# Leech Lattice Vector Quantization for Efficient LLM Compression

**Tycho F. A. van der Ouderaa** **Mart van Baalen** **Paul Whatmough** **Markus Nagel**
Qualcomm AI Research*

## Abstract

Scalar quantization of large language models (LLMs) is fundamentally limited by information-theoretic bounds. While vector quantization (VQ) overcomes these limits by encoding blocks of parameters jointly, practical implementations must avoid the need for expensive lookup mechanisms or other explicit codebook storage. Lattice approaches address this through highly structured and dense packing. This paper explores the Leech lattice, which, with its optimal sphere packing and kissing configurations at 24 dimensions, is the highest dimensional lattice known with such optimal properties. To make the Leech lattice usable for LLM quantization, we extend an existing search algorithm based on the extended Golay code construction, to i) support indexing, enabling conversion to and from bitstrings without materializing the codebook, ii) allow angular search over union of Leech lattice shells, iii) propose fully-parallelisable dequantization kernel. Lastly, we provide a geometric reinterpretation of combining shape–gain quantization with GPTQ-style Hessian corrections: the standard scale-correction step of shape–gain acts as a *retraction* onto a product of spheres, yielding a *Spherical GPTQ* primarily acting on directions. We find that low-angular-distortion LLVQ reduces sensitivity to Hadamard/rotation preprocessing, and enables a strong *Hadamard-free* PTQ in practice. LLVQ delivers state-of-the-art LLM quantization performance, outperforming recent methods such as Quip#, QTIP, and PVQ. The results highlight the effectiveness of high-dimensional lattices for scalable, theoretically grounded model compression.

## 1 Introduction

Quantization is a critical technique for compressing large language models (LLMs). Traditionally, this has been approached through scalar quantization, where individual weights are represented using fewer bits. While simple and widely adopted, classical results in rate–distortion theory (originating with Shannon) show that memoryless mappings are, in general, suboptimal: achieving the optimal distortion at a given rate typically requires coding over blocks rather than symbol-by-symbol mappings (Shannon, 1948; Shannon et al., 1959). Perhaps surprisingly, this even holds for completely independent and isotropically distributed sources, such as Gaussian vectors, where block coding strictly outperforms scalar methods in the rate-distortion trade-off (Gray & Neuhoff, 2002). Consequently, scalar quantization is fundamentally limited when targeting aggressive compression without significant accuracy loss.

To overcome these limitations, vector quantization (VQ) (Gersho & Gray, 2012) encodes *blocks* of weights jointly. Concretely, a $d$-dimensional block represented by a $b$-bit index selects one codeword from a set of size $2^b$, yielding an average rate of $b/d$ bits per weight. From a deep-learning practitioner's perspective, this is akin to assigning a dedicated *dtype* to an entire block of weights rather than to each scalar entry: the block is stored as a single compact integer index instead of many

*Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.
Correspondence to: {touderaa,mart,pwhatmou,markusn}@qti.qualcomm.com.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

independent scalars. A *naive* realization of this idea is to materialize the codebook explicitly and perform nearest-neighbor lookup among its $2^b$ high-dimensional codewords. *GPTVQ* (Van Baalen et al., 2024) demonstrates that such unstructured VQ can be applied to LLMs; however, the explicit-table approach scales poorly with dimension $d$, because both storage and lookup costs grow exponentially with the vector dimensionality.

This underscores a limitation of the classical theory: Shannon's results establish the optimality of block coding in principle, yet offer no guidance on practical implementations. The key challenge, therefore, is to design VQ schemes that avoid an explicitly stored codebook and exhaustive nearest-neighbor search, while still admitting large representable sets. A considerable body of research has explored how to impose structure on vector quantizers to avoid the prohibitive cost of unstructured nearest-neighbor search. Recent work on LLM quantization have exploited such structured representations, such as Quip# (Tseng et al., 2024a), which uses the $E_8$ lattice; QTIP (Tseng et al., 2024b), which employs trellis-based constructions to scale to higher dimensions; and PVQ (van der Ouderaa et al., 2024), which uses flexible high-dimensional pyramids as quantization rules.

The use of lattices for quantizing LLM weights was recently popularized by QuIP#, which employs the $E_8$ lattice in eight dimensions. Together with the Leech lattice, these occupy a distinguished place in mathematics: $E_8$ achieves optimal sphere packing in dimension 8, while the Leech lattice does so in dimension 24. These are the highest dimensions in which optimal lattice packings are known, proven only recently through breakthroughs in harmonic analysis and modular forms, work that earned Maryna Viazovska the 2022 Fields Medal (International Mathematical Union, 2022).

Practical scalable VQ methods must deliver good rate–distortion performance on target distributions while enabling fast quantization and dequantization. To do so, it should support: (i) efficient nearest-neighbour search on the (implicit) quantization grid, (ii) a compact integer or bitstring representation of each quantized vector, and (iii) a fast mapping from this index back to its corresponding representative vector. All without explicitly storing the codebook in memory. Lastly, we find that matching the original norm in shape–gain can be viewed as a *retraction* onto a constant-norm manifold, which when paired with GPTQ, suggests a natural *Spherical GPTQ* interpretation in which error-feedback updates act primarily on directions rather than magnitudes.

Our proposed method, Leech Lattice Vector Quantization (LLVQ), a codebook-free quantization framework built on the structure of the Leech lattice, satisfies all these criteria. LLVQ leverages the geometric structure of the Leech lattice to provide state-of-the-art vector quantization for large language models, delivering superior accuracy–model-size trade-offs. Specifically, our contributions are as follows:

1. Extend codebook-free nearest neighbour algorithm of Adoul & Barth (1988) on the Leech lattice to allow *indexing*, enabling conversion to and from indices/bitstrings without materializing a codebook, required for actual compressed representations.
2. Extend codebook-free nearest neighbour algorithm on the Leech lattice to allow *angular search over union of Leech lattice shells*, required for shape-gain Leech lattice quantization.
3. Propose a fully parallelizable kernel for fast dequantization of spherically bounded Leech lattice points using fast modulo arithmetic.
4. Spherical GPTQ: We show that the standard scale correction used in shape–gain quantization, when interleaved with GPTQ-style Hessian error feedback, is naturally interpreted as a (hyper)sphere-constrained variant of GPTQ on a product of spheres. We find that this is very effective for directional LLVQ codes, and allows for a strong *Hadamard-free* PTQ.
5. Scientific findings on Gaussian source: establish that union of shells achieves lower angular distortion than using single shells (App. G), and demonstrate that Leech shape-gain codes can improve signal-to-noise over spherical shaping (App. H).

LLVQ achieves state-of-the-art **2 bit per weight** PTQ quantization of large language models, consistently surpassing AQLM, QuIP# and QTIP across perplexity metrics and downstream tasks on various common LLM models, such as from the Llama-2, Llama-3, Ministral-3 and Qwen-v3 architectures. In addition, the algorithm naturally allow a wide variety of bit-widths (unlike competing approaches, often relyng on techniques such as residual vector quantization RVQ (Tseng et al., 2024a) to increase bitrates). Further paired with Spherical GPTQ, LLVQ remains competitive even without Hadamard rotations, suggesting a practical route to rotation-free, low-latency PTQ. Altogether, the findings highlight high-dimensional lattice methods as a powerful path toward scalable, low-distortion compression of modern neural networks.

## 2 The Leech Lattice

In this work, we use the *Leech lattice* $\Lambda_{24}$ as the foundation for our quantization codebook. Its exceptional symmetry, high minimum distance, and rich shell structure make it uniquely suited for constructing efficient, low-distortion spherical codes with fast encoding and decoding procedures.

### 2.1 Definition and Properties

A lattice in $\mathbb{R}^n$ is a discrete additive subgroup generated by an integer linear combination of basis vectors $b_1, \ldots, b_n$: $\Lambda = \{\sum_{i=1}^{n} k_i b_i \,|\, k_i \in \mathbb{Z}\}$. The Leech lattice $\Lambda_{24}$ is a renowned 24-dimensional lattice, owing to its numerous optimal geometric properties. It achieves the densest and provably optimal sphere packing in 24 dimensions and exhibits a massive automorphism group (of size roughly $8.3 \times 10^{18}$), reflecting a high degree of symmetry. Dense sphere packing is often cited as a desirable property for quantization because, under the standard high-rate assumption that the source is approximately uniform at the scale of the Voronoi cell. In addition, the Voronoi cell of $\Lambda_{24}$ has an exceptionally low normalized second moment, further minimizing distortion under these second-order (quadratic) approximations. We note that normalizing Leech lattice vectors also produce remarkably uniform spherical codes, allowing high-performant finite bitrate quantization not just in Euclidean, but also in spherical geometry (e.g. shape-gain quantization, see subsection 2.2).

Several equivalent constructions of $\Lambda_{24}$ exist (see Conway & Sloane, 2013 for an authoritative reference). We adopt the formulation based on the extended binary Golay code, which provides an explicit coordinate representation of the lattice. The proposed neighbour search and indexing procedures operate directly in $\mathbb{R}^{24}$ while internally exploiting the Golay-derived structure. This avoids having to enumerate or materialize an astronomically large set of lattice points.

### 2.2 Finite Codes from Lattice Shells

Lattices are infinite, highly structured point sets in $\mathbb{R}^n$. While they provide a rich mathematical structure, quantization requires a finite subset, where the codebook must consist of a limited number of points so that each codeword can be represented using a fixed number of bits (just as integers form an infinite set, but any integer-based datatype must restrict its representable range). To transform a lattice into a practical finite representation, it is common to use *spherical shaping*, in which we retain only those lattice points whose Euclidean norm does not exceed a chosen radius.

A useful property of many lattices, including the Leech lattice, is that their points naturally partition into *shells*: sets of lattice points lying at the same squared Euclidean norm. Consequently, these shells occur at discrete radii (of integer squared norm) and exhibit rich combinatorial structure that can be exploited for fast search and efficient enumeration. We now formalize the partitioning of the Leech lattice into shells. For each integer $m \geq 2$, we define the $m$-th shell as

$$\text{Shell}(m) = \{v \in \Lambda_{24} \quad | \quad \|v\|_2^2 = 2m\}. \tag{1}$$

The minimal squared norm of $\Lambda_{24}$ is 4 (for $m = 2$). The disjoint shells partition the full lattice:

$$\Lambda_{24} = \bigcup_{m=2}^{\infty} \text{Shell}(m), \quad \text{and define ball-cuts } \Lambda_{24}(M) = \bigcup_{m=2}^{M} \text{Shell}(m). \tag{2}$$

up to squared norm $M = 2m$. Since shells are disjoint, the cardinality is a cumulative sum:

$$N(M) = |\Lambda_{24}(M)| = \sum_{m=2}^{M} n(m), \quad \text{with } n(m) = |\text{Shell}(m)| \tag{3}$$

These shell sizes are the coefficients of the Leech lattice theta series (Leech, 1967), but for our purposes they can be treated as fixed combinatorial quantities that can be precomputed or tabulated. Since $M$ controls the size of the bounded lattice subset, it can be thought of as a parameter that governs both the bitrate and the effective fineness of the quantization grid. Table 1 reports the number of points per shell $n(m)$,

Table 1: Shell structure of the Leech lattice $\Lambda_{24}$.

| $m$ | **Radius** $\sqrt{2m}$ | **Shell cardinality** $n(m)$ | **Cumulative count** $N(m)$ | **Bits / dim** |
|---|---|---|---|---|
| 2 | 2 | 196,560 | 196,560 | 0.75 |
| 3 | $\sqrt{6}$ | 16,773,120 | 16,969,680 | 1.042 |
| 4 | $2\sqrt{2}$ | 398,034,000 | 415,003,680 | 1.208 |
| 5 | $\sqrt{10}$ | 4,629,381,120 | 5,044,384,800 | 1.375 |
| . | | | | |
| **13** | $\sqrt{26}$ | 16,993,109,532,672 | 280,974,212,784,720 | **2.000** |
| . | | | | |
| 19 | $\sqrt{38}$ | 1,104,550,081,689,600 | 23,546,209,100,646,960 | 2.292 |

cumulative number of points $N(M)$ together with the required bits per dimension $\lceil \log_2 N(M) \rceil / 24$ needed to store an index identifying lattice points within the bounded set. An alternative way to construct a finite subset from a lattice, beyond spherical shaping, is the *shape–gain* approach (Sabin & Gray, 1982; Hamkins & Zeger, 2002), in which vector magnitudes and directions are quantized separately. The magnitude is handled by a standard scalar quantizer, while the direction is mapped to a *spherical code*, i.e., a finite set of points on the high-dimensional unit sphere. Such spherical codes can be constructed by normalizing all lattice points in one or more shells. For shape–gain, it is not so much the packing density of a lattice, but rather the uniformity of the constructed spherical code that determines the quantization performance. Interestingly, the Leech lattice excels on both fronts: it has exceptionally high packing density (Conway & Sloane, 2013), and its shells yield remarkably uniform spherical codes. In App. H, we compare spherical shaping with shape–gain quantization for Gaussian sources using the Leech lattice and find that both perform very well, with shape–gain achieving slightly improved rate–distortion in our setting, motivating its use.

One may also wonder whether it is preferable to form a spherical code using individual shells or using a cumulative union of shells. We investigate this in App. G and find that using the union of shells up to $m$ yields a more uniform spherical code, by a lower empirical angular error to the closest point per bitwidth, compared to using individual shells. We conjecture that this trend persists, and that cumulative unions of shells achieve strictly lower distortion than the rate–distortion at infinite rate limits. This motivated our use of cumulative shell unions in shape-gain.

### 2.3 Extended Golay Code construction of $\Lambda_{24}$

For efficient search in $\Lambda_{24}$, we follow (Adoul & Barth, 1988) and use the classical construction of the Leech lattice from the extended Golay code (Conway & Sloane, 2013). This representation organizes the lattice as an implicit hierarchy of integer vectors. The extended Golay code $\mathcal{G}_{24} \subset \mathbb{F}_2^{24}$ is the unique binary code of size $4096$, whose nonzero codewords have Hamming weights in $\{8, 12, 16, 24\}$.

