Leech Lattice Vector Quantization for Efficient LLM Compression

Tycho F. A. van der Ouderaa [1] **Mart van Baalen** [1] **Paul Whatmough** [1] **Markus Nagel** [1]

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. Together this yields a practical algorithm, namely *Leech Lattice Vector Quantization* (LLVQ). LLVQ delivers state-of-the-art LLM quantization performance, outperforming recent methods such as Quip#, QTIP, and PVQ. These results highlight the importance 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 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

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

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

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.

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*, enabling shape-gain quantization with the Leech lattice.
3. Propose a fully parallelizable kernel for fast dequantization of spherically bounded Leech lattice points using fast modulo arithmetic.
4. Scientific findings on Gaussian source: establish that union of shells achieves lower angular distortion than using single shells (App. F), and demonstrate that Leech shape-gain codes can improve signal-to-noise over spherical shaping (App. G).

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). 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* Λ_{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 $b_1, \ldots, b_n$:

$$\Lambda = \left\{ \sum_{i=1}^{n} k_i b_i \;\middle|\; k_i \in \mathbb{Z} \right\}. \tag{1}$$

The Leech lattice Λ_{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 8.3×10^{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 Λ_{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 Λ_{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

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

The minimal squared norm of Λ_{24} is 4, corresponding to m=2. The shells are disjoint and partition the full lattice:

$$\Lambda_{24} = \bigcup_{m=2}^{\infty} \text{Shell}(m). \tag{3}$$

with ball-cut including points up to squared norm M=2m,

$$\Lambda_{24}(M) = \bigcup_{m=2}^{M} \text{Shell}(m). \tag{4}$$

Since shells are disjoint, the cardinality is a cumulative sum:

$$N(M) = |\Lambda_{24}(M)| = \sum_{m=2}^{M} n(m), \tag{5}$$

where $n(m) = |\text{Shell}(m)|$. 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 $n(m)$, 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 any lattice point 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. G, 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. F 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 in the limit of infinite bitrate, cumulative unions of shells achieve strictly lower distortion than the rate–distortion curve of individual shells. This observation motivates our use of cumulative shell unions whenever we use shape-gain.

Table 1. Shell structure of the Leech lattice Λ_{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

3. Leech Lattice Vector Quantization (LLVQ)

We build upon the fast nearest neighbour search for single Leech lattice shells by Adoul & Barth (1988). Their 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 Eq. 18 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 quantization), discussed in §A.4; and (jii) 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 §3.1, §A.5).

3.1. Indexing Scheme

While the original method by Adoul & Barth (1988) introduced an elegant search algorithm over leaders, placements, and signs, it does not provide a bijective indexing scheme. For quantization, such a mapping is essential: 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 degrees of freedom.

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

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

$$\widehat{\mathrm{MSE}} = \frac{1}{n}\sum_{i=1}^{n} \|w_i - q(w_i)\|_2^2 / D \tag{6}$$

per weight, $D = \dim(w_i)$, and the empirical SQNR:

$$\widehat{\mathrm{SQNR}}_{\mathrm{bits}} = -\frac{1}{2}\log_2\big(\widehat{\mathrm{MSE}}\big). \tag{7}$$

For an ideal Gaussian source, the Shannon rate–distortion function gives the minimum achievable MSE at rate R as

$$\mathrm{MSE}^*(R) = \sigma^2\, 2^{-2R}. \tag{8}$$

For our unit-variance source ($\sigma^2 = 1$), this becomes $\mathrm{MSE}^*(R) = 2^{-2R}$. Using the same convention for SQNR in bits, the optimal (Shannon) SQNR is

$$\mathrm{SQNR}^*_{\mathrm{bits}}(R) = -\frac{1}{2}\log_2\big(2^{-2R}\big) = R. \tag{9}$$

Thus, following this convention, the Shannon limit for a Gaussian source corresponds to the straight line $\mathrm{SQNR}^*_{\mathrm{bits}}(R) = R$ (i.e., a $y = x$ line) in the SQNR–versus–rate plot. To convert to the other common base-10 decibel (dB) unit, multiply $\mathrm{SQNR}_{\mathrm{bits}}$ with $20\log_{10}(2) \approx 6.0206$.

