Tab. 1, Vodka is ahead of every baseline on all three complex tasks, by 1.31% on IFEval, 0.51% on BFCL and 0.31% on LiveCodeBench over the strongest baseline of each. On the four basic benchmarks the best methods are not separated, differing by 0.08% of Avg here and 0.04% on Llama, within the seed-to-seed variation of these suites, whereas LC-QAT trails PV-Tuning on all three complex tasks. The conclusion holds on Llama3.1-8B, which differs in architecture and scale.

4.3 Efficiency Analysis

We conduct efficiency experiments on Qwen3.6-35B-A3B for training and inference. Vodka decodes at inference by one gather from a $K \times d$ table exactly as affine VQ-QAT does, so it keeps their speed and 2 bits per weight, and for QAT we compare step time and peak memory, normalized to LC-QAT. As shown in Fig. 4, our method without Shakernel costs $4.07\times$ the training step time and $3.67\times$ the peak memory of LC-QAT. Shakernel removes 70.3% and 63.8% of that overhead, leaving $1.21\times$ and $1.33\times$ against the baseline. The residual does not grow with m, since the coefficients enter only construct once.

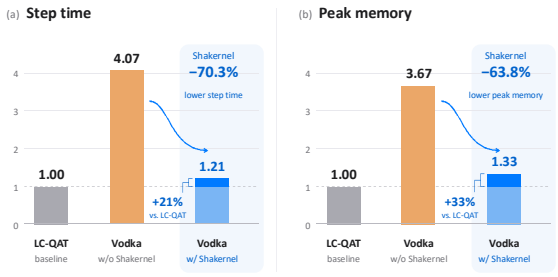


Figure 4: **Efficiency.**

4.4 Convergence Speed

To show the effectiveness of our joint optimization, we evaluate along the whole training trajectory. As shown in Fig. 5, the ordering is fixed early and the curves never cross: Vodka reaches the accuracy LC-QAT attains at 375 steps after roughly 15% of them on IFEval and 40% on BFCL and

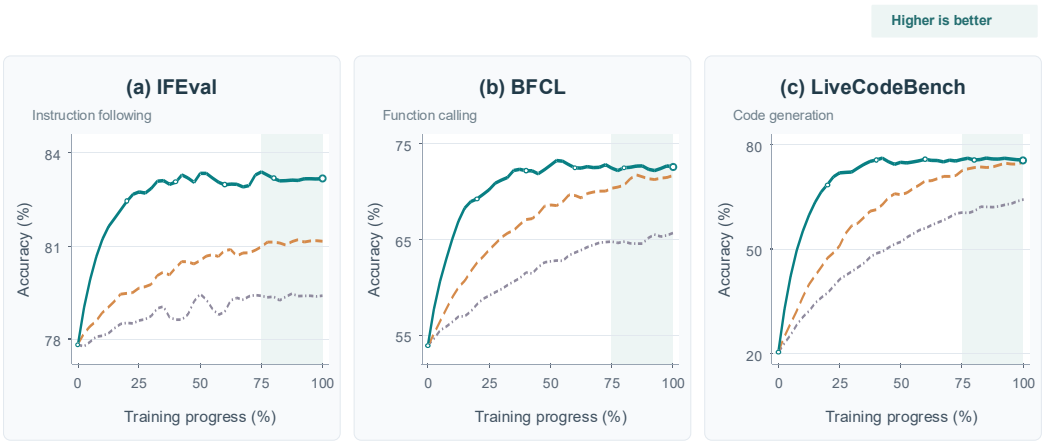


Figure 5: Convergence curve on three benchmarks.

LiveCodeBench, and is flat over the last quarter while LC-QAT is still rising at 100% on all three. With the initialization, the data, and the schedule shared, that difference is the codebook moving toward the weights rather than standing as a fixed target, which Eq. 13 achieves in $d(d+1+m)$ shared coordinates. The budget is therefore conservative rather than favorable, since a longer schedule would narrow the margins of Tab. 1 and not widen them.

4.5 Traversing the Codebook Spectrum

To examine the expressivity at different m, we train on Qwen3.6-35B-A3B with all other settings fixed, at the admissible levels $m \in \{0, 10, 27, 95, 251\}$, whose two ends are the affine and the free one. Since $[F, V_M]$ is a basis of $\mathbb{R}^K$ by Theorem 1(b), the free end is a trainable table reparameterized rather than approximated, while $m = 0$ coincides with LC-QAT, whose map is fixed at initialization. As shown in Fig. 6, accuracy rises monotonically and the gains concentrate at the smooth end: $m = 27$ uses 12.5% of the coefficients of a full table and forgoes at most 0.44% across the three tasks. Every level from $m = 27$ upward is ahead of all baselines on all three, so the conclusion does not rest on which one we report.

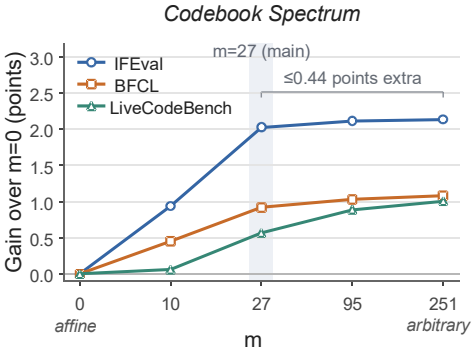


Figure 6: Traversing the Codebook Spectrum.

5 Conclusion

We presented Vodka, a VQ-QAT method built on a functional view of quantization. Treating the index-to-value map as a function places the affine design that current VQ-QAT is restricted to and the unconstrained codebook of vector quantization at the two ends of a single space, and ordering that space by discrete second difference yields the Codebook Spectrum, a family whose ends are exactly those two designs. A codebook in the family is calculated from shared coefficients that receive an exact gradient at fixed indices, so it and the weights optimize the task loss together to improve convergence. Furthermore, since neither the codebook nor its Jacobian has to be stored over the layer, Shakernel calculates both on chip and serves the forward pass and both gradients in one traversal, keeping the step time within $1.21\times$. At 2 bits per weight, Vodka attains the best accuracy among the quantized methods on almost all benchmarks, while a gap to full precision remains and is worth further study, and we hope this is useful to future work on low-bit quantization.

AI USE STATEMENT

In this work, we used generative AI tools for polishing the wording of the manuscript and for reviewing parts of our training and evaluation code. We have not used generative AI tools for research ideation, for deriving or proving the theoretical results, or for designing and running the experiments, and the remaining tasks subject to required disclosure are not applicable to this work. Additionally, we used generative AI tools to proofread the manuscript for broken cross-references, unresolved citations and formatting errors. We have reviewed all AI-assisted work. Every AI-suggested edit to the text was checked against our intended meaning by the authors, and every AI comment on the code was verified by rerunning the affected training and evaluation scripts. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2008.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

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.

Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems*, 36:4396–4429, 2023.

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, 2025.

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.

Anyi Xu DeepSeek-AI, Bangcai Lin, Bing Xue, Bingxuan Wang, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *https://huggingface. co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4. pdf*, 2026.

Jean Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. In *Constructive theory of functions of several variables: proceedings of a conference held at oberwolfach April 25–May 1, 1976*, pp. 85–100. Springer, 2006.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

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, 07 2024. URL `https://zenodo.org/records/12608602`.

Allen Gersho and Robert M Gray. *Vector quantization and signal compression*. Springer Science & Business Media, 2012.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*, pp. 291–326. Chapman and Hall/CRC, 2022.

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *IEEE Micro*, 44(3):33–39, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Liulu He, XuanAng Liu, Juntao Liu, Taolue Feng, Ting Lu, Chunsheng Gan, Zhiyv Peng, Yuan Du, Huanrui Yang, Yijiang Liu, et al. LiftQuant: Continuous Bit-Width LLM via Dimensional Lifting and Projection. *arXiv preprint arXiv:2606.04050*, 2026a.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 138306–138322, 2026b.

Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. *Advances in neural information processing systems*, 32, 2019.

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 *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 2704–2713. IEEE, 2018.

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 *International Conference on Learning Representations*, 2025.

Ian W Kennedy and Nafise Sadat Moosavi. Initialisation determines the basin: efficient codebook optimisation for extreme LLM quantization. *arXiv preprint arXiv:2604.08118*, 2026.

Sehoon Kim, Coleman Hooper, Amir Gholami, Zhen Dong, Xiuyu Li, Sheng Shen, Michael W Mahoney, and Kurt Keutzer. Squeezellm: Dense-and-sparse quantization. *arXiv preprint arXiv:2306.07629*, 2023.

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. *Proceedings of machine learning and systems*, 6:87–100, 2024.

Yifei Liu, Li Lyna Zhang, Yi Zhu, Bingcheng Dong, Xudong Zhou, Ning Shang, Fan Yang, Cheng Li, and Mao Yang. rstar-coder: Scaling competitive code reasoning with a large-scale verified dataset. *Advances in Neural Information Processing Systems*, 38:58780–58807, 2026.

Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. In *Findings of the association for computational linguistics: ACL 2024*, pp. 467–484, 2024.

Vladimir Malinovskii, Denis Mazur, Ivan Ilin, Denis Kuznedelev, Konstantin Burlachenko, Kai Yi, Dan Alistarh, and Peter Richtarik. Pv-tuning: Beyond straight-through estimation for extreme llm compression. *Advances in Neural Information Processing Systems*, 37:5074–5121, 2024.

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

OpenBMB. UltraData-SFT-2605, 2026. URL https://huggingface.co/datasets/openbmb/UltraData-SFT-2605.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. *Proceedings of machine learning and systems*, 5:606–624, 2023.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

MiniCPM Team. UltraData-SFT-Agent-2609. https://huggingface.co/datasets/openbmb/UltraData-SFT-Agent-2609, 2026.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Albert Tseng, Jerry Chee, Qingyao Sun, Volodymyr Kuleshov, and Christopher De Sa. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *Proceedings of machine learning research*, 235:48630, 2024a.

Albert Tseng, Qingyao Sun, David Hou, and Christopher De. Qtip: Quantization with trellises and incoherence processing. *Advances in Neural Information Processing Systems*, 37:59597–59620, 2024b.

Mart Van Baalen, Andrey Kuzmin, Ivan Koryakovskiy, Markus Nagel, Peter Couperus, Cedric Bastoul, Eric Mahurin, Tijmen Blankevoort, and Paul Whatmough. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*, 2024.

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

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*, 2026.

Ruizhe Wang, Yeyun Gong, Xiao Liu, Guoshuai Zhao, Ziyue Yang, Baining Guo, Zhengjun Zha, and Peng Cheng. Optimizing large language model training using FP4 quantization, 2025. URL https://arxiv.org/abs/2501.17116.

Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

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, 2019.

Xi Zhang, Xiaolin Wu, Jiamang Wang, and Weisi Lin. Learning grouped lattice vector quantizers for low-bit llm compression. *Advances in Neural Information Processing Systems*, 38:110672–110698, 2026.

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.

Appendix

CONTENTS

A Related Work **14**
A.1 Post-Training Quantization of Large Models 14
A.2 Quantization-Aware Training 14
A.3 Vector Quantization for Model Compression 14
A.4 Function Spaces and Complexity Ordering 14

B Theoretical Analysis and Proofs **15**
B.1 The Common Parameterization Is Exact 15
B.2 The Intermediate Designs Form a Continuum 16
B.3 The Curvature Seminorm and Theorem 1 16
B.4 The Family and the Rate 18
B.5 Measured Rate and Distortion 19
B.6 Kernel View 20
B.7 The Coefficient Gradient 21
B.8 The Index Surrogate and Its Jacobian 22

C Implementation Details **22**
C.1 The Quantization Pipeline 22
C.2 Initialization from a PTQ Model 23
C.3 The Fused Kernel 24

D Experimental Details **24**

E The Free End Is a Trainable Table **25**

F Comparing with Polynomial Function **26**

G Limitations **27**

H Social Impact **27**

A Related Work

A.1 Post-Training Quantization of Large Models

GPTQ propagates the quantization residual column by column at LLM scale (Frantar et al., 2022). AWQ reduces output error by scaling salient channels before rounding (Lin et al., 2024). Incoherence processing conjugates weights by random orthogonal transforms to remove outliers (Chee et al., 2023); QuIP# applies a randomized Hadamard for the same effect at inference speed (Tseng et al., 2024a), and Eq. 9 adopts this device. What does not carry over to QAT is the cost model: a calibration-time argmin is paid once, whereas a training-time one runs at every step while latent weights drift across assignment boundaries.

A.2 Quantization-Aware Training

The quantizer is piecewise constant, so every QAT method chooses where to inject a gradient the map does not supply. The straight-through estimator passes the loss gradient through unchanged (Bengio et al., 2013); Eq. 1 uses this rule with latent weights that accumulate gradients and are discarded at inference (Helwegen et al., 2019). LSQ sends the gradient to the step size (Esser et al., 2019), a single scalar of the decoder. LC-QAT defines a full affine map but fixes it at initialization and keeps it outside the optimizer, so the differentiability it gains is used to reach the weights and not the map (Wang et al., 2026). Our Θ generalizes the former from a scalar to that full affine map and trains it, and Γ carries the map outside the affine class altogether. At LLM scale the data cost is also relevant: LLM-QAT distills from the full-precision model (Liu et al., 2024) and EfficientQAT restricts the trained parameters (Chen et al., 2025). Our gradient paths are asymmetric: the path to the latent weights uses the differentiable estimator of Wang et al. (2025) (adopted by Wang et al. (2026)) composed with the decoder Jacobian; the path to (Θ, Γ) is exact, since Eq. 11 is linear in those coefficients at fixed indices.

A.3 Vector Quantization for Model Compression

VQ's advantage over SQ at equal bit width decomposes into a space-filling gain, a shape gain, and a memory gain from correlations across coordinates (Gersho & Gray, 2012); nonuniform scalar levels (Kim et al., 2023) capture only the shape gain. At the unconstrained end, GPTVQ fits a per-block table at $d = 2$ (Van Baalen et al., 2024) and AQLM sums codewords from several small tables (Egiazarian et al., 2024); the binding cost at two bits is initialization rather than refinement (Kennedy & Moosavi, 2026). At the constrained end, QuIP# generates codewords from the E_8 lattice (Tseng et al., 2024a), QTIP makes its codes lookup-free (Tseng et al., 2024b), and Leech lattice methods index without storing a table (van der Ouderaa et al., 2026). Between the two, GLVQ learns a generation matrix per group (Zhang et al., 2026), LiftQuant projects a binary lattice from a higher dimension (He et al., 2026a), and LC-QAT applies a learned affine map to an integer grid to eliminate search (Wang et al., 2026); an affine map preserves equal spacing, so the orientation adapts but the structure does not. The distinction Sec. 3.1 draws is that every design above fixes a codebook family and optimizes inside it, whereas Eq. 7 parameterizes the family itself: $m = 0$ is provably the affine design, $m = M$ is provably all of $\mathbb{R}^{K \times d}$, and m costs no index bits.

A.4 Function Spaces and Complexity Ordering

The curvature seminorm of Eq. 6 is the discrete counterpart of the thin-plate seminorm, whose null space on a continuous domain is exactly the affine functions (Duchon, 2006); Theorem 1(a) is the same statement on $\mathcal{Z}$. This is why Eq. 7 is a kernel expansion truncated by complexity rather than a polynomial truncated by degree, and Appendix B.6 makes the correspondence explicit. The intermediate designs form a Grassmann manifold of positive dimension (Absil et al., 2008) (Proposition 3), so a criterion is needed; the endpoints and the projection are properties of the subspace chain and would survive a different ordering.

B Theoretical Analysis and Proofs

This appendix collects the derivations deferred by Sec. 3, in the order that section makes the corresponding claims: that the common parameterization of Eq. 5 is exact (Appendix B.1), that the designs between the two ends form a continuum and therefore require a criterion (Appendix B.2), that the curvature of Eq. 6 is a seminorm vanishing exactly on the affine design, which is Theorem 1 (Appendix B.3), that Definition 1 is well posed and Proposition 1 holds (Appendix B.4). Appendix B.5 reports the rate and distortion measurements quoted in Sec. 3.1, which are measurements and not consequences of the results above, and Appendix B.6 names the construction in the language of kernel methods. The last two subsections turn to Sec. 3.3: Appendix B.7 derives the coefficient gradient and states precisely in what sense it is exact, and Appendix B.8 derives the code surrogate and shows that its Jacobian factor is an analytic derivative rather than an interpolation error.

All symbols are those of Sec. 3.1: $q \geq 2$ and $d \geq 1$ are integers, $b = \log_2 q$, $K = q^d$, $\mathcal{Z} = \{0, \dots, q-1\}^d$, and a map D on $\mathcal{Z}$ is identified with the matrix $C \in \mathbb{R}^{K \times d}$ of its values, $D(z) = C_z^\top$. The main setting is $b = 2$ and $d = 4$, hence $q = 4$, $K = 256$, and $M = 251$. Identities are stated in real arithmetic; an implementation must check them in finite precision separately.

B.1 The Common Parameterization Is Exact

Eq. 5 rewrites the codebook of VQ on the domain of the domain of the affine map, and Sec. 3.1 asserts that this relabeling leaves the quantizer untouched. Two things have to hold: the relabeling is a bijection, and it changes neither the maps available to VQ nor the distortion attained.