Table 2: Composition of lattice vectors grouped by equivalence class. For each class, the columns indicate the multiplicities of coordinates equal to $\pm 7, \pm 6, \ldots, \pm 0$, describing the canonical coordinate pattern of vectors in that class. Shell cardinalities increase with $m$.

| m | parity | count | ±8 | ±7 | ±6 | ±5 | ±4 | ±3 | ±2 | ±1 | ±0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | even | 1104 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 22 |
| | even | 97152 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 16 |
| | odd | 98304 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 23 | 0 |
| 3 | even | 3108864 | 0 | 0 | 0 | 0 | 1 | 0 | 8 | 0 | 15 |
| | even | 5275648 | 0 | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 12 |
| | odd | 98304 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 23 | 0 |
| | odd | 8290304 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 21 | 0 |
| 4 | even | 170016 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 20 |
| | even | 48 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 23 |
| | even | 46632960 | 0 | 0 | 0 | 0 | 2 | 0 | 8 | 0 | 14 |
| | even | 777216 | 0 | 0 | 1 | 0 | 0 | 0 | 7 | 0 | 16 |
| | even | 126615552 | 0 | 0 | 0 | 0 | 1 | 0 | 12 | 0 | 11 |
| | even | 24870912 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 8 |
| | odd | 24870912 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 21 | 0 |
| | odd | 174096384 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 19 | 0 |

Vectors in $\Lambda_{24}$ are first grouped by *shell*, and within shells, vectors share the same unordered multiset of absolute values form a *class*, represented by a canonical leader up to admissible permutations and sign choices. Each class is intrinsically *even* or *odd* (whether it is a subset of $L^{\text{even}}$ or $L^{\text{odd}}$, defined shortly), determining the permissible coordinate permutations and sign assignments, dictated by the underlying Golay structure. Our proposed indexing scheme follows the same hierarchy. For example, the first shell contains $196{,}560$ lattice points, of which the first $1{,}104$ belong to the first class, consisting of vectors with two coordinates equal to 4 and twenty-two equal to 0 (see Table 2). The (nearest neighbour) quantization- and dequantization algorithms exploit the hierarchy to avoid explicit enumeration of lattice points.

**Integer-coordinate formulation.** The Leech lattice can be defined as the union of:

$$L^{\text{int}} = L^{\text{even}} \cup L^{\text{odd}}, \qquad \Lambda_{24} = \frac{1}{\sqrt{8}} L^{\text{int}} \subset \mathbb{R}^{24}, \tag{4}$$

a scaled union of two integer-valued sets which we call the *even* and *odd* cosets,

$$L^{\text{even}} = \left\{ x \in \mathbb{Z}^{24} \;\middle|\; \begin{array}{l} \text{(i) } x_i \equiv 0 \pmod 2 \\ \text{(ii) } (x/2) \bmod 2 \in \mathcal{G}_{24} \\ \text{(iii) } \sum_i x_i \equiv 0 \pmod 8 \end{array} \right\} \quad L^{\text{odd}} = \left\{ x \in \mathbb{Z}^{24} \;\middle|\; \begin{array}{l} \text{(i) } x_i \equiv 1 \pmod 2 \\ \text{(ii) } ((x-\mathbf{1})/2) \bmod 2 \in \mathcal{G}_{24} \\ \text{(iii) } \sum_i x_i \equiv 4 \pmod 8 \end{array} \right\} \tag{5}$$

where the roles of the three constraints are:

(i) *Parity constraint:* forces vector in even or odd coset.
(ii) *Golay constraint:* ensures mod-2 reductions of halved vectors match codewords of $\mathcal{G}_{24}$.
(iii) *Sum constraint:* enforces the global congruence needed for the resulting lattice to be even.

and we write $\mathbf{1} = (1, \ldots, 1) \in \mathbb{Z}^{24}$. With the $1/\sqrt{8}$ normalization, the lattice is even and unimodular.

### 2.4 Class Structure and Leaders

Each shell $\mathrm{Shell}(m)$ can be decomposed into different *classes*, where we define a class as a set of all lattice points obtainable from one another by the allowed permutations of coordinates and sign flips. Table 2 lists the classes in the first three shells of the Leech lattice, together with their cardinalities.

For convenience, we select a single representative from each equivalence class under coordinate permutations; typically this is the point whose coordinates have been reordered so that their absolute values appear in a fixed canonical order. We refer to this representative as the *leader*. In the integer embedding, these leaders admit a compact combinatorial description: up to permutation of coordinates, each class is characterized by a multiset

$$\{\, a_1^{p_1},\, a_2^{p_2},\, \ldots,\, a_k^{p_k} \,\}, \qquad \sum_i p_i = 24, \quad \text{where } a_i \in \tfrac{1}{\sqrt{8}}\mathbb{Z} \tag{6}$$

Absolute coordinates $a_i$ and exponents $p_i$ fully specify vector coordinate compositions of each class.

**Even and Odd Classes** Each class lies entirely in either $L^{\mathrm{even}}$ or $L^{\mathrm{odd}}$, and this determines how its nonzero coordinates may be placed and signed. A leader fixes the multiset of absolute coordinate values and whether the class is even or odd. What remains unspecified is how those fixed coordinate values are placed across the 24 positions and which sign patterns are allowed. Given a Golay codeword $c \in \mathcal{G}_{24}$, define

$$F_0(c)=\{\, i : c_i=0 \,\}, F_1(c)=\{\, i : c_i=1 \,\}. \tag{7}$$

For **even classes**, admissible placements arise only from Golay codewords $c$ whose Hamming weight matches the number of coordinates of the leader congruent to $2 \pmod 4$. For any such $c$, coordinates with $x_i \equiv 0 \pmod 4$ must lie in $F_0(c)$, and those with $x_i \equiv 2 \pmod 4$ must lie in $F_1(c)$, up to permutation. Signs of coordinates in $F_0(c)$ are unrestricted because the $0 \pmod 4$ congruence is invariant under sign flips. Signs in $F_1(c)$ are constrained only by the global condition $\sum_i x_i \equiv 0 \pmod 8$, which fixes the parity of the number of negative signs among the $F_1(c)$ entries. Thus, even classes admit only those placements induced by compatible Golay codewords and a correspondingly restricted set of sign patterns. For **odd classes**, *every* codeword $c \in \mathcal{G}_{24}$ yields a valid placement. The congruence conditions then determine the coordinate types uniquely: positions in $F_0(c)$ carry entries with $x_i \equiv 1 \pmod 4$, and positions in $F_1(c)$ carry entries with $x_i \equiv 3 \pmod 4$. Consequently, the sign pattern is fixed by these congruences (up to an overall sign flip).

### 2.5 Subclass Counts and Class Cardinality

The cardinality of each class follows directly from combinatorial coordinate placements, signs, and coordinate multiplicities. Fix a leader with condensed multiset $\{a_1^{p_1}, \ldots, a_k^{p_k}\}$. The number of lattice points in its class factorizes into: (a) the number $A$ of Golay codewords $c \in \mathcal{G}_{24}$ that yield an admissible placement for this leader (odd classes have $A = 4096$; even classes have $A \in \{1, 759, 2576, 759, 1\}$ depending on the required weight), (b) the number $2^B$ of admissible sign assignments consistent with parity and the global mod–8 sum constraint, (c) the multinomial factor $\frac{24!}{\prod_i p_i!}$ accounting for permutations of coordinates with identical absolute values, and (d) an additional divisor $\prod_j q_j!$ capturing permutations that act trivially because equal-valued coordinates fall within the same Golay subset $F_0(c)$ or $F_1(c)$. The class cardinality is $A \cdot 2^B \cdot \frac{24!}{\prod_{i=1}^k p_i!} \cdot \frac{1}{\prod_j q_j!}$..

## 3 Leech Lattice Vector Quantization (LLVQ)

We build upon the fast exact nearest neighbour search for single Leech lattice shells by Adoul & Barth (1988). This method generates candidates via leaders (the canonical absolute-value patterns) and uses Golay-derived placements, parity-constrained sign patterns, to rank dot products with the input. On a single shell, this dot-product ranking coincides with Euclidean ranking (see Sec. 3.1 below). We generalize and extend the algorithm in two important ways: (i) we enable nearest neighbour search over multiple $\Lambda_{24}(m)$ shells, where candidate norms vary and Euclidean vs. angular scoring no longer coincide; and allow support for both *Euclidean* scoring (for spherical shaping) and *angular* scoring (for shape–gain), discussed in §3.1; and (ii) we introduce a *bijective indexing mechanism* aligned with the Leech lattice hierarchy (shells, classes, and local symmetries), yielding compact integer codes and exact reconstruction through the dequantizer (see §3.2, §A). Finally, we describe *Spherical GPTQ*, a norm-preserving GPTQ variant for directional codes like LLVQ (§3.3).

### 3.1 Extending to Spherical Search in $\Lambda_{24}(m)$

In the original single-shell setting of Adoul & Barth (1988), all candidates share the same norm so Euclidean and angular distances induce the same ordering: $\|x-v\|^2=\|x\|^2+\|v\|^2-2\langle x,v\rangle$ (fixed $\|v\|$) Consequently, ranking by $-\langle x,v\rangle$ is equivalent to minimizing $\|x-v\|$. For LLVQ we generalize the search to multiple shells of $\Lambda_{24}(m)$. Since candidate norms vary across shells the Euclidean vs. angular equivalence no longer holds. We add support for two metrics: *Euclidean distance* (for spherical shaping), and *angular distance* via cosine similarity (for shape–gain). This can be implemented by normalizing both the input and candidates, $\hat{x}=x/\|x\|$ and $\hat{v}=v/\|v\|$ and maximizing $\langle\hat{x},\hat{v}\rangle$.

### 3.2 Indexing Scheme

While nearest neighbour algorithm by Adoul & Barth (1988) introduced an elegant exact search algorithm over leaders, placements, and signs, it does not provide a bijective indexing scheme, which is required for quantization-based compression. For compression, each lattice vector must correspond to a unique integer index (or bitstring), and this mapping must be efficiently invertible. We therefore extend the procedure to allow indexing of Leech lattice vectors aligned with its described hierarchical structure. First, shells are ordered by increasing radius: the first 196,560 indices correspond to the first shell, the next 16,773,120 indices correspond to the second shell, and so on. Within each shell, we assign consecutive index ranges to the classes (e.g., based on cardinality or any other fixed, consistent ordering). Inside each class, the remaining degrees of freedom: permutations, sign flips, or other symmetries, are indexed locally. The local class index is combined with the shell and class indices through standard index linearization (as used when flattening a multidimensional array into a 1-dimensional array). This yields a unique and invertible global index for every vector. The inverse mapping follows the usual unflattening procedure: integer division recovers the shell and class, and modulo recovers the index inside the class.

We index vectors according to the natural hierarchy of shells, classes, and intra-class ordering.

**(1) Shell level** Shells are ordered by increasing squared norm. Let $n(m) = |\mathrm{Shell}(m)|$ and $N(m) = \sum_{\ell\le m} n(\ell)$ be cumulative offsets. The global indices $\{0,\dots,N(2)-1\}$ enumerate $\mathrm{Shell}(2)$, the next $\{N(2),\dots,N(3)-1\}$ enumerate $\mathrm{Shell}(3)$, and so on.

**(2) Class level** Within shell $m$, we fix a deterministic total order over its classes (e.g., lexicographic on leaders, then parity, then a fixed tie-break) and assign to each class a contiguous subrange whose length equals its cardinality. Per shell, leaders, class sizes, and cumulative offsets are stored solely to support dequantization (shell/class lookup and vector reconstruction from indices).

**(3) Local symmetry level** Within each class, the local index is obtained by decomposing the integer into successive choices: first the Golay refinement, then the sign pattern, and finally the permutation coset. This is implemented by repeated modulo and integer-division operations.

### 3.3 Spherical GPTQ: A geometric interpretation of Scale-corrected Shape-Gain with GPTQ

When LLVQ is used in a shape–gain regime, the quantizer primarily determines a vector direction, while magnitudes are handled by an explicit gain quantizer. In practice, shape–gain pipelines are often coupled with a scale-correction step (App. E): after quantizing the direction, the reconstruction is rescaled to minimize Euclidean distance with the original vector.

We observe that, when this scale correction is interleaved with GPTQ-style Hessian error feedback, it admits a clean geometric interpretation: the gain reset is precisely a *retraction* onto a product of constant-norm manifolds (hypersphere). This yields a *Spherical GPTQ* viewpoint in which the greedy PTQ dynamics operate predominantly on directions. Concretely, given a block $\boldsymbol{w}$ and a candidate $\tilde{\boldsymbol{w}} = \mathcal{Q}(\boldsymbol{w})$, the gain reset can be written as

$$\hat{\boldsymbol{w}} \;=\; \Pi_{\|\boldsymbol{w}\|}(\tilde{\boldsymbol{w}}) \;:=\; \frac{\|\boldsymbol{w}\|_2}{\|\tilde{\boldsymbol{w}}\|_2}\,\tilde{\boldsymbol{w}}, \tag{8}$$

and GPTQ residuals are formed using $\hat{\boldsymbol{w}}$ rather than $\tilde{\boldsymbol{w}}$. We refer to this combined procedure as *Spherical GPTQ*. The interpretation also enables a clean ablation showing that the Leech codebook and norm-preserving corrections each improve PTQ in isolation and are complementary when combined (further details and ablations in App. I).

## 4 SQNR and Retention on Gaussian Source

We evaluate quantization performance by generating i.i.d. samples $w \sim \mathcal{N}(0,1)$ from a zero-mean, unit-variance Gaussian distribution, applying the quantization scheme at different bitrates, and measuring the resulting distortion. The distortion is computed as the mean squared error (MSE) between the original and quantized samples. For $n$ samples $\{w_i\}_{i=1}^n$ and quantizer $q(\cdot)$, we can obtain an unbiased estimate of the empirical quantization error per weight

$$\widehat{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n} \|w_i - q(w_i)\|_2^2 / D, D = \dim(w_i) \quad (9)$$

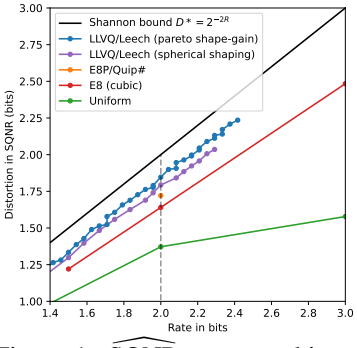



Figure 1: $\widehat{\text{SQNR}}_{\text{bits}}$ versus bitrate (bits/dim) on Gaussian source.

and the empirical $\widehat{\text{SQNR}}_{\text{bits}} = -\frac{1}{2}\log_2\big(\widehat{\text{MSE}}\big)$. For an ideal Gaussian source, the Shannon rate–distortion function gives the minimum achievable MSE at rate $R$ as $\text{MSE}^*(R) = \sigma^2 2^{-2R}$. For our unit-variance source ($\sigma^2 = 1$), this becomes $\text{MSE}^*(R) = 2^{-2R}$. Using the same convention for SQNR in bits, the optimal (Shannon) SQNR is $\text{SQNR}^*_{\text{bits}}(R) = -\frac{1}{2}\log_2\big(2^{-2R}\big) = R$. Thus, following this convention, the Shannon limit for a Gaussian source corresponds to the straight line $\text{SQNR}^*_{\text{bits}}(R) = R$ (i.e., a $y = x$ line) in the SQNR–versus–rate plot. To convert to the other common base-10 decibel (dB) unit, multiply $\text{SQNR}_{\text{bits}}$ with $20\log_{10}(2) \approx 6.0206$.

Due to structural constraints, practical quantizers inevitably fall below the theoretical limit. Figure 1 shows SQNR versus bitrate behavior on an idealistic Gaussian source. Among the evaluated methods, our LLVQ construction attains the highest empirical SQNR at each tested bitrate, consistently outperforming existing approaches and tracking the Shannon bound more closely than the baselines.

We quantify closeness to the Shannon limit by measuring *retention*: $\text{Ret}(\%) = \frac{\widehat{\text{SQNR}}_{\text{bits}}}{\text{SQNR}^*_{\text{bits}}(R)} \times 100$ with conveniently $\text{SQNR}^*_{\text{bits}}(R) = R$, under our convention of measuring SQNR in bits.

Table 3: Gaussian information retention at 2 bits/dim.

| Method | Dim | $\widehat{\text{MSE}}\downarrow$ | $\widehat{\text{SQNR}}$ (bits) $\uparrow$ | Ret (%) $\uparrow$ |
|---|---|---|---|---|
| Uniform | 1 | 0.15 | 1.37 | 69 |
| Lloyd-Max | 1 | 0.12 | 1.53 | 77 |
| E8 (cubic) | 8 | 0.103 | 1.64 | 82.0 |
| LLVQ/Leech [spherical shaping] (**Ours**) | 24 | **0.084** | **1.79** | **89.4** |
| LLVQ/Leech [shape-gain] (**Ours**) | 24 | **0.078** | **1.84** | **92.1** |
| Theoretical limit | - | 0.0625 | 2 | 100 |

As summarized in Table 3, the empirical SQNR and retention scores at $R = 2$ bits/dim reveal clear gaps in performance across quantizers. Uniform quantization performs worst, reflecting its known suboptimality in Gaussian sources, while structured lattice-based schemes such as cubically bounded E8 and E8P/Quip # achieve notably higher retention. Our LLVQ constructions achieve the best performance, achieving MSE and SQNR closest to the Shannon limit. This shows that LLVQ uses the available bitrate more efficiently, yielding substantially lower distortion on Gaussian inputs than existing lattice-based approaches.

## 5 LLM Quantization Results

For empirical results, we compute (GPTQ-style) layer-wise Hessians on 6,100 sequences from DCLM-edu (Li et al., 2024; Allal et al., 2025), matching the calibration set size used in prior work (Tseng et al., 2024a). When finetuning is applied, we update only the input scales, scalars shared across rows, which add a negligible number of bits per weight and relatively little risk of overfitting.

### 5.1 PTQ Results (Own Pipeline)

We begin by evaluating post-training quantization (PTQ) using a unified quantization pipeline enabling a strict apples-to-apples comparison across methods. Table 10 summarizes PTQ results on Llama-2 7B, Llama-3 8B, Ministral-3 8B, and Qwen-v3 4B and 8B under this setup. The gap between scalar baselines (RTN, GPTQ (Frantar et al., 2022), Quarot (Ashkboos et al., 2024)) and higher-dimensional VQ methods is immediately evident: naive 2-bit RTN yields extremely high perplexity and severe task degradation, while GPTQ and Quarot improve stability through Hessian curvature and Hadamard rotations, yet remain constrained by the limited representational capacity of 1D quantization.

Table 4: Comparison with wider set of methods for quantizing Llama-v2 7B model using different quantization methods, evaluated by wikitext-2 (Wiki) perplexity at 4096 context length and downstream task performance (CSR and MMLU), comparing results in literature. LLVQ consistently outperforms standard vector quantization approaches in performance against bits per parameter (Bits).

| | | | | | Llama-2 7B | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Details** (from literature) | **Finetuned** | **LM Eval** | **BPW** | **Wiki ↓** | **Arc-C ↑** | **Arc-E ↑** | **BoolQ ↑** | **Winogrande ↑** | **Hellaswag ↑** | **PiQA ↑** |
| Baseline | (**Ours**) | - | acc | 16 | 5.11 | **43.2** | **75.6** | **79.3** | **69.9** | **57.1** | **78.1** |
| Quip# | Tables 4 & 10 of Quip# paper | No | acc/acc_norm | 2 | 8.22 | 32.5 | 42.8 | 62.3 | 62.4 | - | 71.2 |
| LLVQ (**Ours**) | [spherical shaping] | No | acc | 2 | 7.61 | 34.7 | 69.3 | 67.5 | 64.6 | 46.6 | 73.5 |
| LLVQ (**Ours**) | [shape-gain, 2 bit gain] | No | acc | 2 | **6.83** | **35.5** | **69.8** | **73.0** | **66.9** | **49.7** | **75.2** |
| Quip | Table 3 of Quip# paper | Yes | acc | 2 | - | 19.4 | 26.0 | 54.6 | 51.8 | - | - |
| OmniQ | Table 3 of Quip# paper | Yes | acc | 2 | - | 21.6 | 35.2 | 57.5 | 51.5 | - | - |
| AQLM | Tables 5 & 6 of QTIP paper | Yes | not reported | 2.07 | 6.93 | 32.8 | 63.7 | 74.8 | 65.7 | - | - |
| Quip# | Tables 5 & 6 of QTIP paper | Yes | not reported | 2 | 6.19 | 35.2 | 65.3 | 75.4 | 64.9 | - | - |
| QTIP | Tables 5 & 6 of QTIP paper | Yes | not reported | 2 | 5.86 | 35.7 | 65.6 | **75.9** | 64.7 | - | - |
| PV-tuning | Table 8 of PV-tuning paper | Yes | acc | 2 | 5.84 | 38.4 | 71.2 | - | **66.7** | 53.5 | 77.0 |
| LLVQ (**Ours**) | [spherical shaping] | Yes | acc | 2 | **5.60** | **40.6** | **72.9** | 70.9 | 65.1 | 52.5 | 75.5 |
| LLVQ (**Ours**) | [shape-gain, 2 bit gain] | Yes | acc | 2 | **5.48** | **39.8** | **72.9** | 75.3 | 66.3 | **54.1** | **77.1** |

Higher-dimensional approaches, such as Quip#, substantially reduce this gap. However, our LLVQ method, employing Leech-lattice-based 24-dimensional vector quantization, consistently achieves the strongest 2-bit performance across all metrics. Wikitext perplexity, MMLU accuracy, and CSR all show clear gains over Quip#, demonstrating that high-dimensional structured lattices provide markedly more efficient weight-space packing than both scalar and E8-based quantizers.

### 5.2 Spherical GPTQ and Hadamard-Free PTQ

Following prior work (Ashkboos et al., 2024; Chee et al., 2023), we consider randomized Hadamard rotations on layer inputs/outputs, which often improve quantization but may require online application when not fusible into weights, increasing latency. We additionally evaluate *Spherical GPTQ*, a norm-preserving GPTQ variant obtained by applying the standard shape–gain scale-corrections within the GPTQ loop (see Sec. I). Empirically, we find Spherical GPTQ substantially reduces the reliance on Hadamard/rotation preprocessing, enabling strong Hadamard-free (or Hadamard-reduced) PTQ for LLVQ (Table 5).

Table 5: Sherical GPTQ interpretation and Hadamard-free PTQ: ablation with and without Hadamard rotations, evaluated on Wikitext-2 perplexity (PPL) and downstream tasks (CSR and MMLU). LLVQ remains competitive without rotations when norm-preserving corrections are used, indicating reduced reliance on online transforms.

| | | | | Llama-2 7B | | |
|---|---|---|---|---|---|---|
| **Method** (no finetune) | **Dim** | **BPW** | **Hadamard** | **Wiki ↓** | **MMLU ↑** | **CSR ↑** |
| Baseline | 1 | 16 | - | 5.12 | 45.7 | 70.4 |
| Integer (GPTQ) | 1 | 2 | No Rotation | 3411.6 | 26.6 | 39.7 |
| Integer (Quarot) | 1 | 2 | Input | 41.87 | 27.0 | 41.7 |
| Integer | 1 | 2 | Input + Output | 37.83 | 26.1 | 48.4 |
| E8P | 8 | 2 | No Rotation | 105.98 | 24.8 | 44.9 |
| E8P | 8 | 2 | Input | 9.24 | 31.0 | 59.8 |
| E8P (Quip#) | 8 | 2 | Input + Output | 7.96 | 30.5 | 61.4 |
| LLVQ [spherical shaping] | 24 | 2 | No Rotation | 191.90 | 24.0 | 35.5 |
| LLVQ [spherical shaping] | 24 | 2 | Input | **6.80** | **35.1** | **65.4** |
| LLVQ [spherical shaping] | 24 | 2 | Input + Output | **7.61** | **33.4** | **62.1** |
| LLVQ [shape-gain, 2 bit gain] | 24 | 2 | No Rotation | **7.27** | **29.8** | **61.5** |
| LLVQ [shape-gain, 2 bit gain] | 24 | 2 | Input | **6.90** | **36.0** | **63.6** |
| LLVQ [shape-gain, 2 bit gain] | 24 | 2 | Input + Output | **6.83** | **34.9** | **64.6** |

**Hadamard-free PTQ.** Table 5 summarizes an ablation over (i) scalar integer baselines, (ii) E8P/Quip#, and (iii) LLVQ, under different preprocessing choices. Two trends emerge. First, rotations strongly benefit scalar quantization and low-dimensional codes, consistent with their limited representational flexibility. Second, LLVQ remains competitive *without* rotations when paired with the Spherical GPTQ interpretation: norm preservation eliminates the dominant failure mode (radial drift), making Hadamard preprocessing far less critical. In particular, LLVQ shape–gain exhibits strong Hadamard-free PTQ behavior, indicating that sufficiently high-dimensional, low-angular-distortion codes can mitigate or even remove the need for online rotations.

From a geometric perspective, the combination of (a) high-dimensional lattice-based directional quantization and (b) norm-preserving GPTQ updates yields an effectively (hyper-)Spherical PTQ procedure: quantization primarily induces angular error, while block norms are preserved and refined via closed-form scaling. We emphasize that Spherical GPTQ is not the only reason LLVQ outperforms baselines such as E8P/Quip# and QTIP, since even when we remove shape–gain completely and use standard (Euclidean) GPTQ, Table 5 shows that LLVQ still outperforms the $E_8$/Quip# codebook. This is expected: beyond its favourable spherical-code properties, the Leech lattice also provides a lower-distortion Euclidean quantization grid than $E_8$ at comparable rates, so its advantage persists even without norm-preserving (spherical) corrections.

### 5.3 Comparison to Results in Literature

In addition to the unified PTQ evaluation presented earlier, where all methods are assessed under the exact same pipeline for strict apples-to-apples comparison, Table 4 further broadens the analysis to include results previously reported in the literature. Although published baselines such as OmniQ,

Table 6: Comparison of performance after quantizing Llama-2, Llama-3, Ministral-3 and Qwen-v3 language models using different quantization methods, evaluated by Wikitext-2 (Wiki) perplexity at 4096 context length and downstream task performance (CSR and MMLU) on own consistent training pipeline. LLVQ consistently outperforms standard vector quantization approaches.