Lemma 1 (Base-q labeling)**.** *The map $\iota(z) = 1 + \sum_{j=1}^{d} z_j q^{d-j}$ is a bijection from $\mathcal{Z}$ onto $\{1, \dots, K\}$.*

Proof. Every integer in $\{0, \dots, q^d - 1\}$ has exactly one representation $\sum_{j=1}^{d} z_j q^{d-j}$ with digits $z_j \in \{0, \dots, q-1\}$, by uniqueness of the base-q expansion. Both sets have K elements, so ι is a bijection. □

Proposition 2 (Relabeling preserves the design and its distortion)**.** *Composition with ι is a linear bijection between the maps on $\{1, \dots, K\}$ and the maps on $\mathcal{Z}$. Under it, $\widetilde{D}_{\mathrm{VQ}}$ ranges over all of $\mathbb{R}^{K \times d}$ and $\widetilde{D}_{\mathrm{aff}}$ over the affine maps on $\mathcal{Z}$, and for every weight group $w \in \mathbb{R}^d$,*

$$\min_k \left\| w - D_{\mathrm{VQ}}(k) \right\|_2^2 = \min_{z \in \mathcal{Z}} \left\| w - \widetilde{D}_{\mathrm{VQ}}(z) \right\|_2^2. \tag{15}$$

Proof. Composition with a bijection is linear and invertible, its inverse being composition with ι^{-1}, which exists by Lemma 1. An unconstrained codebook is an arbitrary map on $\{1, \dots, K\}$, so its image under this bijection is an arbitrary map on $\mathcal{Z}$, that is, an arbitrary C; the affine maps are a subset of these. For Eq. 15, both minimizations range over the same finite set of values, since ι is onto, and a minimum over a set depends on the set alone. □

Proposition 2, together with Remark 1 on why the labeling is forced, is what Sec. 3.1 uses: the two designs of Eq. 4 become two subspaces of one $\mathbb{R}^{K \times d}$ with no approximation taken along the way, and in particular the claim that VQ realizes an arbitrary map survives the change of domain.

Remark 1 (The labeling is forced, not chosen). The curvature of Eq. 6 is not invariant under the choice of bijection: composing ι with a permutation of the labels permutes the rows of C and changes both the curvature and the family of Definition 1. The choice is nonetheless not free. The free end is indifferent to it, spanning the whole space for any bijection, so the labeling is pinned by the other end alone: $\widetilde{D}_{\mathrm{aff}}$ is affine in z only if the digits of the label are the coordinates of the grid point, which is Lemma 1. Any other bijection would place the affine design somewhere other than at the affine subspace, and the family would no longer start at the design current VQ-QAT uses. This is also the labeling the forward pass of Sec. 3.2 computes, so no permutation is applied anywhere in the implementation.

B.2 The Intermediate Designs Form a Continuum

Sec. 3.1 states that a design between the two ends corresponds to a subspace containing $\mathrm{col}(F)$, that infinitely many such subspaces exist at every intermediate dimension, and that selecting one therefore needs a criterion. The two ends, by contrast, admit no choice at all, which is why the criterion is needed only in between.

Lemma 2 (Rank of the affine design)**.** *For $q \geq 2$ and $d \geq 1$, the matrix F has full column rank $d+1$.*

Proof. Suppose $a_0 + \sum_j a_j z_j = 0$ for every $z \in \mathcal{Z}$. Evaluating at $z = 0$ gives $a_0 = 0$, and evaluating at $z = e_j$, which lies in $\mathcal{Z}$ because $q \geq 2$, then gives $a_j = 0$ for every j. The $d+1$ columns of F are therefore linearly independent. □

A design generates C as a fixed matrix times free coefficients, and the codebooks it reaches form the column space of that matrix, taken in each output column; two designs with the same column space are the same design. Requiring a design to contain the affine one is therefore requiring its column space to contain $\mathrm{col}(F)$, so the intermediate designs of a given size are exactly the subspaces of $\mathbb{R}^K$ that contain $\mathrm{col}(F)$ and have that dimension.

Proposition 3 (A positive-dimensional family of candidates)**.** *The subspaces of $\mathbb{R}^K$ that contain $\mathrm{col}(F)$ and have dimension $d+1+m$ are in bijection with the m-dimensional subspaces of $\mathrm{col}(F)^\perp$, hence with the points of a Grassmann manifold of dimension*

$$m\,(M-m). \tag{16}$$

This is strictly positive for $0 < m < M$ and zero at $m = 0$ and $m = M$.

Proof. By Lemma 2, $\mathrm{col}(F)^\perp$ has dimension M. A subspace containing $\mathrm{col}(F)$ is determined by its intersection with $\mathrm{col}(F)^\perp$, of dimension $d+1+m-(d+1) = m$, and conversely an m-dimensional subspace of $\mathrm{col}(F)^\perp$ determines one as its direct sum with $\mathrm{col}(F)$; the two assignments are mutually inverse. Eq. 16 is the dimension of the real Grassmann manifold of m-planes in M dimensions (Absil et al., 2008), and both factors are positive exactly when $0 < m < M$. □

Two consequences are what Sec. 3.1 appeals to. For $0 < m < M$ the candidates form a positive-dimensional manifold and are therefore uncountable, so no counting argument selects an intermediate design and a criterion has to be supplied; at the main setting Eq. 16 is 250 at $m = 1$ and 15 750 at its largest. At $m = 0$ and $m = M$ the manifold is a single point, so there is exactly one affine design and exactly one unconstrained codebook, and the family of Definition 1 does not select among variants at its ends but reproduces the only ones available. This is the sense in which its endpoints are exact.

B.3 The Curvature Seminorm and Theorem 1

Eq. 6 asserts a matrix form for the curvature and a positive semidefinite Ω depending only on q and d; Sec. 3.1 further asserts that neither the order of the difference nor the mixed terms may be dropped, and Theorem 1 identifies the kernel and the rank of Ω. This subsection constructs Ω, establishes the seminorm property, proves the theorem, and shows that both restrictions are necessary.

Construction of Ω. The difference Δ_{jk} is evaluated wherever its arguments lie in $\mathcal{Z}$, that is, at those z with $z_j \leq q-3$ when $j = k$ and with $z_j, z_k \leq q-2$ when $j \neq k$, the first set being empty when $q = 2$. Collect the coefficients of these differences, applied to a single output column of C, into a matrix H: one row for each pure difference Δ_{jj} at each admissible z, and one row $\sqrt{2}\,\Delta_{jk}$ for each $j < k$ at each admissible z, the factor $\sqrt{2}$ accounting for the ordered pair (k, j) that the sum in Eq. 6 includes as well. For $d \geq 2$ the number of rows is $d(q-2)q^{d-1} + \binom{d}{2}(q-1)^2 q^{d-2}$, which is $512 + 864 = 1376$ at $q = d = 4$; for $d = 1$ only the first term is present. Applying H to every output column gives

$$\mathcal{J}(C) = \|HC\|_F^2 = \mathrm{tr}(C^\top \Omega C), \qquad \Omega = H^\top H, \tag{17}$$

which is Eq. 6. Hence Ω is positive semidefinite, acts identically on every output coordinate, and depends only on q and d, as claimed there.

Why a seminorm. Since $\|\cdot\|_F$ is a norm and $C \mapsto HC$ is linear, $\mathcal{J}^{1/2}$ is absolutely homogeneous and satisfies the triangle inequality, so it is a seminorm on $\mathbb{R}^{K\times d}$. It is not a norm, because H has a nontrivial kernel by Theorem 1(a) below. The degeneracy is the reason this is the right functional here: a norm would measure how large a codebook is, whereas a seminorm vanishing exactly on the affine design measures only how far a codebook departs from that design, and assigns zero to every affine map regardless of its scale. Sec. 3.1 ranks directions by this quantity, so it is the degeneracy that makes the ranking start at the affine end.

Proof of Theorem 1. *(a)* If $D(z) = Az + t$ then every pure and mixed second difference vanishes identically, so $\mathrm{col}(F) \subseteq \ker\Omega$. For the converse it suffices, by Eq. 17 and the fact that Ω acts on each output column separately, to treat a single column $h \in \mathbb{R}^K$ with $Hh = 0$. Fix j and let $g(z) = h(z+e_j) - h(z)$, defined for $z_j \le q-2$. The equation $\Delta_{jj}h(z) = 0$ says $g(z+e_j) = g(z)$ whenever both arguments are defined, so g does not vary along coordinate j; when $q = 2$ no such z exists, but then $z_j = 0$ is the only available value and the conclusion holds with no pure difference used. For $k \neq j$, the equation $\Delta_{jk}h(z) = 0$ says $g(z + e_k) = g(z)$, so g does not vary along coordinate k either. The domain of g is a rectangular grid and is therefore connected by single coordinate steps, so g is a constant a_j. Summing first differences along a coordinate path from the origin gives $h(z) = h(0) + \sum_j a_j z_j$, which is affine, hence $\ker\Omega = \mathrm{col}(F)$.