| | Fine- | | Llama-2 7B | | | Llama-3 8B | | | Ministral-3 8B instruct | | | Qwen-3 4B | | | Qwen-3 8B | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Method** (same pipeline) | **tuned** | **BPW** | **Wiki↓** | **MMLU↑** | **CSR↑** | **Wiki↓** | **MMLU↑** | **CSR↑** | **Wiki↓** | **MMLU↑** | **CSR↑** | **Wiki↓** | **MMLU↑** | **CSR↑** | **Wiki↓** | **MMLU↑** | **CSR↑** |
| Baseline | - | 2 | 5.11 | 45.7 | 70.4 | 5.75 | 65.5 | 74.6 | 6.44 | 65.1 | 76.4 | 12.41 | 70.2 | 71.2 | 8.99 | 74.9 | 74.0 |
| GPTQ + Rotation (Quarot) | No | 2 | 41.87 | 27.0 | 41.7 | 94.37 | 25.2 | 43.3 | 41.22 | 26.7 | 44.4 | 280.7 | 26.3 | 43.6 | 41.62 | 29.9 | 47.8 |
| Quip#/E8P12 | No | 2 | 7.96 | 30.5 | 61.4 | 12.25 | 40.5 | 62.0 | 10.83 | 49.6 | 65.7 | 21.15 | 48.6 | 57.2 | 12.80 | 60.5 | 67.0 |
| QTIP (3INST) | No | 2 | 7.28 | 34.2 | 63.0 | 9.59 | **50.1** | 66.4 | 8.96 | 55.0 | 71.6 | 17.04 | 57.4 | 63.5 | 11.17 | 66.1 | 71.1 |
| LLVQ [spherical shaping] (**ours**) | No | 2 | 7.61 | 33.4 | 62.1 | 11.49 | 41.9 | 64.8 | 10.32 | 50.2 | 66.5 | 21.80 | 50.5 | 58.7 | 12.20 | 63.7 | 68.7 |
| LLVQ [shape-gain, 2 bit gain] (**ours**) | No | 2 | 6.83 | 34.9 | 64.6 | 9.35 | 48.7 | 66.4 | 8.56 | 56.6 | 71.3 | **15.54** | 59.3 | **64.1** | 10.82 | 67.2 | 69.9 |
| LLVQ [shape-gain, 0 bit gain] (**ours**) | No | 2 | **6.48** | **35.4** | **66.5** | **8.50** | **52.6** | **69.3** | **8.11** | **58.4** | **72.2** | 17.05 | **60.7** | 63.6 | **10.19** | **69.3** | **70.2** |
| Quip#/E8P12 | Yes | 2 | 5.73 | 30.6 | 64.9 | 7.92 | 48.1 | 66.7 | 7.54 | 54.9 | 70.6 | 10.52 | 52.9 | 65.2 | 8.31 | 63.7 | 70.1 |
| QTIP (3INST) | Yes | 2 | 5.50 | 36.9 | 66.5 | 7.28 | 53.5 | 69.1 | 7.06 | 56.7 | 73.3 | 9.61 | 59.5 | 66.9 | 7.82 | 68.0 | 72.8 |
| LLVQ [spherical shaping] (**ours**) | Yes | 2 | 5.60 | 35.8 | 65.3 | 7.64 | 47.8 | 68.7 | 7.34 | 53.8 | 70.5 | 10.13 | 54.9 | 65.1 | 8.09 | 66.4 | 71.5 |
| LLVQ [shape-gain, 2 bit gain] (**ours**) | Yes | 2 | 5.48 | 37.3 | 66.8 | 7.29 | 53.4 | 70.0 | 7.04 | 57.6 | 72.5 | 9.51 | 60.9 | 67.6 | 7.79 | 68.8 | 72.6 |
| LLVQ [shape-gain, 0 bit gain] (**ours**) | Yes | 2 | **5.33** | **38.0** | **68.1** | **6.99** | **55.7** | **70.7** | **6.83** | **59.2** | **73.0** | **9.26** | **62.8** | **66.1** | **7.59** | **69.6** | **72.3** |

AQLM, Quip#, and QTIP may slightly differ in training conditions, calibration set composition, and dataset sizes, these comparisons remain highly meaningful performance across independent pipelines.

Optionally, we incorporate a lightweight fine-tuning step that learns an element-wise multiplicative correction on the inputs of each linear layer, following (Tseng et al., 2024a); equivalently, this can be viewed as learning per-column scaling factors for the weight matrices. Because these scalars are shared across rows, the overhead is negligible (less than $0.001$ bits per weight even in full 32-bit precision). We train only these scale parameters for a short run of roughly 52M tokens. This minimal adaptation reliably improves perplexity, MMLU, and CSR across quantization methods, acting as a lightweight alignment step.

Notably, LLVQ maintains a clear advantage compared to the strongest results reported across the broader literature. Crucially, LLVQ *without* any fine-tuning is competitive, sometimes surpassing, the performance of the best baselines *with* fine-tuning. This despite that we use a very strict definition of "no fine-tuning" for our own method, meaning that all corrections arise solely from layer-local, Hessian-based updates derived from activation statistics, with no reliance on gradient updates, such as the inter-layer fine-tuning used in Quip# and QTIP, and without any end-to-end tuning of the quantized model. Overall, LLVQ achieves state-of-the-art performance in a strictly PTQ setting, outperforming methods that rely on additional fine-tuning for recovery. When fine-tuning is added (for LLVQ, only shared row-/column-wise scale terms), the performance gap widens further, yielding results close to the baseline model (2.5%–7.6% degradation in benchmark accuracies), pushing the frontier of practical LLM quantization into the ultra-low-bitrate regime of just 2 bits per weight.

## 6 Conclusion

We introduced *Leech Lattice Vector Quantization* (LLVQ), a practical high-dimensional vector quantizer, grounded in the geometric and combinatorial structure of the Leech lattice. LLVQ provides an expressive and computationally efficient alternative to conventional scalar and low-dimensional vector quantizers. Our contributions include: (i) an extended shell-based search procedure supporting multi-shell codes, (ii) a fully invertible indexing scheme enabling codebook-free quantization and dequantization, and (iii) demonstrating Leech-lattice-based vector quantization of LLMs.

Finally, we provided a geometric interpretation of combining shape–gain with scale-corrections and GPTQ-style corrections: we show the standard scale-correction used in shape–gain acts as a retraction onto a product of spheres, yielding a *Spherical GPTQ* viewpoint where the residual distortion is dominated by angular error. This perspective is particularly natural for *spherical* (directional) codes, which we show can also be constructed from the Leech lattice, and empirically demonstrate reduced sensitivity to Hadamard/rotation preprocessing enabling *Hadamard-free PTQ*, reducing or eliminating the need for online Hadamard transforms at inference time.

Experimentally, LLVQ achieves state-of-the-art performance in both idealized and practical settings. On Gaussian sources, LLVQ realizes the highest SQNR among competing quantizers, achieving over 92% retention of the Shannon limit at 2 bits/dim. On all assessed large language models of the Llama-2 and Llama-3, Ministral-3, and Qwen-v3 model families, LLVQ consistently outperforms existing PTQ baselines such as AQLM, Quip# and QTIP across perplexity and downstream task performance. This shows that the theoretical benefits of high-dimensional lattices on Gaussian data translates to practical benefits for modern LLM compression.

Overall, LLVQ demonstrates that high-dimensional lattices offer substantial benefits for modern neural network compression. We hope this work inspires further exploration of mathematically grounded quantization schemes for scalable and efficient large model deployment.

## References


Adoul, J.-P. and Barth, M. Nearest neighbor algorithm for spherical codes from the leech lattice. *IEEE transactions on information theory*, 34(5):1188–1202, 1988.

Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlíček, H., Lajarín, A. P., Srivastav, V., et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Ashkboos, S., Mohtashami, A., Croci, M. L., Li, B., Cameron, P., Jaggi, M., Alistarh, D., Hoefler, T., and Hensman, J. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Bannai, E. and Sloane, N. J. Uniqueness of certain spherical codes. *Canadian Journal of Mathematics*, 33(2):437–449, 1981.

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

Conway, J. H. and Sloane, N. J. A. *Sphere packings, lattices and groups*, volume 290. Springer Science & Business Media, 2013.

Ericson, T. and Zinoviev, V. *Codes on Euclidean spheres*, volume 63. Elsevier, 2001.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gersho, A. and Gray, R. M. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012.

Gray, R. M. and Neuhoff, D. L. Quantization. *IEEE transactions on information theory*, 44(6): 2325–2383, 2002.

Hamkins, J. and Zeger, K. Gaussian source coding with spherical codes. *IEEE Transactions on Information Theory*, 48(11):2980–2989, 2002.

International Mathematical Union. Fields medals 2022, 2022. URL `https://www.mathunion.org/imu-awards/fields-medal/fields-medals-2022`.

Kabatiansky, G. A. and Levenshtein, V. I. On bounds for packings on a sphere and in space. *Problemy peredachi informatsii*, 14(1):3–25, 1978.

Leech, J. Notes on sphere packings. *Canadian Journal of Mathematics*, 19:251–267, 1967.

Li, J., Fang, A., Smyrnis, G., Ivgi, M., Jordan, M., Gadre, S., Bansal, H., Guha, E., Keh, S., Arora, K., et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Nagel, M., Amjad, R. A., Van Baalen, M., Louizos, C., and Blankevoort, T. Up or down? adaptive rounding for post-training quantization. In *International conference on machine learning*, pp. 7197–7206. PMLR, 2020.

Odlyzko, A. M. and Sloane, N. J. New bounds on the number of unit spheres that can touch a unit sphere in n dimensions. *Journal of Combinatorial Theory, Series A*, 26(2):210–214, 1979.

Sabin, M. and Gray, R. Product code vector quantizers for speech waveform coding. In *Globecom 1982-Global Telecommunications Conference*, volume 3, pp. 1087–1091, 1982.

Shannon, C. E. A mathematical theory of communication. *The Bell system technical journal*, 27(3): 379–423, 1948.

Shannon, C. E. et al. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat. Conv. Rec*, 4(142-163):1, 1959.

Tseng, A., Chee, J., Sun, Q., Kuleshov, V., and De Sa, C. Quip#: Even better llm quantization with hadamard incoherence and lattice codebooks. *arXiv preprint arXiv:2402.04396*, 2024a.

Tseng, A., Sun, Q., Hou, D., and De Sa, C. M. Qtip: Quantization with trellises and incoherence processing. *Advances in Neural Information Processing Systems*, 37:59597–59620, 2024b.

Tseng, A., Sun, Z., and De Sa, C. Model-preserving adaptive rounding. *arXiv preprint arXiv:2505.22988*, 2025.

Van Baalen, M., Kuzmin, A., Koryakovskiy, I., Nagel, M., Couperus, P., Bastoul, C., Mahurin, E., Blankevoort, T., and Whatmough, P. Gptvq: The blessing of dimensionality for llm quantization. *arXiv preprint arXiv:2402.15319*, 2024.

van der Ouderaa, T. F., Nagel, M., Van Baalen, M., Asano, Y. M., and Blankevoort, T. The llm surgeon. *arXiv preprint arXiv:2312.17244*, 2023.

van der Ouderaa, T. F., Croci, M. L., Hilmkil, A., and Hensman, J. Pyramid vector quantization for llms. *arXiv preprint arXiv:2410.16926*, 2024.

## NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

**The checklist answers are an integral part of your paper submission.** They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the contributions (LLVQ, indexing, multi-shell search, parallel dequantization) and these are directly supported by both theoretical discussion and empirical results (Sections 1, 6, 7).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: Practical considerations are discussed and assumptions stated.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

4. **Theory assumptions and proofs**

   Answer: [N/A]

   Justification: The paper contains theoretical formulations and derivations (e.g., lattice structure, indexing, and rate–distortion discussion) and refers to relevant proofs or stated assumptions when needed.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

5. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

Justification: The paper describes the quantization pipeline, datasets, calibration size, evaluation metrics, and comparisons in sufficient detail to reproduce the experiments (Section 7 and tables).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

6. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [N/A]

Justification: The paper does not yet provide an open-source implementation or explicit code release, although it describes the algorithmic details needed for reproduction. Code will be provided upon acceptance.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

7. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Key details such as calibration data size, evaluation datasets (Wikitext-2, MMLU, CSR), model architectures, and quantization pipeline are specified sufficiently for understanding the experimental setup (Section 7).

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

8. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [N/A]

   Justification: The paper reports single-run results for large-scale post-training quantization experiments, which is standard practice in the literature due to their computational cost. Furthermore, methods are consistently evaluated across multiple model architectures, providing robust evidence for the reported performance trends despite potential minor run-to-run variance.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

9. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The paper does specify experimental settings and experimentation details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

10. **Code of ethics**

    Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

    Answer: [Yes]

    Justification: The research complies with the NeurIPS Code of Ethics and does not involve human subjects, sensitive data, or disallowed practices.

    Guidelines:

    - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
    - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
    - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

11. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: The paper does not explicitly discuss broader societal impacts; however, the work relates to model compression, which may positively affect accessibility, efficiency, and deployment of large models.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.

- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

12. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release trained models or datasets, and thus no additional safeguards are required.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

13. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper uses standard datasets and benchmarks (e.g., Wikitext-2, MMLU) and cites relevant prior work and models appropriately.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

14. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: The paper does not introduce new datasets, models, or publicly released code assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

15. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve human subjects or crowdsourced data collection.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

16. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: No human subjects are involved in this work.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

17. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: Large language models are studied as objects of quantization, not used as part of the methodology itself.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

# A Dequantizer

The dequantizer recovers a 24-dimensional integer vector from its global integer index:

$$\text{Dequantizer} : \{1, \ldots, N(m)\} \to L^{\text{int}}(m) \subset \mathbb{Z}^{24}. \tag{10}$$

Because the indexing is hierarchical, the inverse map of the dequantizer mirrors this structure and consists of a small number of inexpensive integer operations.

**1. Shell Identification.** Given an index $I$, determine the shell by locating the unique $k$ such that $N(k) < I \leq N(k+1)$. This requires only a lookup in a small table of cumulative shell sizes. The shell-local index is $I_{\text{shell}} = I - N(k)$.

**2. Class Identification.** Each shell contains a fixed, precomputed list of classes with cumulative offsets $C_1, C_2, \ldots, C_J$. The class index $j$ satisfies $C_{j-1} < I_{\text{shell}} \leq C_j$, and the class-local index is $I_{\text{class}} = I_{\text{shell}} - C_{j-1}$.

**3. Unpacking local symmetries.** Within a class, degrees of freedom factor into (i) a Golay refinement of cardinality $A$, (ii) a valid sign configuration with $2^B$ possibilities, and (iii) a permutation coset encoded by a rank in its orbit. The class-local index is unflattened by modulo and integer-division:

$$r = I_{\text{class}} \bmod A, \qquad I' = \lfloor I_{\text{class}}/A \rfloor, \quad s = I' \bmod 2^B, \quad I'' = \left\lfloor I'/2^B \right\rfloor, \tag{11}$$

where $r$ selects the Golay refinement, $s$ selects the sign pattern, and $I''$ encodes the permutation rank.

**4. Reconstruction of the integer vector.** Starting from the class leader, reconstruction proceeds in three conceptual stages while avoiding enumerations. First, the absolute-value pattern is rearranged by applying the permutation encoded by $I''$, yielding the correct coordinate placement for the class. Next, signs are assigned in a manner consistent with the class constraints: for *odd leaders*, all $4096$ Golay codewords are admissible, and sign allocation follows the fixed $1 \bmod 4$ versus $3 \bmod 4$ structure; for *even leaders*, only refinements of the appropriate Golay weight are allowed, and the sign vector must satisfy the Conway–Sloane parity and sum constraints. Finally, the Golay refinement $r$ specifies the congruence class of $(x - m\mathbf{1})/2 \mod 2$, thereby fixing the remaining binary degrees of freedom and completing the integer vector in $L^{\text{int}}(m)$.

**5. Parallel Implementation (kernel).** All components of the dequantizer, shell lookup, class lookup, and local symmetry unflattening, depend only on small static tables, integer prefix-sum scans, integer division and modulo, and local combinatorial reconstruction. There are no dependencies between vectors, no large memory accesses. Further, the procedure is therefore trivially parallelizable across blocks of 24-dimensional vectors and maps naturally to GPU execution (e.g., a CUDA kernel).

## B Overall of Shape-gain Quantization Procedure with Hessian Corrections

We quantize each layer in a sequential, block-wise manner. Starting from $\widetilde{\mathbf{W}} \leftarrow \mathbf{W}^{(l)}$, we process input-channel blocks $Q_1, \ldots, Q_m$ in a fixed left-to-right order. At step $t$, we apply shape–gain quantization to the current (already corrected) block $\widetilde{\mathbf{W}}_{:,Q_t}$ by quantizing its directions and then resetting the scale to match the original block norm. We form the GPTQ error $\mathbf{E}_{:,Q_t} = \widetilde{\mathbf{W}}_{:,Q_t} - \hat{\mathbf{W}}_{:,Q_t}$, which is then propagated into the remaining, unprocessed columns $R = \bigcup_{u>t} Q_u$ using the analytic Hessian correction derived in Sec. F.2. Using the current $\widetilde{\mathbf{W}}$ (rather than the original $\mathbf{W}$) is essential: it ensures that each subsequent block is quantized only after compensating for errors introduced earlier, so the procedure greedily minimizes the local quadratic proxy at every step while keeping already-quantized blocks fixed. Interestingly, the combination of resetting the scales which is standard in shape-gain quantization and GPTQ-like updates admits a simple geometric interpretation, the scale reset enforces a constant-norm constraint on each block, so quantization primarily induces angular (directional) error on a product of spheres. The subsequent Hessian-based correction then propagates this residual angular error through the local quadratic surrogate, yielding a GPTQ-style update that effectively operates on the hypersphere rather than in unconstrained Euclidean space (see Sec. I).

**Algorithm 1** Overall Shape–Gain Quantization with Hessian Corrections

**Require:** Layer weights $\mathbf{W}^{(l)} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, calibration inputs $\mathbf{X} \in \mathbb{R}^{N \times d_{\text{in}}}$, block size $b$=24, direction quantizer $\mathcal{Q}_{\text{shape}}$ (Leech), optional gain quantizer $\mathcal{Q}_{\text{gain}}$

**for** each layer $l$ **do**
  $\mathbf{H} \leftarrow \frac{1}{N}\mathbf{X}^\top\mathbf{X}$
  $\mathbf{U} \leftarrow \mathrm{chol}(\mathbf{H}^{-1})^\top$
  $\widetilde{\mathbf{W}} \leftarrow \mathbf{W}^{(l)}$
  Partition $\{1, \ldots, d_{\text{in}}\}$ into blocks $Q_1, \ldots, Q_m$ of size $b$ (last may be smaller)
  **for** $t = 1$ **to** $m$ **do**
    $Q \leftarrow Q_t, \quad R \leftarrow \bigcup_{u>t} Q_u$
    **for each row** $i = 1$ **to** $d_{\text{out}}$ **(in parallel) do**
      $\mathbf{v} \leftarrow \widetilde{\mathbf{W}}_{i,Q}$
      $\hat{\mathbf{v}} \leftarrow \|\mathbf{v}\|_2 \cdot \mathcal{Q}_{\text{dir}}(\mathbf{v}/\|\mathbf{v}\|_2)$
      **optionally:** $\hat{\mathbf{v}} \leftarrow \mathcal{Q}_{\text{gain}}(\|\mathbf{v}\|_2) \cdot \mathcal{Q}_{\text{dir}}(\mathbf{v}/\|\mathbf{v}\|_2)$
      $\hat{\mathbf{W}}_{i,Q} \leftarrow \hat{\mathbf{v}}$
    **end for**
    $\mathbf{E}_{:,Q} \leftarrow \widetilde{\mathbf{W}}_{:,Q} - \hat{\mathbf{W}}_{:,Q}$
    $\widetilde{\mathbf{W}}_{:,Q} \leftarrow \hat{\mathbf{W}}_{:,Q}$
    **if** $R \neq \emptyset$ **then**
      $\widetilde{\mathbf{W}}_{:,R} \leftarrow \widetilde{\mathbf{W}}_{:,R} - \left(\mathbf{E}_{:,Q}\mathbf{U}_{QQ}^{-1}\right)\mathbf{U}_{QR}$
    **end if**
  **end for**
  $\mathbf{W}^{(l)} \leftarrow \widetilde{\mathbf{W}}$
**end for**

# C Efficient Implementation and CUDA Kernel

## C.1 Timing results on NVIDIA with CUDA kernel

We implement a fused CUDA kernel that performs both dequantization and dot product accumulation directly, avoiding materialization of the weight matrix and reduces memory traffic and improves throughput. This is especially important for autoregressive inference regime of large language models, where computation is typically memory-bound. We stick to the single shell $M = 3$ case for simplicity.

Empirically, we observe that the fused LLVQ kernel achieves competitive performance on NVIDIA GPUs. The LLVQ kernel is faster than the FP16 matrix–vector baseline, demonstrating that the structured LLVQ representation can be evaluated efficiently without materializing the full weight matrix. While our current implementation is slightly slower than QTIP, LLVQ outperforms QTIP in terms of representation quality, as reflected by the perplexity and downstream evaluation results reported in the main paper.

**Remark.** We emphasize that this work does not aim to make definitive claims about optimized runtime performance, since low-level kernel engineering and hardware-specific optimizations can likely further improve the runtime of our kernel implementations. Such optimizations are largely orthogonal to the main contribution of this paper, which focuses on the quality of the representation and its impact on perplexity and downstream metrics. Nevertheless, we include these timing measurements to demonstrate the feasibility of the approach and its compatibility with efficient fused implementations.

Table 7: CUDA kernel runtime for fused dequantization with matrix–vector (matvec) products

| Shape | Kernel | Runtime |
|---|---|---|
| FP16 matvec | $(4096\times4096)\cdot(4096\times1)$ | 16.3 $\mu$s |
| FP16 matvec | $(4096\times4104)\cdot(4104\times1)$ | 17.69 $\mu$s |
| LLVQ-FP16 (fused dequant + matvec) | $(4096\times4104)\cdot(4104\times1)$ | 11.94 $\mu$s (1.36/1.48$\times$ speedup) |

# D Shape-Gain Quantization and Constructing Spherical Codes from Lattices

An alternative to spherical shaping, where all lattice points within a prescribed radius are taken as the finite quantization grid, is *shape–gain* quantization (Sabin & Gray, 1982; Hamkins & Zeger, 2002; Gersho & Gray, 2012). In shape–gain, the magnitude of a vector (its "gain") is quantized separately using a standard scalar quantizer, while its direction (its "shape") is quantized using a spherical quantizer defined over a spherical code, i.e., a finite subset of points on the high-dimensional unit sphere. Although this approach is slightly more involved than spherical shaping, as it requires two quantizers rather than one, it can yield superior rate–distortion performance for Gaussian sources, as we demonstrate in App. H.

More formally, shape–gain quantization considers (polar-)decomposing each vector into its *shape* (direction) and *gain* (magnitude). Any non-zero vector $\mathbf{x} \in \mathbb{R}^n$ can be written as

$$\mathbf{x} = r\,\mathbf{u}, \qquad r = \|\mathbf{x}\| \in \mathbb{R}, \;\; \mathbf{u} = \mathbf{x}/\|\mathbf{x}\| \in \mathbb{S}^{n-1}. \tag{12}$$

Quantization then treats $r$ and $\mathbf{u}$ separately, so that the overall code becomes the Cartesian product of a scalar quantizer on $\mathbb{R}$ and a quantizer on the sphere $\mathbb{S}^{n-1}$. For Gaussian sources, this factorization is particularly appealing: the gain $r$ follows a root chi-square distribution (since $r^2 \sim \chi^2_n$), which can be accurately quantized using an analytically tractable inverse CDF. The core design challenge therefore lies in constructing expressive, approximately uniformly distributed, and high-rate spherical codes, together with algorithms that efficiently map a vector to its nearest code point on the sphere and represent that point using a compact index.

**Spherical codes** A spherical code is formally defined as a $(d, N, s)$-code: a collection of $N$ points on the sphere $S^{d-1}$ such that the inner product between any two distinct points is at most $s$. An optimal $(d, N, s)$-code is one for which no $(d, N_0, s)$-code with $N_0 > N$ exists. Among the most celebrated examples are the minimal vectors of the $E_8$ and Leech lattices, which yield an $(8, 240, \frac{1}{2})$-code and a $(24\ 196{,}560, \frac{1}{2})$-code, respectively. Their optimality was established by Kabatiansky & Levenshtein (1978); Odlyzko & Sloane (1979), and their uniqueness by Bannai & Sloane (1981). These spherical codes also solve the kissing number problem in their respective dimensions, achieving $240$ contacts in $\mathbb{R}^8$ and $196{,}560$ in $\mathbb{R}^{24}$ (Ericson & Zinoviev, 2001; Conway & Sloane, 2013).

Despite their optimality, the first shells of $E_8$ and the Leech lattice contain too few points to serve as high-bitrate spherical codes: $\log_2(240)/8 \approx 1$ bit/dim for $E_8$ and $\log_2(196{,}560)/24 \approx 0.73$ bit/dim for the Leech lattice. Such bitrates are insufficient for practical neural network quantization, which typically operates in the 2–3 bits/dim or higher (up to 16) range. To obtain richer directional codebooks, we consider a normalized spherically bounded subset of the Leech lattice, i.e., the union of all lattice points within a chosen radius, which forms a dense spherical code after normalization. These multi-shell Leech-based codes serve as the foundation for our angular quantization scheme. Their geometric uniformity minimizes directional distortion, and their high density supports bitrates compatible with modern compression and quantization needs. When combined with an appropriate scalar quantizer for the gain, the resulting shape-gain quantizer performs well not only for ideal Gaussian sources but also for real-world neural network weight distributions.

## E Quantizer/dequantizer block diagrams

We quantize a vector $\mathbf{w} \in \mathbb{R}^d$ using a mapping $q : \mathbb{R}^d \to [1, N(m)]$, where the integer index range $[1, N(m)]$ is sized such that $\log_2(N(m))/24$ equals the desired bits per dimension. The quantizer output is $i_{\hat{\mathbf{w}}} = q(\mathbf{w})$, and the reconstruction is obtained via the dequantizer Dequantizer : $[1, N(m)] \to \mathbb{R}^d$, yielding $\hat{\mathbf{w}} = \text{Dequantizer}(i_{\hat{\mathbf{w}}})$. This abstraction covers a family of encoders with geometric preprocessing (e.g., shaping or factorization) followed by index selection, and a corresponding inverse mapping at the decoder.

**Spherical shaping.** Figure 2 depicts the *spherical shaping* variant based on a ball cut of the Leech lattice. The vector $\mathbf{w}$ is first projected into a spherical shaping region, typically the Euclidean ball $\mathbb{B}(0, R)$, and the index is selected by nearest-neighbor search over $\Lambda_{24} \cap \mathbb{B}(0, R)$:

$$i_{\hat{\mathbf{w}}} = q(\mathbf{w}) = \arg \min_{i:\, \mathbf{c}_i \in \Lambda_{24} \cap \mathbb{B}(0,R)} \|\mathbf{w} - \mathbf{c}_i\|_2^2, \qquad \hat{\mathbf{w}} = \mathbf{c}_{i_{\hat{\mathbf{w}}}}.$$

This realizes high shaping efficiency while enforcing a finite-energy codebook.

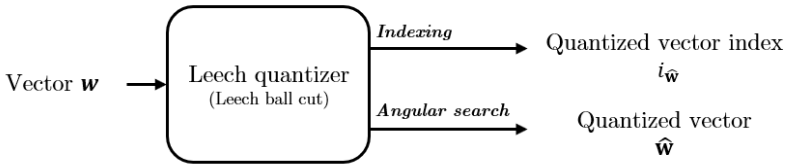



Figure 2: Spherical shaping: ball cut of the Leech lattice with nearest-neighbor selection over $\Lambda_{24} \cap \mathbb{B}(0, R)$.

**Shape–gain.** In the *shape–gain* framework, $\mathbf{w} = g\,\mathbf{s}$ with $g = \|\mathbf{w}\|_2$ and $\mathbf{s} = \mathbf{w}/\|\mathbf{w}\|_2 \in \mathbb{S}^{d-1}$. The independent variant in Fig. 3 applies separate quantizers to shape and gain,

$$i_{\hat{\mathbf{w}}} = q(\mathbf{w}) = \big(q_{\text{shape}}(\mathbf{s}),\, q_{\text{gain}}(g)\big) \in [1, N(m)],$$

and the dequantizer reconstructs $\hat{\mathbf{s}}$ and $\hat{g}$ and returns $\hat{\mathbf{w}} = \hat{g}\,\hat{\mathbf{s}}$. While computationally simple, independence may induce norm mismatch because angular errors perturb the effective magnitude.

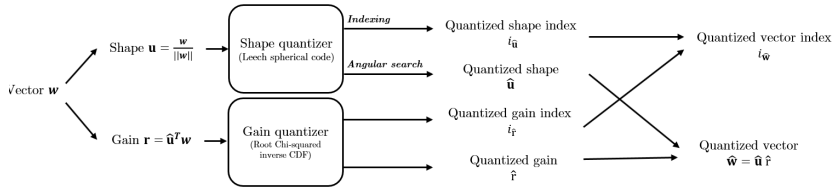



Figure 3: Shape–gain with independent quantization of shape $\mathbf{s}$ and gain $g$, producing an index in $[1, N(m)]$ and reconstruction $\hat{\mathbf{w}} = \hat{g}\,\hat{\mathbf{s}}$.