(b) By Lemma 2 the kernel has dimension $d + 1$, so part (a) and the rank-nullity theorem give $\mathrm{rank}\,\Omega = K - d - 1 = M$. As Ω is symmetric, $\mathbb{R}^K$ decomposes orthogonally into $\ker\Omega = \mathrm{col}(F)$ and the span of the eigenvectors with positive eigenvalues, so those M eigenvectors form an orthonormal basis of $\mathrm{col}(F)^{\perp}$. □

The order of the difference is forced. Sec. 3.1 claims that second differences are the only order placing the affine design, and nothing besides it, at zero. First differences vanish only on constant maps, since $D(z+e_j) = D(z)$ for every j and every z forces D to be constant on the connected grid, so their kernel has dimension one and the term Az that every deployed quantizer already uses would be charged for, placing the start of the family strictly below the design in use. Third differences vanish on every polynomial of coordinate degree at most two, a class containing non-affine maps such as z_1^2, so their kernel strictly contains $\mathrm{col}(F)$ and the start would lie strictly above it. Orders beyond the third are excluded with the third: a p-th difference annihilates every polynomial of coordinate degree at most $p-1$, so its kernel contains that of the third for all $p \ge 3$. Order two is therefore the only remaining possibility, and Theorem 1(a) shows it meets the requirement exactly.

The mixed differences are necessary. Sec. 3.1 claims that dropping the terms $j \neq k$ would leave the family short of VQ. Suppose they are removed. Vanishing pure second differences require h to be affine in each coordinate separately, which is exactly the class of multilinear maps, spanned by the 2^d products of distinct coordinates and the constant; these are linearly independent already on $\{0,1\}^d$. The kernel therefore grows from $d+1$ to 2^d and contains non-affine maps such as $z_1 z_2$, so part (a) of the theorem fails. What fails with it is the free end rather than the affine one: since F is appended explicitly in Eq. 7, the affine end survives any choice of operator, whereas a pure-only operator has $K - 2^d$ positive modes, so $[F, V_M]$ has rank $K - 2^d + d + 1$ and a subspace of dimension $2^d - d - 1$, namely the non-affine multilinear maps, is unreachable at every level. At $q = d = 4$ that is 11 directions out of 256, and the family would terminate at a proper subspace of $\mathbb{R}^{K\times d}$. The mixed differences are needed for this construction of a family terminating at the free end; they are not the only conceivable construction of one.

Remark 2 (The weight on the mixed terms). Replacing the factor $\sqrt{2}$ above by any positive weight leaves the proof of Theorem 1 intact, since it uses only that both groups of differences vanish, so the kernel is $\mathrm{col}(F)$ and the rank is M for every such weight, and with them both ends of the family. What the weight changes is the eigenvalues, hence the order in which directions are admitted and the contents of a given level. The value $\sqrt{2}$ follows the convention by which the Frobenius norm of a symmetric Hessian counts its off-diagonal entries twice, and the construction is insensitive to it: on the reconstruction probe of Appendix B.5, squared weights spanning an order of magnitude, 1, 2, 5.56 and 10, leave the held-out error at $m = 43$ unchanged to within the resolution of the probe. The ordering is aligned to the coordinates of the grid by design, which is what the domain supports: an orthogonal map sends the centered grid to itself only if it permutes and reflects the axes, and Ω commutes with that action for every positive weight.

B.4 The Family and the Rate

Definition 1 is a definition and is not proved. What requires an argument is that it is well posed and that it has the properties Sec. 3.1 attributes to it: the stated dimension, the strict nesting, and the two exact ends. Proposition 1 is proved at the end of the subsection.

Proposition 4 (The family is well posed)**.** *For every $m \le M$, the set $\mathcal{C}_m$ of Eq. 7 is a linear subspace of $\mathbb{R}^{K\times d}$ of dimension $d(d+1+m)$, and*

$$\mathcal{C}_0 \subsetneq \cdots \subsetneq \mathcal{C}_M, \qquad \mathcal{C}_0 = \{F\Theta\}, \qquad \mathcal{C}_M = \mathbb{R}^{K\times d}. \tag{18}$$

Proof. $\mathcal{C}_m$ is the image of the linear map $(\Theta, \Gamma) \mapsto F\Theta + V_m\Gamma$ and is therefore a subspace. The columns of $[F, V_m]$ are linearly independent: those of F are, by Lemma 2, and those of V_m are orthonormal and lie in $\mathrm{col}(F)^\perp$, by Theorem 1(b). Each of the d output columns of C therefore carries $d+1+m$ free coefficients and $\dim \mathcal{C}_m = d(d+1+m)$. Passing from m to $m+1$ appends a direction orthogonal to all those already present, so the inclusion is strict. At $m = 0$ the set is $\{F\Theta\}$, the affine design, and at $m = M$ the columns of $[F, V_M]$ are K linearly independent vectors of $\mathbb{R}^K$ by Theorem 1(b), so they span it and $\mathcal{C}_M = \mathbb{R}^{K\times d}$. □

With Proposition 2, Eq. 18 says that the two ends of the family are the two designs of Eq. 4 exactly, and with Proposition 3 that they are the only designs available at those two dimensions.

Admissible levels. Sec. 3.1 restricts m to the levels that close a complete eigenspace. The reason is that $\mathcal{C}_m$ is determined by $\mathrm{col}(V_m)$, and a level cutting inside an eigenspace of multiplicity greater than one does not determine that span: any orthonormal basis of the eigenspace is a valid set of eigenvectors, so the first m of them depend on which basis an eigensolver returns. Admitting only those m with $\rho_m < \rho_{m+1}$, together with $m = 0$ and $m = M$, makes $\mathrm{col}(V_m)$ a sum of whole eigenspaces, hence a function of Ω alone, and leaves the family of Proposition 4 strictly increasing along the admitted levels. At the main setting the 251 positive eigenvalues take only 44 distinct values with multiplicities up to 12, so 45 levels are admissible, beginning 0, 6, 10, 14, 15, 27. The degeneracy is structural: Ω treats the index axes symmetrically and is unchanged by the reflection $z \mapsto q - 1 - z$, so it commutes with the action of the signed axis permutations, of order $2^d d!$, which is 384 at $d = 4$, and its eigenspaces are unions of isotypic components of that action; the observed multiplicities $1, 2, 3, 4, 6, 8, 12$ are dimensions attached to them. The eigenbasis does not, however, factor across axes: at $q = 4$ the Gram matrices of the first and second differences in a single coordinate do not commute. The list depends only on q, d, and the weight of Remark 2, and is the capacity axis used throughout the paper. Admissibility is a well-posedness condition and not a claim that the admissible levels are equally useful.

Proof of Proposition 1. A group of d weights stores one index into $K = q^d$ codewords, which is $\log_2 K = d \log_2 q = db$ bits, or $\log_2(K)/d = b$ bits per weight. This count is the same at every level, because Eq. 7 changes what a codeword is and not how many codewords there are. A decoder at level m has $d(d+1+m)$ coefficients by Proposition 4; at β_c bits each, shared by G groups and therefore by Gd weights, they add $d(d+1+m)\beta_c/(Gd) = (d+1+m)\beta_c/G$ bits per weight. The index bits and the coefficient bits are disjoint parts of the stored representation, so the two terms sum, which is Eq. 8. □

We call the second term negligible when it stays below 0.05% of the index budget, which at $b = 2$ is 10^{-3} bits per weight, and read every threshold below against that tolerance. The term is decreasing in G and increasing in m, so the tolerance is met above a threshold on G that grows linearly in $d+1+m$, reaching $K\beta_c$ divided by the tolerance at $m = M$, since $d+1+M = K$. That threshold is exponential in d at fixed b, so the regime in which the term is negligible is one the configuration enters and not one the family guarantees: at $b = 2$ it asks for $G \gg 256\,\beta_c$ at $d = 4$ and for $G \gg 65{,}536\,\beta_c$ at $d = 8$. Tab. 2 evaluates the term for the three sharing units of the main model, one decoder per projection slot, and two readings follow. First, the term is negligible because G is large and not because the family economizes on coefficients: at $m = M$ the coefficient count $d(d+1+M) = dK$ equals the size of a raw table exactly, so the reduction in decoder parameters, by a factor of $1024/20 = 51.2$ at $m = 0$, exists only for $m < M$. Second, the amortization is uneven across units. Averaged over the weights of a layer the term is 2.4×10^{-6} bits per weight at

Table 2: Decoder side information, in bits per weight, for Qwen3.6-35B-A3B at $b = 2$, $d = 4$, with $\beta_c = 32$. The index term is 2.000 throughout. Percentages are of the 2-bit index budget.

sharing unit	weights Gd	$m = 0$	$m = M$
MoE `gate_up` slot	5.4×10^8	1.2e-6	6.1e-5
MoE `down` slot	2.7×10^8	2.4e-6	1.2e-4
shared-expert linear	1.0×10^6	6.1e-4	3.1e-2
as a fraction of b, worst unit		0.03%	1.56%
as a fraction of b, layer average		$< 0.001\%$	0.01%

Table 3: Information delivered at a level, $b = 2$, $d = 4$. The nominal rate is 2.000 bits per weight in every row. Gap closed is the fraction of the distortion difference between the affine and free ends.

m	params	of table	$\mathcal{E}$	eff. rate	entropy
0	20	2.0%	0.11896	1.536	1.904
15	80	7.8%	0.11472	1.562	1.912
27	128	12.5%	0.10721	1.611	1.945
43	192	18.8%	0.10569	1.621	1.949
95	400	39.1%	0.10160	1.650	1.964
139	576	56.2%	0.09891	1.669	1.966
251	1024	100%	0.09629	1.688	1.968

$m = 0$ and 1.2×10^{-4} at $m = M$, in both cases below the last digit of the nominal 2.000, since the two MoE slots hold 99.87% of the quantized weights and clear that threshold by a factor of 8 or more even at the free end. The shared-expert linear does not: with $G = 2.6 \times 10^5$ groups it crosses the threshold at the second admissible level, $m = 6$, and reaches 1.56% of the index budget at $m = M$. The claim of Sec. 3.1 is therefore one about the layer as a whole and about the MoE slots that dominate it, not about every unit taken in isolation.

Proposition 1 counts the index and the decoder coefficients only. A deployed model also stores the per-slot scales and the sign vectors of the random transform of Sec. 3.2, which are common to all levels and to the baselines, and a serialized artifact that does not deduplicate the fixed basis across decoders will exceed this count.

B.5 Measured Rate and Distortion

Sec. 3.1 reports how much of the nominal rate an index delivers at a level and attributes the shortfall to uneven occupancy of the codewords and to the affine constraint. Those statements are measurements on a calibrated source, not consequences of the preceding results.

Three quantities are measured on an i.i.d. standard Gaussian source, which the random transform of Sec. 3.2 makes an accurate stand-in for the transformed weights: the per-coordinate distortion $\mathcal{E}$ under nearest-neighbor encoding, the per-weight entropy of the resulting index distribution, and the effective rate $\frac{1}{2}\log_2(1/\mathcal{E})$, which is the rate at which an ideal code would attain the same distortion, obtained by inverting the Gaussian rate-distortion function. Each level is fitted to the source by alternating nearest-neighbor assignment with least squares inside $\mathcal{C}_m$, the inner step being the orthogonal projection onto $\mathcal{C}_m$. The calibration is checked against the measured weights: the affine level attains $\mathcal{E} = 0.11896$ on the synthetic source against 0.11898 on held-out experts, and an unconstrained codebook fitted by Lloyd's algorithm 0.09609 against 0.09630. The fitted level $m = M$ in Tab. 3 attains 0.09629, within 0.2% of that reference, the residual being the alternating fit and not the family, which is unconstrained at $m = M$ by Proposition 4.

Tab. 3 substantiates the quoted figures. The affine end delivers 1.904 of its 2 nominal bits, losing the remainder to uneven occupancy of the 256 cells, and its distortion is that of an ideal code at 1.536 bits, so a further 0.46 bits is lost to the affine constraint itself, the larger of the two shortfalls. That end is also where a scalar quantizer sits in distortion: an optimal 2-bit uniform quantizer on this source attains 0.11885 against the 0.11896 of the affine level, so the extra freedom of a full A yields orientation and not error. Across the family, 0.064 of the 0.096 bits of entropy and 0.152 of the

0.464 bits of effective rate are recovered, which are the two thirds and one third quoted in Sec. 3.1, and 51.6% of the distortion gap between the two ends is closed at $m = 27$, taking the fitted free end as the reference, using 12.5% of the parameters of a table.

Why the target is the free end and not a wider integer format. On the same source, 4-bit uniform integer quantization attains $\mathcal{E} = 0.01172$, whereas the Shannon lower bound at rate 2 is $2^{-2b} = 0.0625$, which is larger. No code at 2 bits per weight therefore reaches the weight error of 4-bit integers, including the free end of this family and including vector quantization at any d; conversely the distortion of the free end corresponds to uniform integer quantization at 2.19 bits. Comparisons between 2-bit Vodka and 4-bit baselines are accordingly about downstream accuracy and not about weight reconstruction. The two differ because the induced weight error is the codebook error weighted by how often each codeword is used, and the resulting output error depends further on the input distribution and on the sensitivity of the network.

Scope of the measurement. These numbers are for one model configuration and one synthetic source calibrated to it, and they concern weight reconstruction at fixed assignments rather than task accuracy, under the encoder of Remark 3 rather than Eq. 10. They also concern the distortion attainable at a level, not the ordering of directions within it: what the curvature ordering is shown to provide is a family with exact endpoints (Proposition 4), a closed-form projection at every level, and a basis-independent list of levels. Whether it also orders the non-affine directions better than an ordering by polynomial degree is an empirical question, settled by the ablations and not by any result in this appendix.

Remark 3 (The encoder coincides with nearest neighbor at the affine end). The measurement above encodes by nearest neighbor, whereas Eq. 10 rounds coordinate-wise. The two coincide at $m = 0$ under the initialization of Appendix C.2: there $A = s_0\Psi$ with Ψ orthogonal, so for $u = A^{-1}(w - t)$ one has $\|w - (Az + t)\|_2^2 = s_0^2\|u - z\|_2^2$, and minimizing over $z \in \mathcal{Z}$ is exactly the coordinate-wise rounding and clipping of u. Beyond the affine end the decoder is no longer an isometry of the grid and the two encoders separate, so Table 3 reports what a level can deliver rather than what Eq. 10 attains at it, which is how Sec. 3.1 quotes it. This is also the sense in which Vodka holds the encoder fixed and moves the expressivity into the decoder: the search-free forward pass of Sec. 3.2 is the nearest-neighbor encoder at the affine end and a fixed coordinate-wise rule at every level above it.

B.6 Kernel View

The construction of Sec. 3.1 is the standard machinery of a semi-Hilbert space, which names each of its pieces and is the sense in which the codebook of this paper is kernel induced. The curvature of Eq. 6 is the discrete counterpart of the second-order Beppo-Levi seminorm, the integrated squared Hessian, and because it vanishes on $\mathrm{col}(F)$ by Theorem 1(a) it is a seminorm rather than a norm, so the space is semi-Hilbert and $\Omega^\dagger$ is not a reproducing kernel on all of $\mathbb{R}^K$. It is instead conditionally positive definite of order 2, a kernel modulo affine maps. Take $h^\top \Omega h'$ as the semi-inner product, which is the polarization of Eq. 6 on a single output column. Then the column of $\Omega^\dagger$ at a grid point reproduces the value of h there,

$$h^\top \Omega\, \Omega^\dagger e_z = h(z) \qquad \text{for } h \in \mathrm{col}(F)^\perp, \tag{19}$$

Eq. 19 holds because $\Omega\Omega^\dagger$ is the orthogonal projector onto $\mathrm{col}(F)^\perp$ by Theorem 1(b) and h already lies there. The columns of $\Omega^\dagger$ lie in $\mathrm{col}(F)^\perp$ for the same reason, so that subspace under this semi-inner product is a reproducing kernel Hilbert space with kernel $\Omega^\dagger$. Its Karhunen–Loève basis is $v_1, \ldots, v_M$: the decomposition $\Omega^\dagger = \sum_{j \le M} \rho_j^{-1} v_j v_j^\top$ gives features $v_j/\sqrt{\rho_j}$ that are orthonormal under it and ordered by decreasing kernel eigenvalue ρ_j^{-1}. Adding the affine part back, $\mathcal{C}_m$ is spanned by the m leading eigenfunctions of that kernel together with $\mathrm{col}(F)$, so Eq. 7 is a kernel expansion truncated by complexity rather than a polynomial truncated by degree, and the reparameterization between Γ and kernel coefficients is a change of basis. The kernel is fixed once q, d, and the weight of Remark 2 are chosen; training adjusts the expansion coefficients and not the kernel or its eigenvectors. The space is over the finite grid $\mathcal{Z}$, so the continuous extension used for index gradients in Sec. 3.3 is a separate choice and is not implied by this construction.

B.7 THE COEFFICIENT GRADIENT

Sec. 3.3 states that the decoder coefficients receive an exact gradient at fixed indices and contrasts where that gradient goes with Eq. 3. Both statements are derived here, and the scope of the word exact is made precise.