To mitigate this, Fig. 4 implements *shape–gain with optimal scales* (post-shape gain optimization). After selecting the shape index, the quantized direction $\hat{\mathbf{s}}$ is known; the encoder computes

$$\gamma^\star = \langle \mathbf{w}, \hat{\mathbf{s}} \rangle, \qquad i_{\hat{\mathbf{w}}} = \big(q_{\text{shape}}(\mathbf{s}),\, q_{\text{gain}|\hat{\mathbf{s}}}(\gamma^\star)\big) \in [1, N(m)],$$

and the dequantizer mirrors this conditional mapping to obtain $\hat{g}$ and reconstruct $\hat{\mathbf{w}} = \hat{g}\,\hat{\mathbf{s}}$. Unless stated otherwise, this *shape–gain with optimal scales* configuration is the one used in our main LLVQ experiments due to its superior rate–distortion behavior.

**Dequantizer.** Given the integer index $i_{\hat{\mathbf{w}}} \in [1, N(m)]$, the dequantizer in Fig. 5 maps back to $\mathbb{R}^d$ by retrieving the appropriate representatives (lattice point or shape/gain codewords) and recombining them:

$$\text{Dequantizer} : [1, N(m)] \to \mathbb{R}^d, \qquad \hat{\mathbf{w}} = \text{Dequantizer}(i_{\hat{\mathbf{w}}}).$$

This guarantees consistency with the encoder's shaping or factorization policy.

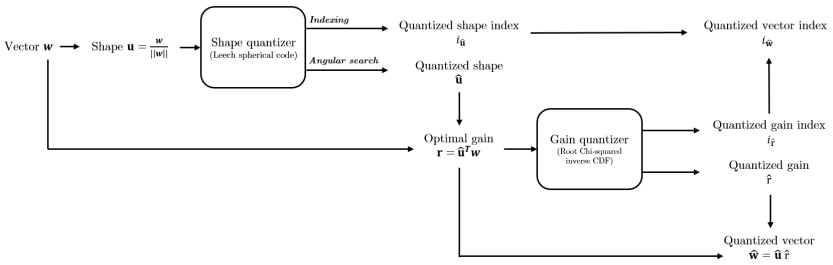



Figure 4: Shape–gain with optimal scales: post-shape gain optimization via a shape-conditioned gain quantizer.

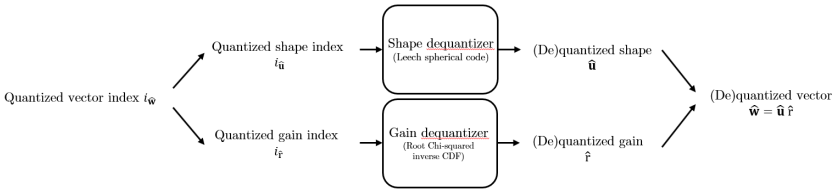



Figure 5: Dequantizer mapping the integer index $i_{\hat{\mathbf{w}}} \in [1, N(m)]$ to the reconstruction $\hat{\mathbf{w}} \in \mathbb{R}^d$.

A detailed comparison between spherical shaping and shape–gain is provided in Appendix H. There, we found shape–gain to yield improved performance over spherical shaping, when using the Leech lattice; accordingly, we adopt the shape–gain with optimal scales scheme in our main LLVQ experiments unless explicitly noted otherwise.

# F Algorithm specification

## F.1 Optimal scales under shape–gain

A key benefit of spherical (shape–gain) vector quantization is that the *shape* quantizer can be made scale-invariant: it depends only on direction. Concretely, let $\boldsymbol{u} = \boldsymbol{w}/\|\boldsymbol{w}\|_2$ and define a spherical quantizer $q(\boldsymbol{w}) \equiv q(\boldsymbol{u})$ whose codewords have fixed norm (e.g., $\|q(\boldsymbol{u})\|_2 = 1$). Then for any $s > 0$,

$$q(s\boldsymbol{w}) = q\left(\frac{s\boldsymbol{w}}{\|s\boldsymbol{w}\|_2}\right) = q\left(\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|_2}\right) = q(\boldsymbol{w}),$$

i.e., $q$ is invariant to positive rescalings.[2] As a result, a scaled reconstruction of the form $\hat{\boldsymbol{w}}(\beta) = \beta\, q(\boldsymbol{w}/\beta)$ simplifies (for $\beta > 0$) to

$$\hat{\boldsymbol{w}}(\beta) = \beta\, q(\boldsymbol{w}),$$

eliminating the need for a line search over $\beta$ and yielding closed-form optimal scales.

**Optimal closed-form scale for weight-space error.** With $\boldsymbol{q} \coloneqq q(\boldsymbol{w})$ fixed, the scale minimizing the weight reconstruction error is

$$\beta_* = \arg\min_{\beta} \|\boldsymbol{w} - \beta\, \boldsymbol{q}\|_2^2 = \frac{\boldsymbol{q}^\top \boldsymbol{w}}{\boldsymbol{q}^\top \boldsymbol{q}}, \tag{13}$$

i.e., the projection of $\boldsymbol{w}$ onto its quantized shape $\boldsymbol{q}$. In the group-wise case, where weights are partitioned into blocks $\boldsymbol{w}_i$, the same form applies per block:

$$\beta_i^* = \frac{q(\boldsymbol{w}_i)^\top \boldsymbol{w}_i}{q(\boldsymbol{w}_i)^\top q(\boldsymbol{w}_i)}.$$

**Optimal closed-form scales for activation-space (local output) error.** Consider a linear map with columns $\boldsymbol{w}_i \in \mathbb{R}^{d_{\text{out}}}$ and input $x \in \mathbb{R}^G$:

$$Wx = \sum_{i=1}^{G} \boldsymbol{w}_i\, x_i.$$

Approximating $\boldsymbol{w}_i \approx \beta_i\, q(\boldsymbol{w}_i)$ yields $\widehat{W}x = \sum_{i=1}^{G} \beta_i\, q(\boldsymbol{w}_i)\, x_i$. Define

$$\boldsymbol{A} \coloneqq \big[\, q(\boldsymbol{w}_1)x_1, \ldots, q(\boldsymbol{w}_G)x_G \,\big] \in \mathbb{R}^{d_{\text{out}} \times G}, \qquad \boldsymbol{b} \coloneqq Wx \in \mathbb{R}^{d_{\text{out}}}.$$

Then the optimal group-wise scales are the least-squares solution

$$\boldsymbol{\beta}_* = \arg\min_{\boldsymbol{\beta}} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{\beta}\|_2^2 = (\boldsymbol{A}^\top \boldsymbol{A})^{-1} \boldsymbol{A}^\top \boldsymbol{b}, \tag{14}$$

or, more robustly, with damping $\lambda > 0$:

$$\boldsymbol{\beta}_* = (\boldsymbol{A}^\top \boldsymbol{A} + \lambda I)^{-1} \boldsymbol{A}^\top \boldsymbol{b},$$

and for multiple calibration inputs one can stack the corresponding $(\boldsymbol{A}, \boldsymbol{b})$ pairs across samples and solve the resulting (regularized) least-squares problem.

[2] For $s < 0$, directions flip. If the codebook is antipodally symmetric (contains $\pm\boldsymbol{c}$), then one can also enforce $q(-\boldsymbol{w}) = -q(\boldsymbol{w})$. Otherwise we restrict to $s > 0$.

### F.2 Hessian-based corrections

In post-training quantization, analytic correction steps are interleaved with quantization to compensate for errors introduced by already quantized weights by adjusting the remaining ones. Instead of full fine-tuning, these corrections rely on fast analytic solutions.

**Local Hessian Corrections (block / vector form)** Consider a linear layer $\mathbf{y} = \mathbf{W}\mathbf{x}$ and a quantized weight matrix $\hat{\mathbf{W}} = q(\mathbf{W})$, with error $\Delta\mathbf{W} = \hat{\mathbf{W}} - \mathbf{W}$. A standard local proxy objective (Nagel et al., 2020) is the expected output MSE

$$\mathcal{L}_{\text{local}} = \mathbb{E}\big[\|\Delta\mathbf{W}\mathbf{x}\|_2^2\big] = \mathrm{Tr}\big(\Delta\mathbf{W}\,\mathbf{H}_{\text{in}}\,\Delta\mathbf{W}^\top\big), \qquad \mathbf{H}_{\text{in}} = \mathbb{E}[\mathbf{x}\mathbf{x}^\top]. \tag{15}$$

In practice $\mathbf{H}_{\text{in}}$ is approximated by the empirical second moment $\tilde{\mathbf{H}}_{\text{in}} = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$ from a calibration set $\mathbf{X} \in \mathbb{R}^{N\times D_{\text{in}}}$, which yields a Monte Carlo approximation that converges to the population objective as $N \to \infty$ under i.i.d. sampling.

The loss decomposes over rows: $\mathcal{L}_{\text{local}} = \sum_r \Delta\mathbf{w}_r^\top \mathbf{H}_{\text{in}} \Delta\mathbf{w}_r$,
so each row can be optimized independently. For a single row, suppose we quantize a *set of columns* $Q$ (possibly a block of size $|Q| > 1$) and treat the remaining columns as $R$. Partition the Hessian and the error accordingly:

$$\mathbf{H}_{\text{in}} = \begin{bmatrix} \mathbf{H}_{QQ} & \mathbf{H}_{QR} \\ \mathbf{H}_{RQ} & \mathbf{H}_{RR} \end{bmatrix}, \qquad \Delta\mathbf{w} = [\Delta\mathbf{w}_Q;\, \Delta\mathbf{w}_R].$$

Holding $\Delta\mathbf{w}_Q$ fixed and minimizing $\Delta\mathbf{w}^\top\mathbf{H}_{\text{in}}\Delta\mathbf{w}$ over $\Delta\mathbf{w}_R$ gives the analytic correction

$$\boxed{\Delta\mathbf{w}_R^\star = -\,\mathbf{H}_{RR}^{-1}\mathbf{H}_{RQ}\,\Delta\mathbf{w}_Q.}$$

Applied row-wise, this yields the matrix update (":" denotes all rows)

$$\boxed{\Delta\mathbf{W}_{:,R}^\star = -\,\Delta\mathbf{W}_{:,Q}\,\mathbf{H}_{RR}^{-1}\mathbf{H}_{RQ}^\top} \quad \text{(equivalently } \Delta\mathbf{W}_{:,R}^\star = -\,\Delta\mathbf{W}_{:,Q}\,\mathbf{H}_{QQ}^{-1}\mathbf{H}_{QR} \text{ under block identities).}$$

**Efficient implementation via inverse-Hessian Cholesky (GPTQ-style).** Let $\mathbf{G} = \mathbf{H}_{\text{in}}^{-1}$ and compute an upper-triangular factor $\mathbf{U}$ such that $\mathbf{G} = \mathbf{U}^\top\mathbf{U}$ (e.g., $\mathbf{U} \leftarrow \mathrm{chol}(\mathbf{G})^\top$). When processing columns left-to-right, at any step where $Q$ precedes $R$ in the current ordering, $\mathbf{U}$ has the block structure $\mathbf{U} = \begin{bmatrix} \mathbf{U}_{QQ} & \mathbf{U}_{QR} \\ \mathbf{0} & \mathbf{U}_{RR} \end{bmatrix}$, and the GPTQ compensation for quantization error on $Q$ can be written without forming inverses as

$$\boxed{\mathbf{W}_{:,R} \;\leftarrow\; \mathbf{W}_{:,R} \;-\; \big(\mathbf{E}_{:,Q}\,\mathbf{U}_{QQ}^{-1}\big)\,\mathbf{U}_{QR},} \qquad \mathbf{E}_{:,Q} = \mathbf{W}_{:,Q} - \hat{\mathbf{W}}_{:,Q}.$$

This generalizes the scalar GPTQ/OPTQ update (Frantar et al., 2022) ($|Q| = 1$) to vector/block updates, and can be interpreted as a Schur-complement / conditioning step in the local quadratic.

**Question: Are local proxy objectives problematic?** The local proxy objective above minimizes the expected *layer-output* distortion $\mathbb{E}\|(\hat{\mathbf{W}}-\mathbf{W})\mathbf{x}\|_2^2$ using only the input second moment $\mathbf{H}_{\text{in}} = \mathbb{E}[\mathbf{x}\mathbf{x}^\top]$ (Nagel et al., 2020). From a second-order perspective, this corresponds to a very coarse approximation of the *global* curvature of the end-to-end loss: (i) it effectively treats layers as decoupled (a block-diagonal approximation across layers, ignoring cross-layer coupling and error propagation), and (ii) within each layer it retains only the *input-side* curvature while discarding loss-dependent *output-side* weighting and the effects of downstream nonlinearities. Equivalently, it can be viewed as a Kronecker-/Gauss–Newton-like surrogate that keeps $\mathbf{H}_{\text{in}}$ but replaces the output factor by an identity (or constant) term, thereby ignoring how sensitive different output directions are under the task loss.

More faithful curvature surrogates that incorporate loss information and/or cross-layer interactions (e.g., approximations to the global Hessian or related second-order structure) can yield stronger compression performance, but typically require backward passes and higher calibration cost (van der Ouderaa et al., 2023; Tseng et al., 2025). Since our focus in this work is on the *representation/quantizer* rather than on increasingly powerful correction procedures, we evaluate all methods under the same local, GPTQ-like correction scheme (generalized to vector updates) to isolate gains attributable to the representation (Frantar et al., 2022). Global-Hessian-based corrections are orthogonal to our contribution, and combining them with our representation is a natural direction for future work (van der Ouderaa et al., 2023; Tseng et al., 2025).