The chain has three links and each is linear, which is the substance of the claim: the quantized group is the row of C that its index selects, the table is $C = F\Theta + V_m\Gamma$ by Eq. 11, and the loss depends on the group only through $\hat{w}_g$. Write $\Xi = [\Theta; \Gamma]$ for the coefficients stacked, so that $C = [F, V_m]\,\Xi$. Fix the indices $\{z_g\}$ and let $k_g = \iota(z_g)$. The forward pass reads $\hat{w}_g^\top = C_{k_g,:} = e_{k_g}^\top [F, V_m]\,\Xi$, which is linear in Ξ, so for a perturbation $\delta\Xi$ the first-order change of the loss is

$$\delta L = \sum_g \eta_g^\top\, \delta\hat{w}_g = \sum_g \eta_g^\top \left(e_{k_g}^\top [F, V_m]\,\delta\Xi\right)^\top = \operatorname{tr}\Big(\delta\Xi^\top [F, V_m]^\top \underbrace{\textstyle\sum_g e_{k_g}\eta_g^\top}_{=G_C}\Big), \tag{20}$$

with $\eta_g = \partial L/\partial\hat{w}_g$ as in Sec. 3.3. The inner sum is exactly the row-pooled matrix of Eq. 13, since $e_{k_g}\eta_g^\top$ deposits $\eta_g^\top$ in row k_g and the sum over g therefore accumulates $[G_C]_{k,:} = \sum_{g:\,\iota(z_g)=k} \eta_g^\top$. Reading off the coefficient of $\delta\Xi$ gives $\nabla_\Xi L = [F, V_m]^\top G_C$, whose two blocks are $\nabla_\Theta L = F^\top G_C$ and $\nabla_\Gamma L = V_m^\top G_C$. No step of Eq. 20 linearizes anything, which is the sense in which the coefficient path needs no estimator.

Remark 4 (The basis convention of the implementation). Eq. 13 is stated in the basis $[F, V_m]$, which is convenient for reading the two blocks separately but is not orthonormal, since the columns of F are neither unit vectors nor mutually orthogonal. The implementation instead stores the single matrix B obtained by orthonormalizing $[F, V_m]$, together with a scale λ on the coefficients, so that $C = B(\lambda\Xi)$ and the gradient it computes is $\lambda B^\top G_C$. The two are the same object written in different bases: B and $[F, V_m]$ span the same $\mathcal{C}_m$ by Proposition 4, and the change of basis together with λ is absorbed into the coefficients, which are trainable and initialized by solving $B\Xi^0 = C^0$ for the PTQ table rather than by reading Eq. 7 directly. Nothing in Theorem 1 or in the projection onto $\mathcal{C}_m$ depends on the choice, since both refer to $\mathrm{col}(V_m)$ and to $\mathrm{col}(F)$ and not to particular spanning vectors. An orthonormal B is preferred in the implementation because $\|\Xi\|_F = \|C\|_F/\lambda$ then holds, so a single learning rate acts comparably on the affine and the non-affine coefficients.

Remark 5 (In what sense exact). The gradient above is the true derivative of the loss with respect to (Θ, Γ) when the indices $\{z_g\}$ are held fixed, and that is the only sense claimed. It is not a derivative of the map that also lets the indices respond to the coefficients: as Θ moves, a latent weight sitting near a rounding boundary of Eq. 10 may cross it, and the loss is then discontinuous in the coefficients. The set of coefficient values at which some group sits exactly on a boundary has measure zero, so the formula is the derivative almost everywhere, but the loss is not differentiable across those surfaces and no expansion of the composite map is claimed. The finite-difference check released with our code verifies the formula under exactly this condition, holding the assignment fixed: at $q = d = 4$ with 5000 groups the analytic gradient matches a central difference to a maximum relative error of 9.2×10^{-11} at $m = 0$, 1.3×10^{-10} at $m = 6$, 2.5×10^{-10} at $m = 43$ and 7.8×10^{-10} at $m = 251$.

Remark 6 (Where the gradient goes). A stored codebook is differentiable at fixed assignments too, so differentiability is not what distinguishes the two designs. The difference is how a step is parameterized. In Eq. 3 the Kd entries of the table move independently, and the k-th codeword is moved only by the groups currently assigned to it. Under Eq. 13 the gradient reaches $d(d + 1 + m)$ coefficients instead, each contributing to every row of C through F and V_m, so every group moves every codeword and each coefficient is estimated by pooling over all the groups a decoder serves. At the main setting that is 1024 numbers at the free end and 128 at $m = 27$, against the 1024 of a table, updated from order 10^8 groups. The distinction is not the reach of a step on its own: occupancy is uneven, with the 256 cells of the affine level selected at rates spanning a factor of 22 on the calibration source of Appendix B.5, so a stored codebook updated from a batch of 256 groups leaves 46% of its codewords untouched in expectation, but the figure falls below 1% past 4096 groups and a single decoder here supplies far more. What remains at any batch size is the dimension in which the step is taken, which is why Sec. 3.3 attributes the convergence of Sec. 4.4 to the shared parameterization rather than to unreached codewords. This is an argument about how a step is estimated and not a claim that any particular step improves the loss.

B.8 The Index Surrogate and Its Jacobian

Eq. 14 factors the path to a latent weight into the derivative of the discretization and the Jacobian of the decoder. The first factor is the DGE estimator of Wang et al. (2025) and is not derived here. The second is not an estimator at all, which is the point of this subsection.

The map D is defined only on $\mathcal{Z}$, so differentiating it requires extending it to a neighborhood, and an arbitrary extension would make J an artifact of that choice. It is not arbitrary here.

Proposition 5 (The stencil is the analytic derivative)**.** *Every column of $C \in \mathbb{R}^{K\times d}$ is the value table of a unique polynomial in z of degree at most $q-1$ in each coordinate separately, and the q-point Lagrange cardinal basis on the nodes $0,\dots,q-1$ spans exactly that space. Consequently, contracting C along coordinate j with the cardinal derivative matrix $[\mathcal{D}_q]_{rs} = \ell_s'(r)$ returns the derivative of that polynomial at the grid points, not an approximation of it.*

Proof. Write $\Pi_{q,d}$ for the space of real polynomials in z of degree at most $q-1$ in each coordinate separately. A basis is the set of monomials z^α with $\alpha \in \{0,\dots,q-1\}^d$, so $\dim \Pi_{q,d} = q^d = K$, which is also the number of nodes of $\mathcal{Z}$. Let $\ell_0,\dots,\ell_{q-1}$ be the Lagrange cardinal basis on the nodes $0,\dots,q-1$, so $\ell_s(r) = \delta_{rs}$, and let $L_\alpha(z) = \prod_j \ell_{\alpha_j}(z_j)$ be its tensor product. Then $L_\alpha \in \Pi_{q,d}$ and $L_\alpha(z') = \delta_{\alpha z'}$ for $z' \in \mathcal{Z}$, so the K functions L_α are linearly independent and span $\Pi_{q,d}$ by the dimension count. Given a column $h \in \mathbb{R}^K$, the polynomial $p = \sum_\alpha h(\alpha) L_\alpha$ therefore lies in $\Pi_{q,d}$ and satisfies $p(z) = h(z)$ on $\mathcal{Z}$; it is unique, since the difference of two such interpolants lies in $\Pi_{q,d}$ and vanishes at all K nodes, hence is zero by the same independence. Tensor-product interpolation on $\mathcal{Z}$ is thus unisolvent for $\Pi_{q,d}$, which is the first claim.

For the second, differentiation along coordinate j is linear, so it is represented in the cardinal basis by its action on the basis functions. Expanding p and differentiating one factor gives $\partial_j p(z) = \sum_\alpha h(\alpha)\, \ell_{\alpha_j}'(z_j) \prod_{k\neq j} \ell_{\alpha_k}(z_k)$, and evaluating at a node $z \in \mathcal{Z}$ collapses every factor with $k \neq j$ to $\delta_{\alpha_k z_k}$. What survives is the contraction of h along coordinate j with the matrix $[\mathcal{D}_q]_{rs} = \ell_s'(r)$, so that contraction returns $\partial_j p$ at the grid points exactly. Since p is the unique interpolant, no approximation is involved and the stencil is not a finite difference accurate only to some order. Applying the argument to each column of C and collecting the results for $j = 1,\dots,d$ gives the Jacobian $J(z)$ of Eq. 14. □

Two consequences follow for Sec. 3.3. First, since $\mathcal{C}_m \subseteq \mathbb{R}^{K\times d}$ for every m by Proposition 4, the proposition applies at every level of the family, so one implementation of the Jacobian serves the whole capacity axis and the gradient formula does not change with m. Second, at $m = 0$ the codebook is affine and its polynomial extension is that same affine map, so J reduces to the constant matrix of Eq. 4 and Eq. 14 degenerates to the estimator of the affine design. Both consequences are checked numerically by the same script. At $q = d = 4$ the contraction of Proposition 5 agrees with the analytic derivative of the interpolant to 2.2×10^{-10} in the maximum norm, at $m = 0$, 43 and 251, and at $m = 0$ the Jacobian returned by the implementation is constant across codewords and equals the affine matrix A of Eq. 4 to 7×10^{-15}.

Remark 7 (What remains a surrogate). Proposition 5 removes the approximation from the second factor only. The first factor replaces the derivative of a piecewise constant map, which is zero almost everywhere, by a nonzero function, so the product in Eq. 14 is not the derivative of the hard forward pass and the tilde in that equation is not superfluous. A step along it changes the quantized weight only once a latent weight crosses a rounding boundary. The stencil carries one practical consequence: its largest entry is 3 at $q = 4$ against 1 for a central difference, so index gradients are scaled differently from a central-difference implementation and the two legs are not directly comparable.

C Implementation Details

C.1 The Quantization Pipeline

A quantized projection applies Eq. 9 in five steps: the input diagonal scale $\mathrm{diag}(\sigma_{\mathrm{in}})$, the block Hadamard transform H_{in}, the linear map against the dequantized $\hat{W}$, the output transform H_{out}, and the output diagonal scale $\mathrm{diag}(\sigma_{\mathrm{out}})$. The two diagonal scales are trainable and the two transforms

are fixed. Since H_{in} and H_{out} are orthogonal and the scales are invertible, the pair surrounding $\hat{W}$ is exactly invertible and contributes no error of its own: the only approximation in the pipeline is the rounding of Eq. 10.

Groups are formed from d consecutive entries along the input dimension, so a weight matrix of $d_{\text{in}} \times d_{\text{out}}$ carries $d_{\text{in}} d_{\text{out}}/d$ indices, and a decoder serving G groups covers one projection slot, which for the mixture-of-experts model spans the corresponding matrix of every expert. At $d = 4$ the input dimensions of both models we quantize are divisible by d, so no padding is required. The whole pass is as shown in Algorithm 1.

Algorithm 1. Quantized forward pass at level m

Input: activations $X \in \mathbb{R}^{n \times d_{\text{in}}}$; latent weights $\{u_g\}$, $u_g \in \mathbb{R}^d$, for the G groups the decoder serves; coefficients $\Theta \in \mathbb{R}^{(d+1)\times d}$ and $\Gamma \in \mathbb{R}^{m \times d}$; trainable diagonal scales $\sigma_{\text{in}}, \sigma_{\text{out}}$.

Fixed: the affine basis F and the admitted directions V_m of Definition 1; the block Hadamard matrices $H_{\text{in}}, H_{\text{out}}$; the label map ι.

Once per decoder.

1. Materialize the codebook of Eq. 11,
$$C = F\Theta + V_m\Gamma \in \mathbb{R}^{K \times d},$$
at cost $O\big(Kd(d+1+m)\big)$, which depends on q, d, m alone and not on G.

Once per group $g = 1, \ldots, G$**.**

2. Round and clip coordinate-wise, by Eq. 10,
$$z_g = \text{clip}\,\big(\,\text{round}(u_g), 0, q-1\big) \in \mathcal{Z},$$
which requires no search over the K codewords.
3. Read the label $\iota(z_g)$ from the d coordinates of z_g as base-q digits.
4. Decode the group from that row, $\hat{w}_g^\top = C_{z_g}$, in the notation of Eq. 5.

The three steps together cost $O(Gd)$ and are independent of m.

Once per layer.

5. Place $\{\hat{w}_g\}$ into $\hat{W}$ at the positions their groups occupy, and evaluate Eq. 9,
$$Y = \big(X\,\text{diag}(\sigma_{\text{in}})\,H_{\text{in}}\big)\,\hat{W}\,H_{\text{out}}\,\text{diag}(\sigma_{\text{out}}).$$

Output: the layer output Y; the indices $\{z_g\}$, which Eq. 13 and Eq. 14 reuse on the backward.

C.2 Initialization from a PTQ Model

Following Wang et al. (2026), every run starts from a vector-quantized PTQ model rather than from a random codebook, and the conversion is the one that paper specifies, read into the parameterization of Eq. 11.

The PTQ stage produces a codebook of the affine form $c = Az + t$ with $A \in \mathbb{R}^{d\times d}$ and $t \in \mathbb{R}^d$, whose entries are set from a random orthogonal $\Psi \in \mathbb{R}^{d\times d}$ as

$$A = s_0\Psi, \qquad t = -s_0\Psi\mu\mathbf{1}, \qquad \mu = \frac{q-1}{2}, \qquad s_0 = \sqrt{\frac{12}{q^2-1}}, \tag{21}$$

so that the codebook it defines is centered, of approximately unit variance, and weakly correlated across coordinates. The quantized weights themselves are produced by LDLQ (Chee et al., 2023), which quantizes column-wise and compensates the reconstruction error of the columns not yet visited, under the Hadamard transform of Eq. 9.

Reading the codebook of Eq. 21 into a level of the family is a change of basis alone. Collecting the values of $c = Az + t$ at the grid points of $\mathcal{Z}$ gives the table $C^0 = F\Theta^0$ with $\Theta^0 = [\,t^\top; A^\top]$, so we initialize

$$\Theta = \Theta^0, \qquad \Gamma = 0, \tag{22}$$

and set the latent weight of each group to the index that stage assigned it, $u_g = z_g^0$. Since $\text{col}(F) \subseteq \mathcal{C}_m$ for every m, Eq. 22 reproduces the PTQ table exactly at every level and not approximately. Under the basis convention of Remark 4 the same initialization is obtained by solving $B\Xi^0 = C^0$, since B and $[F, V_m]$ span the same subspace.

C.3 The Fused Kernel

Sec. 3.4 states that the forward and the backward of a quantized projection fit in one kernel because the two tables stay resident. This subsection gives the traversal and the one numerical condition.

Both tables are addressed by the index and sized by q and d: C holds Kd entries and J holds Kd^2, so neither grows with the layer, and at $d = 4$ both are held on chip for a whole projection slot. Algorithm 2 builds them at launch from the $d(d + 1 + m)$ coefficients and then traverses the groups once, so m enters the launch and not the traversal.

Algorithm 2. Fused traversal of one projection slot

Input: coefficients Θ, Γ; latent weights $\{u_g\}$ over the G groups of the slot; upstream η_g on the backward pass.

Resident: the basis F, V_m; the derivative matrix $\mathcal{D}_q$ of Proposition 5.

At launch, once per decoder.

1. Build $C = F\Theta + V_m\Gamma$ by Eq. 11, and build J by contracting C along each coordinate with $\mathcal{D}_q$.
2. Zero the Kd accumulators of G_C in `fp32` (Remark 8).

Forward, per group of a contiguous block.

3. Round and clip u_g by Eq. 10 to obtain z_g.
4. Accumulate the label by Horner's rule over the d coordinates, $\iota \leftarrow q\,\iota + z_{g,j}$, in one register rather than an index tensor over the layer.
5. Decode $\hat{w}_g$ from row ι of C.

Backward, same blocking.

6. Recover ι, by recomputation or by reading it back.
7. Accumulate η_g into row ι of G_C, which is Eq. 13.
8. Contract the same η_g against the block of J at row ι and scale by $S(u_g)$, which is Eq. 14.

On completion. Form $\nabla_\Theta L = F^\top G_C$ and $\nabla_\Gamma L = V_m^\top G_C$.

Output: $\hat{W}$ on the forward pass; $\nabla_\Theta L$, $\nabla_\Gamma L$ and $\widetilde{\nabla}_{u_g} L$ on the backward pass.

Neither the per-group Jacobian of Eq. 14 nor the gathered codebook of Eq. 11 is written out, so the only tensors of the size of the layer are the ones the surrounding autograd graph already requires.

Remark 8 (The reduction must accumulate in higher precision). Eq. 13 pools every group of a slot into Kd accumulators. For the main model that is order 10^8 terms per slot spread over 10^3 accumulators, so each one receives order 10^6 additions. Accumulating those in the `bf16` of the training dtype is not adequate: the running sum grows until its unit in the last place exceeds the individual terms, and the tail of the reduction is then discarded. The accumulators are therefore kept in `fp32` regardless of the dtype of the activations, which costs Kd words per decoder and nothing per group. This is a condition on the implementation and not a qualification of Appendix B.7: the gradient is exact at fixed codes as a statement about the derivative, and the precision of the reduction is what makes the computed value match it.

D Experimental Details

The quantized set follows the baselines: in all 40 layers we quantize the routed experts, both the fused `gate_up` and the `down` projection, together with the three projections of the shared expert, which is 32.3B of the 35B weights. Attention, the router, the embeddings and the language-modeling head are left in `bf16`. A decoder serves a projection slot, so the 256 routed experts of one slot share one, giving $40 \times (2 + 3) = 200$ decoders for the model. The trainable tensors are the latent weights $\{u_g\}$, the diagonal scales $\sigma_{\text{in}}, \sigma_{\text{out}}$ of Eq. 9, the normalization weights, and the decoder coefficients Θ, Γ; the basis $[F, V_m]$ is a buffer and is not trained, since it is what fixes the level. This set is that of the VQ-QAT baselines together with the coefficients: the latent weights, the scales and the normalization weights are trained for every method and with the same schedule, so the coefficients are the only tensor Vodka adds to what LC-QAT optimizes, and the comparison in Tab. 1 holds the rest fixed. We confirm this set by comparing every tensor of a trained checkpoint against its initialization: the coefficients and the scales move, and the tensors listed as frozen are bit-identical. Since a configuration that silently froze the coefficients would still produce a falling loss curve and

a plausible final accuracy, this check is what establishes that the levels of Sec. 4.5 are what the runs actually differ in.