## G Question: Is it better to construct spherical codes from a single Leech shell or a union of shells?

High–dimensional spherical codes derived from the Leech lattice can be constructed in multiple ways, depending on how lattice points are selected and normalized. In practice, two natural options arise: selecting points from a *single* Leech shell, or taking the *union of multiple shells* up to a radius threshold. Since these constructions differ in both cardinality growth and geometric diversity, it is not immediately clear which yields better angular uniformity per bit. Here, we empirically compare the two. Given a spherical code constructed from the Leech lattice in $\mathbb{R}^{24}$, is it better (in terms of angular uniformity per bit) to take a *single* shell $m$ or to take the *union of all shells up to* $m$?

For a finite set of unit vectors $\mathcal{C} \subset \mathbb{S}^{23}$, we define:

$$\text{bits} = \log_2 |\mathcal{C}|, \qquad q(x) = \arg \min_{y \in \mathcal{C} \setminus \{x\}} \arccos(x^\top y)/\pi,$$

where $q(x)$ is the nearest neighbor of $x$ in $\mathcal{C}$ under angular distance. We measure angular distance $D_{\mathbb{S}^{23}} : \mathbb{R}^{24} \times \mathbb{R}^{24} \to [0, 1]$, shorthand $D$, between a point and its closest radial neighbour:

$$D(x, q(x)) = \arccos(x^\top q(x))/\pi,$$

the angular distance between $x$ and its nearest neighbor. We report the distribution of $D(x, q(x))$ after sampling $x$ from a radially uniform source (such as $x \sim \mathcal{N}(0, \boldsymbol{I})$ normalized to unit norm).

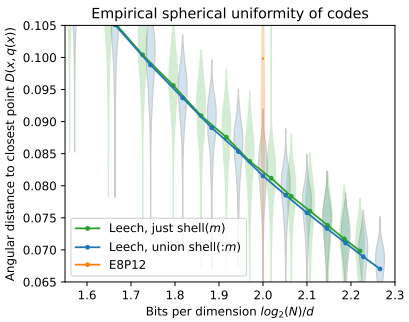



Figure 6: Empirical angular separation vs. rate. For each code, we show the distribution over nearest–neighbor angle against bits per dimension $\log_2(N)/d$. The two Leech-based constructions are traced by varying $m$ (single shell $m$ vs. union $1{:}m$). We compare to E8P12 and find that it lies above the Leech curves at the same bits (worse angular separation), while between the Leech variants the union has slightly superior angular distance.

We evaluate three code families as a function of bits: (1) Leech (single shell $m$), i.e., points from a single Leech shell projected to the unit sphere; (2) Leech (union up to $m$), i.e., the cumulative union of shells $1{:}m$, all normalized to unit norm; and (3) the E8P12 product code for reference, normalized to the unit sphere and stacked three times to allow comparison in 24 dimensions. Figure 6 shows violin plots of the resulting distributions over $D_{24}(x, q(x))$ versus bits for these three constructions. Empirically, we find that both Leech constructions, single shell $m$ and union up to shell $m$, perform similarly, with the union exhibiting slightly better code quality per bit as measured by angular uniformity.

**Key finding 1: Union of shells provide lowest angular distortion.** Our empirical results indicate that spherical codes formed by normalizing a union of Leech lattice shells yields a slightly better Gaussian rate–distortion curves compared to forming spherical codes from individual Leech lattice shells. We therefore adopt this approach in our method and recommend doing the same.

**Key finding 2: Single shell provides a simpler algorithm.** Although a union of shells results in slightly lower angular distortion, the difference between the two is small. From a hardware perspective, however, using a single shell offers significant advantages. In particular, a constant norm implies a fixed scaling between dot products, eliminating the need to rescale intermediate dot product results before aggregation (as in group-wise or block quantization), along with its associated complications.

# H Question: which has lowest distortion: 'spherical shaping' or 'shape–gain'?

We consider two natural ways to construct a finite representation from a finite set of spherically bounded Leech lattice points (or from a single lattice shell), namely *spherical shaping*, which uses the unnormalized points directly, and *shape–gain*, which separates direction and magnitude via a projected spherical code for shape and a scalar gain code. Define $C_{\text{spherical-shaping}} = \Lambda_{24}(m)$, the set of all lattice points in $\Lambda_{24}$ whose squared norm is at most $m$. We keep every lattice point inside a ball, and the resulting codebook directly inherits both its directions and its radii from the geometry of $\Lambda_{24}$, up to a channel-wise or global scale. Intuitively, this is a very simple construction: one hyperparameter $m$ determines both how far the code extends in radius and how many points it contains. The tradeoff between shape and gain is fixed implicitly by the lattice itself.

For shape–gain, we instead separate a vector into its magnitude and direction, $x = r\,u$ with $u = x/\|x\|$ and $r = \|x\|$, and quantize these two pieces independently. Formally, define the shape code $C_{\text{shape}} = \{\, x/\|x\| : x \in \Lambda_{24}(m) \,\}$, i.e., lattice points normalized to lie on the unit sphere, which gives a dense spherical code on $S^{23}$. The gain is represented by a scalar gain code $C_{\text{gain}}$ (e.g., matched to a $\chi_{24}$ distribution for Gaussian sources). The full shape–gain code is then $C_{\text{shape-gain}} = \{\, \hat{r}\,\hat{u} : \hat{u} \in C_{\text{shape}},\ \hat{r} \in C_{\text{gain}} \,\} = C_{\text{gain}} \times C_{\text{shape}}$. Intuitively, spherical shaping uses the raw lattice: you pick a lattice vector and both its direction and its length are "baked in." In contrast, shape–gain forces every codeword direction to live on the sphere (from the projected lattice) and assigns its radius explicitly through the gain quantizer.

Because shape–gain has two components, it requires a choice of how to allocate bits between the direction (shape) part and the radius (gain) part. High-resolution theory provides a useful heuristic: in $n$ dimensions, about $1/n$ of the total bits should be devoted to gain, meaning that for $n = 24$ one should allocate roughly $1/24$ of the rate to $C_{\text{gain}}$ and the remainder to $C_{\text{shape}}$. This gives a principled starting point and greatly reduces the effective search over hyperparameters. In contrast, the spherically bounded code $C_{\text{spherically-bounded}} = \Lambda_{24}(m)$ has only one hyperparameter $m$, which makes it easier to sweep but less expressive: it cannot optimize radius resolution independently of directional resolution.

We therefore compare these two constructions empirically under matched rate budgets: (i) the spherically bounded approach using $C_{\text{spherically-bounded}} = \Lambda_{24}(m)$, and (ii) the shape–gain approach using $C_{\text{shape-gain}} = C_{\text{gain}} \times C_{\text{shape}}$, where $C_{\text{shape}} = \{x/\|x\| : x \in \Lambda_{24}(m)\}$ and $C_{\text{gain}}$ is a $\chi_{24}$-matched scalar quantizer with approximately $1/24$ of the available bits. This allows us to isolate the effect of separating radius and direction from the effect of simply using a spherically truncated lattice.

| **Method** | **Code** | | **Bits/dim** | $\widehat{\textbf{MSE}}$ | $\widehat{\textbf{SQNR}}$ (bits) | **Ret**(%) |
|---|---|---|---|---|---|---|
| Leech (spherical bounding) | $\Lambda_{24}(13)$ | | 2.0 | 0.084 | 1.787 | 89.37 |
| Leech (shape-gain) | norm($\Lambda_{24}(13)$) + 0 $\chi$-gain bits | 2.00000 + 0.00000 | 2.0 | 0.085 | 1.782 | 89.12 |
| Leech (shape-gain) | norm($\Lambda_{24}(12)$) + 1 $\chi$-gain bits | 1.95833 + 0.04167 | 2.0 | 0.078 | 1.843 | **92.14** |
| Leech (shape-gain) | norm($\Lambda_{24}(11)$) + 2 $\chi$-gain bits | 1.91667 + 0.08333 | 2.0 | 0.080 | 1.825 | 91.24 |
| Leech (shape-gain) | norm($\Lambda_{24}(10)$) + 4 $\chi$-gain bits | 1.83333 + 0.16667 | 2.0 | 0.085 | 1.780 | 89.01 |

Table 8: Comparison between spherical shaping and shape–gain quantization at a fixed budget of 2 bits/dim. For shape–gain we vary the allocation between directional (shape) bits and radial (gain) bits while keeping the total rate fixed.

Taken together, the results in Table 8 reveal several consistent trends regarding the relative behaviour of spherical shaping and the shape–gain construction at a fixed rate of 2 bits/dim. In particular, although spherical shaping provides a clean geometric baseline, the shape–gain decomposition offers clear advantages in both distortion and retention.

- **Key finding 1.** Shape–gain can improve performance over spherical shaping.
- **Key finding 2.** The $1/n$ gain-bit heuristic is reasonable but not always optimal at finite rate. High-resolution theory in $n$ dimensions suggests allocating approximately $1/n$ of the bits to the gain; for $n = 24$ this corresponds to 2 gain bits (0.083 bits/dim), whereas our empirical optimum is 1 gain bit (0.041 bits/dim).
- **Key finding 3.** For the Leech-lattice shape–gain code at 2 bits/dim, we recommend using 1 or 2 gain bits per vector of length 24 (0.041-0.08333 bits/dim), though practitioners should consider running similar sweeps when targeting other bitrates to obtain the best quantization performance.

# I Spherical GPTQ and Rotation/Hadamard-free PTQ

This appendix provides full derivations and pseudocode for *Spherical GPTQ*, which is introduced in the main paper in subsection 3.3. Applying vector or shape–gain quantization inside GPTQ immediately raises a practical question: *what do we do with magnitudes?* Given a weight block (or row/column vector) $w$, a shape–gain view decomposes it into a direction and a gain, $w = su$ with $u \in \mathbb{S}^{d-1}$ and $s = \|w\|_2$. After quantizing the direction (the "shape"), there are multiple reasonable choices for the gain before also quantizing it: (i) keep the original gain ($s = \|w\|_2$), (ii) choose $s$ to minimize the post-quantization MSE (optionally Hessian-weighted), and (a) quantize $s$ immediately per GPTQ block, or (b) keep $s$ in high precision during GPTQ and only quantize/solve for it after all blocks are processed. While these choices are often treated as engineering details, in a Hessian-aware procedure like GPTQ they *change the effective update rule*: the residual corrections depend on the magnitude of previously-quantized blocks, hence different gain policies induce different error-feedback dynamics.

**Vector GPTQ (Euclidean)** Generalizing GPTQ to vector quantization can be viewed as approximately solving, block-by-block or coordinate-by-coordinate,

$$\min_{\hat{w}\in\mathcal{C}} \frac{1}{2}(w-\hat{w})^\top H(w-\hat{w}), \tag{16}$$

by greedily quantizing a subset of coordinates/blocks and propagating the resulting error to the remaining ones using a Cholesky factor of $H^{-1}$. Algorithm 1 (Vector GPTQ) instantiates this Euclidean mechanism: after quantizing the current coordinate (or vector chunk), the residual is updated by a Hessian-preconditioned correction on the yet-unquantized coordinates.

When the quantizer $\mathcal{Q}$ is primarily *directional* (e.g., shape–gain codes or lattice-based VQ used as a spherical code), the Euclidean GPTQ inner loop can exhibit an important interaction with the Hessian-weighted error-feedback updates. Although quantization always induces both angular and radial error, we empirically find that the *radial component* can become disproportionately large across greedy block steps.

**Spherical GPTQ (Riemmanian)** Motivated by the standard "gain reset" step in classical shape–gain quantization, where one quantizes direction and then matches the original norm, we introduce a simple drop-in modification to the GPTQ loop: after quantizing a block, we *retract* the candidate back to the original constant-norm manifold (keeping high-precision), preserving direction while eliminating norm error. For a vector $w$ and its quantized candidate $\tilde{w} = \mathcal{Q}(w)$, the spherical retraction is

$$\hat{w}_{\mathrm{sph}} \;=\; \Pi_{\|w\|}(\tilde{w}) \;:=\; \frac{\|w\|_2}{\|\tilde{w}\|_2}\,\tilde{w}. \tag{17}$$

In our block setting, quantization is performed row-wise on each row-block $W_{i,B} \in \mathbb{R}^{|B|}$, and we apply the same retraction per row:

$$\hat{W}_{i,B} \;=\; \frac{\|W_{i,B}\|_2}{\|\tilde{W}_{i,B}\|_2}\,\tilde{W}_{i,B}.$$

Geometrically, equation 17 is a *retraction* onto a product of spheres (one sphere per row-block), turning the otherwise Euclidean GPTQ inner loop into a *Riemannian* step: quantization selects a direction (a point in the tangent space proxy), and the retraction maps it back onto the fixed-norm manifold. This motivates the name *Spherical GPTQ*. Importantly, the retraction is codebook-agnostic: any quantizer that produces a candidate $\tilde{w}$ can be followed by $\Pi_{\|w\|}(\cdot)$ as a drop-in modification.

Operationally, this corresponds to keeping gains in high precision during the GPTQ loop: we quantize primarily *directions*, set the block norm to match the original, and then compute the GPTQ error using the retracted weights. Concretely, for each block $B$ we quantize to obtain $\tilde{R}_{:,B}$, retract row-wise to $R_{:,B}$ as above, form the residual $E_{:,B} = W_{:,B} - R_{:,B}$, and propagate this residual to the remaining blocks using the standard GPTQ correction.

After all blocks are quantized (directions fixed), we optionally refine magnitudes by solving for optimal gains (*group scales*) in closed form. The post-processing step in Algorithm 2 forms, for each row $i$, a small linear system over blocks with

$$M_{i,pq} = R_{i,B_p} H_{B_p B_q} R_{i,B_q}^\top, \qquad r_{i,p} = R_{i,B_p} H W_{i,:}^\top,$$

and solves $M_i s_i = r_i$, followed by $R_{i,B} \leftarrow s_{i,B} R_{i,B}$. Conceptually, this computes the Hessian-weighted best gains given fixed directions: we set directions via (spherical) GPTQ, then perform a single global gain update at the end.

### I.1 Pseudocode for Vector GPTQ (Euclidean) and Spherical GPTQ (Riemmanian)

**Algorithm 2** Vector GPTQ (Euclidean)

**Require:** $\mathbf{W} \in \mathbb{R}^{k \times d}$, SPD $\mathbf{H} \in \mathbb{R}^{d \times d}$, blocks $Q_1, \ldots, Q_m$ (processing order), quantizer $\mathcal{Q}$

$\mathbf{U} \leftarrow \mathrm{chol}(\mathbf{H}^{-1})^\top$ ▷ $\mathbf{H}^{-1} = \mathbf{U}^\top \mathbf{U}$, $\mathbf{U}$ upper-triangular
$\widetilde{\mathbf{W}} \leftarrow \mathbf{W}$
**for** $t = 1$ **to** $m$ **do**
  $Q \leftarrow Q_t, \quad R \leftarrow \bigcup_{u>t} Q_u$
  $\hat{\mathbf{W}}_{:,Q} \leftarrow \mathcal{Q}(\widetilde{\mathbf{W}}_{:,Q})$
  $\mathbf{E}_{:,Q} \leftarrow \widetilde{\mathbf{W}}_{:,Q} - \hat{\mathbf{W}}_{:,Q}$
  $\widetilde{\mathbf{W}}_{:,Q} \leftarrow \hat{\mathbf{W}}_{:,Q}$
  **if** $R \neq \emptyset$ **then**
    $\widetilde{\mathbf{W}}_{:,R} \leftarrow \widetilde{\mathbf{W}}_{:,R} - \big(\mathbf{E}_{:,Q} \mathbf{U}_{QQ}^{-1}\big) \mathbf{U}_{QR}$ ▷ right-side triangular solve in $\mathbf{U}_{QQ}$
  **end if**
**end for**
**return** $\hat{\mathbf{W}} \leftarrow \widetilde{\mathbf{W}}$

**Algorithm 3** Spherically Projected GPTQ with Group-scales (Riemannian)

**Require:** $\mathbf{W} \in \mathbb{R}^{k \times d}$, SPD $\mathbf{H} \in \mathbb{R}^{d \times d}$, blocks $Q_1, \ldots, Q_m$, quantizer $\mathcal{Q}$, damping $\lambda \geq 0$

$\mathbf{U} \leftarrow \mathrm{chol}(\mathbf{H}^{-1})^\top$
$\widetilde{\mathbf{W}} \leftarrow \mathbf{W}$
**for** $t = 1$ **to** $m$ **do**
  $Q \leftarrow Q_t, \quad R \leftarrow \bigcup_{u>t} Q_u$
  $\widetilde{\mathbf{W}}_{:,Q} \leftarrow \frac{\|\mathbf{W}_{:,Q}\|_F}{\|\mathcal{Q}(\widetilde{\mathbf{W}}_{:,Q})\|_F} \mathcal{Q}(\widetilde{\mathbf{W}}_{:,Q})$
  $\mathbf{E}_{:,Q} \leftarrow \mathbf{W}_{:,Q} - \widetilde{\mathbf{W}}_{:,Q}$
  **if** $R \neq \emptyset$ **then**
    $\widetilde{\mathbf{W}}_{:,R} \leftarrow \widetilde{\mathbf{W}}_{:,R} - \big(\mathbf{E}_{:,Q} \mathbf{U}_{QQ}^{-1}\big) \mathbf{U}_{QR}$
  **end if**
**end for**
▷ Final per-row block scale refinement in the Hessian metric
**for** $i = 1$ **to** $k$ **do**
  $\mathbf{M}_i[p, q] \leftarrow \widetilde{\mathbf{W}}_{i,Q_p} \mathbf{H}_{Q_p Q_q} \widetilde{\mathbf{W}}_{i,Q_q}^\top \quad \forall\, p, q \in \{1, \ldots, m\}$
  $\mathbf{r}_i[p] \leftarrow \widetilde{\mathbf{W}}_{i,Q_p} \mathbf{H}_{Q_p,:} \mathbf{W}_{i,:}^\top \quad \forall\, p \in \{1, \ldots, m\}$
  $\mathbf{s}_i \leftarrow (\mathbf{M}_i + \lambda \mathbf{I})^{-1} \mathbf{r}_i$
  **for** $p = 1$ **to** $m$ **do**
    $\widetilde{\mathbf{W}}_{i,Q_p} \leftarrow \mathbf{s}_i[p]\ \widetilde{\mathbf{W}}_{i,Q_p}$
  **end for**
**end for**
**return** $\hat{\mathbf{W}} \leftarrow \widetilde{\mathbf{W}}$

### I.2 Ablation: Ablating need for Hadamard rotations under Spherical GPTQ

Table 9: Extended ablation study with and without Hadamard rotations, evaluated on wikitext-2 perplexity (PPL) and downstream tasks (CSR and MMLU). LLVQ consistently outperforms standard VQ approaches in performance against bits per weight (BPW).

| **Method** (all results in this table are without finetuning) | **Dim** | **BPW** | **Hadamard** | **Effective Correction** | Llama-2 7B **Wiki** ↓ | **MMLU** ↑ | **CSR** ↑ |
|---|---|---|---|---|---|---|---|
| Baseline | 1 | 16 | - | | 5.12 | 45.7 | 70.4 |
| Integer (GPTQ) | 1 | 2 | No Rotation | GPTQ | 3411.6 | 26.6 | 39.7 |
| Integer (Quarot) | 1 | 2 | Input | GPTQ | 41.87 | 27.0 | 41.7 |
| Integer | 1 | 2 | Input + Output | GPTQ | 37.83 | 26.1 | 48.4 |
| E8P | 8 | 2 | No Rotation | GPTQ | 105.98 | 24.8 | 44.9 |
| E8P | 8 | 2 | Input | GPTQ | 9.24 | 31.0 | 59.8 |
| E8P (Quip#) | 8 | 2 | Input + Output | GPTQ | 7.96 | 30.5 | 61.4 |
| LLVQ [spherical shaping] | 24 | 2 | No Rotation | GPTQ | 191.90 | 24.0 | 33.5 |
| LLVQ [spherical shaping] | 24 | 2 | Input | GPTQ | **6.80** | **35.1** | **65.4** |
| LLVQ [spherical shaping] | 24 | 2 | Input + Output | GPTQ | **7.61** | **33.4** | **62.1** |
| LLVQ [spherical shaping] (forced angular correction) | 24 | 2 | No Rotation | Spherical GPTQ | **6.90** | **37.4** | **63.8** |
| LLVQ [spherical shaping] (forced angular correction) | 24 | 2 | Input | Spherical GPTQ | **6.70** | **35.1** | **65.4** |
| LLVQ [spherical shaping] (forced angular correction) | 24 | 2 | Input + Output | Spherical GPTQ | **6.75** | **36.9** | **63.8** |
| LLVQ [shape-gain w/ 2 bit gain] (forced Euclidean correction) | 24 | 2 | No Rotation | GPTQ | 13.17 | 26.5 | 58.5 |
| LLVQ [shape-gain w/ 2 bit gain] (forced Euclidean correction) | 24 | 2 | Input | GPTQ | **7.28** | **34.1** | **62.8** |
| LLVQ [shape-gain w/ 2 bit gain] (forced Euclidean correction) | 24 | 2 | Input + Output | GPTQ | **7.31** | **35.3** | **62.8** |
| LLVQ [shape-gain w/ 2 bit gain] | 24 | 2 | No Rotation | Spherical GPTQ | **7.27** | **29.8** | **61.5** |
| LLVQ [shape-gain w/ 2 bit gain] | 24 | 2 | Input | Spherical GPTQ | **6.90** | **36.0** | **63.6** |
| LLVQ [shape-gain w/ 2 bit gain] | 24 | 2 | Input + Output | Spherical GPTQ | **6.83** | **34.9** | **64.6** |

Empirically, we find that:

1. **Key finding 1. Spherical GPTQ can improve results over Euclidean GPTQ.** Spherical GPTQ serves as a simple drop-in modification to standard GPTQ that improves quantization quality by explicitly removing radial error through norm preservation. By projecting quantized vectors back onto the constant-norm manifold, it ensures that residual corrections focus on the most relevant degrees of freedom. Empirically, this can lead to improved performance across quantizers without requiring changes to the underlying codebook, demonstrating that spherical projection itself is a robust and broadly applicable enhancement to GPTQ.
2. **Key finding 2. Spherical GPTQ is most effective with low angular distortion codes.** Empirically, we find that quantizers with low angular distortion (such as LLVQ) achieve the best overall performance. Notably, improvements correlate more strongly with reduced angular distortion than with low MSE under a Gaussian source model. We find that allocating bits to gain becomes largely unnecessary: capacity is better spent entirely on encoding directions. We observe an the empirical shift from using explicit gain quantization (e.g., 2-bit gains under Euclidean GPTQ) to effectively zero-bit gains under Spherical GPTQ, since norm preservation combined with low angular distortion makes additional gain resolution redundant.
3. **Key finding 2. LLVQ outperforms under both Euclidean GPTQ and Spherical GPTQ.** LLVQ achieves the best accuracy–compression trade-offs compared to baselines regardless of whether the correction step is Euclidean (standard GPTQ) or spherical, indicating that its gains stem from the representation rather than a particular correction heuristic. The strong results under *forced* mismatched corrections further suggest that LLVQ is robust to the specific correction dynamics used.
4. **Key finding 3. Diminishing returns from Hadamard rotations.** With spherical corrections, the incremental benefit of Hadamard preprocessing is substantially reduced: once radial errors are eliminated and the residual updates focus on directions, the remaining distortion is dominated by the intrinsic angular resolution of the codebook rather than by coordinate anisotropy that rotations would mitigate.
5. **Key finding 4. Bit allocation shifts toward directions ("0-bit gains").** When the underlying quantization grid has low angular distortion (e.g., strong vector/lattice codes), spherical GPTQ makes allocating bits to gains less attractive. In this regime, we observe that the optimal shape–gain budget shifts from explicit gain quantization (e.g., "2-bit gains") toward effectively *no gain bits*: all capacity is used to encode directions, while magnitudes are handled by the high-precision spherical constraint during GPTQ and a single closed-form Hessian update afterward.

# J Additional results on smaller Llama-3.2 1B and Llama-3.2 3B models

Table 10: Comparison of performance after quantizing smaller Llama-3.2 1B and Llama 3.2 3B language models using different quantization methods, evaluated by Wikitext-2 (Wiki) perplexity at 4096 context length and downstream task performance (CSR and MMLU) on own consistent training pipeline. LLVQ consistently outperforms standard vector quantization approaches.

| | Fine- | | | Llama-3.2 1B | | | Llama-3.2 3B | | |
|---|---|---|---|---|---|---|---|---|---|
| **Method** (same pipeline) | **tuned** | **BPW** | **Hadamard** | **Wiki↓** | **MMLU↑** | **CSR↑** | **Wiki↓** | **MMLU↑** | **CSR↑** |
| Baseline | - | 2 | Input + Output | **12.14** | **46.2** | **61.8** | **10.48** | **59.8** | **67.6** |
| GPTQ + Rotation (Quarot) | No | 2 | Input + Output | 1158.7 | 24.2 | 36.9 | | | |
| E8P/Quip# (Euclidean GPTQ) | No | 2 | Input + Output | 38.07 | 26.4 | 49.4 | 30.77 | 36.4 | 54.9 |
| E8P (+Spherical GPTQ) | No | 2 | Input + Output | 24.21 | 31.29 | 52.1 | 20.34 | 42.3 | 58.3 |
| LLVQ (spherical shaping, Euclidean GPTQ) (**ours**) | No | 2 | Input + Output | 25.36 | 33.1 | 53.1 | 18.40 | 46.0 | 60.5 |
| LLVQ (spherical shaping, Spherical GPTQ) (**ours**) | No | 2 | Input + Output | 23.76 | 30.2 | 53.5 | 19.70 | 47.1 | 59.2 |
| LLVQ (shape-gain w/ 0 bit gain, Euclidean GPTQ) (**ours**) | No | 2 | Input + Output | 47.78 | 25.1 | 47.5 | 32.20 | 37.9 | 52.0 |
| LLVQ (shape-gain w/ 0 bit gain, Spherical GPTQ) (**ours**) | No | 2 | Input + Output | **21.36** | **32.5** | **54.1** | **15.93** | **48.6** | **60.0** |
| LLVQ (shape-gain w/ 2 bit gain, Euclidean GPTQ) (**ours**) | No | 2 | Input + Output | 26.85 | 27.7 | 51.3 | 22.96 | 44.8 | 58.2 |
| LLVQ (shape-gain w/ 2 bit gain, Spherical GPTQ) (**ours**) | No | 2 | Input + Output | 23.08 | 31.1 | 52.4 | 18.81 | 45.6 | 59.1 |
| E8P/Quip# | Yes | 2 | Input + Output | 16.67 | 26.5 | 52.0 | 11.13 | 42.1 | 61.0 |
| LLVQ (spherical shaping, Euclidean GPTQ) (**ours**) | Yes | 2 | Input + Output | 14.25 | 32.7 | 54.4 | 10.10 | 49.3 | 62.4 |
| LLVQ (spherical shaping, Spherical GPTQ) (**ours**) | Yes | 2 | Input + Output | 14.18 | 28.6 | 55.1 | 10.10 | 50.2 | **63.0** |
| LLVQ (shape-gain w/ 0 bit gain, Euclidean GPTQ) (**ours**) | Yes | 2 | Input + Output | 16.92 | 26.1 | 51.4 | 11.01 | 43.4 | 58.9 |
| LLVQ (shape-gain w/ 0 bit gain, Spherical GPTQ) (**ours**) | Yes | 2 | Input + Output | **13.66** | **34.5** | **55.3** | **9.73** | **51.3** | **62.9** |
| LLVQ (shape-gain w/ 2 bit gain, Euclidean GPTQ) (**ours**) | Yes | 2 | Input + Output | 14.94 | 27.6 | 53.6 | 10.40 | 48.9 | 61.9 |
| LLVQ (shape-gain w/ 2 bit gain, Spherical GPTQ) (**ours**) | Yes | 2 | Input + Output | 14.42 | 33.3 | 54.4 | 10.17 | 49.2 | 62.9 |