Both gradient paths are enabled throughout training: the coefficient path of Eq. 13, which is the exact derivative at fixed indices (Appendix B.7), and the index path of Eq. 14, whose Jacobian is the analytic derivative of the decoder (Appendix B.8) composed with the DGE surrogate for the rounding. We do not attribute the reported gains to one path or the other, and a decomposition of the two is left to future work.

Every number we report is a mean over runs that differ only in the random seed, which enters the data order and the initialization of the trainable scales, the quantized weights and the basis being fixed by the PTQ stage of Appendix C.2. Vodka and the baselines are averaged over the same seeds, so the comparison in Tab. 1 holds the seed set fixed as well as the data and the schedule. The budget of 375 steps is where our runs stop improving, as the last quarter of Fig. 5 shows, and it is the same budget for every method; since LC-QAT is still rising at the end of it, the reported margins are what a converged Vodka holds over a baseline that has not yet converged, and a longer schedule would narrow them rather than widen them. The remaining hyper-parameters are as shown in Tab. 4.

Table 4: QAT configuration. Identical for Vodka and for every baseline unless says otherwise.

Setting	**Value**
Optimizer	AdamW, $\beta = (0.9,\ 0.999)$
Peak learning rate	1e-4
Schedule	warmup-stable-decay, 3/67/30
Weight decay	0
Gradient clipping	1.0
Gradient accumulation	2
Effective batch	16
Context length	64K
Steps	375
Group size d	4
Codebook size K	256
Bit width b	2

E THE FREE END IS A TRAINABLE TABLE

Sec. 4.5 reports the free end as a jointly trained unconstrained codebook rather than as an approximation of one, and reports $m = 27$ in the main table even though accuracy rises to that end. Both statements are settled here, the first by the construction and the second by what the family costs at the top.

Proposition 6 (The free end reparameterizes a table)**.** *At $m = M$ the map $\Xi \mapsto [F, V_M]\,\Xi$ is a linear bijection of $\mathbb{R}^{K\times d}$ onto itself. Its coefficient count is $d(d+1+M) = dK$, the number of entries of a stored codebook, and under the orthonormal basis of Remark 4 the two parameterizations differ by an orthogonal change of coordinates.*

Proof. By Theorem 1(b) the columns of $[F, V_M]$ are K linearly independent vectors of $\mathbb{R}^K$, so the matrix is square and invertible and the map is a bijection, which is Eq. 18 at $m = M$. Since $d + 1 + M = K$, the count is dK. Orthonormalizing $[F, V_M]$ gives a B with $B^\top B = I$ spanning the same space, and $C = B\Xi'$ with $\Xi' = B^\top[F, V_M]\Xi$; the two parameterizations therefore differ by an orthogonal map, and B is a basis of $\mathbb{R}^K$, so C ranges over every table as Ξ' does. □

Three consequences follow, and together they are why the level we report is not the top of the axis. First, by Proposition 6 the free end reaches the same set of codebooks as a stored table, with the same number of trainable numbers, so the right end of Fig. 6 is a codebook trained jointly with the weights against the task loss and not a constrained stand-in for one. The two differ only in the basis in which a step is taken, which a per-coordinate optimizer preconditions differently but which leaves the reachable set and the parameter count identical. Second, the family economizes on decoder parameters only below the top: at $m = M$ the coefficients cost exactly what a table costs,

Table 5: **Comparing the Polynomial with our Vodka at 2 bits.**

	Agent		IF
Method	LiveCodeBench	BFCL	IFEval
Qwen3.6-35B-A3B			
FP16	81.42	75.99	85.40
Vodka(m=27)	75.55	72.58	83.18
P_2	72.48	72.18	81.73
P_3	72.03	71.98	80.68
P_5	71.74	72.01	80.57

so the reduction, a factor of $1024/20$ at the affine end and $1024/128$ at $m = 27$, exists for $m < M$ and vanishes at M. Third, the pooling behind Eq. 13 narrows as m grows: each coefficient of a decoder is estimated from all the groups that decoder serves, of order 10^8 for a MoE slot, spread over $d(d + 1 + m)$ coefficients, so at $m = 27$ that is 128 numbers and at $m = M$ it returns to the 1024 of a table (Remark 6). The coefficients are also not free statistically, as the polynomial family of Appendix F shows at a matched budget: a sevenfold larger coefficient count there costs 0.74 points on LiveCodeBench rather than buying any. Reporting $m = 27$ is therefore a choice of the smallest level that captures the available gain, and Sec. 4.5 records that every level from $m = 27$ upward is ahead of all baselines on the three complex tasks, so the comparison in Tab. 1 does not rest on which of them we report.

Two readings of Fig. 6 follow from the same construction. The affine end coincides with LC-QAT, whose map is fixed at initialization (Appendix A.2), so the first segment of the curve moves both the level and the trainability of the map, while the segments above it move the level alone. The two legs are entangled by construction rather than by omission: a trainable decoder has little to gain at the affine end, where Table 3 puts a full affine map within 0.1% of an optimal 2-bit scalar quantizer in distortion, so the capacity of Sec. 3.1 is what makes training the map worth anything and the gradient of Eq. 13 is what makes that capacity reachable.

F Comparing with Polynomial Function

Sec. 3.1 selects the non-affine directions by curvature, which makes Eq. 7 a kernel expansion truncated by complexity, and Appendix B.2 establishes that some ranking is needed but not that this one is best. The natural competitor is the other truncation of the same space, by polynomial degree. This appendix reports that comparison at a matched coefficient budget.

The polynomial family. Fix the total degree t and let Φ_t collect the monomials $\prod_j z_j^{a_j}$ with $|a| \le t$, evaluated at the grid points of $\mathcal{Z}$. Writing P_t for the resulting decoder,

$$\mathcal{P}_t = \{\Phi_t \Lambda\}, \qquad \Lambda \in \mathbb{R}^{n_t \times d}, \qquad n_t = \Big|\{a \in \{0, \dots, q-1\}^d : |a| \le t\}\Big|, \tag{23}$$

which is a subspace of $\mathbb{R}^{K \times d}$ containing $\mathrm{col}(F)$ at every $t \ge 1$, so $\mathcal{P}_t$ shares the two endpoints of Eq. 7: $\mathcal{P}_1$ is the affine design and $\mathcal{P}_{d(q-1)}$ is the unconstrained codebook. The two families therefore differ only in which directions they admit first, which isolates the comparison.

As shown in Tab. 5, raising the degree t of Eq. 23 does not yield expressivity. From P_2 to P_5 the coefficient budget grows sevenfold, from 60 to 424, yet LiveCodeBench falls from 72.48 to 71.74 and IFEval from 81.73 to 80.57, while BFCL stays within 0.20 of itself throughout. The directions that a degree truncation admits first are therefore not the ones the task rewards, and admitting more of them only expends side information. Against P_3, the closest budget to it, Vodka at $m = 27$ uses fewer coefficients, 128 against 140, and is ahead by 3.52 on LiveCodeBench, 2.50 on IFEval and 0.60 on BFCL; against the best polynomial on each task, which is P_2 throughout, it is ahead by 3.07, 1.45 and 0.40. What separates the two families at a matched budget is which directions the basis admits, not how many.

G LIMITATIONS

Activation quantization Our setting is weight-only quantization, applied to both dense and mixture-of-experts models, and activations are kept at their original precision throughout. Since the criterion is orthogonal to how activations are represented, extending the construction to the joint weight-activation setting is the most practical direction we leave open.

Composition with other techniques We report the proposed axis in isolation, so how it performs in combination with the rest of the pipeline remains open. Techniques such as on-policy distillation and speculative decoding are orthogonal to Vodka rather than alternatives to it, and how much they compose is worth establishing.

The rounding operation The gradient to the decoder coefficients is exact at fixed indices and its Jacobian is analytic, while the rounding itself is still handled by the DGE surrogate of Eq. 14. Making the pipeline differentiable throughout is, we believe, necessary for further gains, so whether this estimator can be replaced or instead absorbed into a theoretical account is worth investigating.

H SOCIAL IMPACT

Positive This work reduces the memory required to store and serve large language models by representing weights at two bits each. A smaller footprint allows a given model run on less capable hardware, which broadens who can deploy it without massive GPU, and reduces the energy cost where memory bandwidth dominates. The method is also initialized from a post-training quantized model and trained on a small dataset, so its own cost is modest relative to pretraining.

Negative Quantization is indifferent to which model it compresses, so our method can be applied to models whose deployment is undesirable. Safety training is carried by the same weights being compressed, and our benchmarks measure capability rather than alignment, so a deployment decision should rest on a safety evaluation of the quantized model as well.