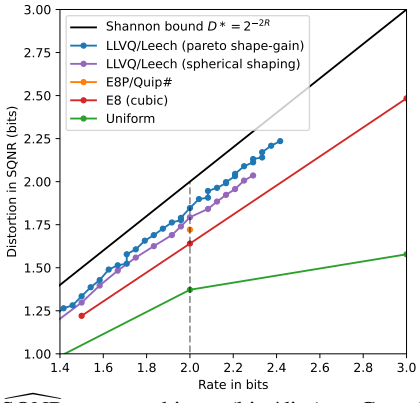

Figure 1. $\widehat{\mathrm{SQNR}}_{\mathrm{bits}}$ versus bitrate (bits/dim) on Gaussian source.

4. Performance on Idealistic Gaussian Source

4.1. Measured SQNR and Retention

Due to structural constraints, practical quantizers inevitably fall below the theoretical limit. Figure 1 shows the resulting 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.

To quantify how close a given method comes to the Shannon bound, we report a retention score that measures the fraction of the optimal SQNR achieved at a given bitrate. At a fixed bitrate (e.g., $R = 2$ bits/dim), we report the empirical MSE and SQNR (in bits), and quantify how close the quantizer is to the Shannon limit using a *retention* score:

$$\mathrm{Ret}(\%) = \frac{\widehat{\mathrm{SQNR}}_{\mathrm{bits}}}{\mathrm{SQNR}^*_{\mathrm{bits}}(R)} \times 100 = \frac{\widehat{\mathrm{SQNR}}_{\mathrm{bits}}}{R} \times 100, \tag{10}$$

since conveniently $\mathrm{SQNR}^*_{\mathrm{bits}}(R) = R$, under our convention of measuring signal-to-noise in bits.

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.

Table 2. 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.

			Llama-2 7B			Llama-3 8B			Ministral-3 8B instruct			Qwen-3 4B			Qwen-3 8B		
Method (same pipeline)	**Fine-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**

Table 3. Information retention at 2 bits/dim on Gaussian source.

Method	**Dim**	$\widehat{\mathbf{MSE}}$ ↓	$\widehat{\mathbf{SQNR}}$ (bits) ↑	**Ret** (%) ↑
Uniform	1	0.15	1.37	69
Lloyd-Max	1	0.12	1.53	77
E8 coset code (Stellat)	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

5. LLM Quantization Results

5.1. Experimental Set-up

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 introduce relatively little risk of overfitting.

5.2. 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 2 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.

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.

5.3. Impact of Hadamard Rotations

Following recent quantization literature (Ashkboos et al., 2024; Chee et al., 2023), we reparameterize the weights of each linear layer, using randomized Hadamard transforms on the input and output as in (Tseng et al., 2024a). The goal of such transformations is to preserve the model's functional output while making the marginal weight statistics more Gaussian-like, reducing outliers, and making them substantially easier to quantize. Table 5 examines the impact of Hadamard-based input/output rotations across three quantizer families: scalar integer methods (GPTQ/Quarot), the E8P/Quip# codebook, and the proposed LLVQ quantizers based on the Leech lattice. Across all methods, the "Input + Output" rotation consistently delivers the best downstream performance, confirming the value of decorrelating activation and weight spaces prior to quantization. Scalar integer quantization benefits the most from rotations due to the minimal flexibility of 1D codebooks.

Interestingly, although rotations consistently improve performance, LLVQ with shape–gain even *without* rotations performs remarkably well, even surpassing the performance of Quip#'s E8P codebook *with* rotations, using consistent quantization pipeline without finetuning. This suggests that higher-dimensional vector quantization intrinsically reduces the relience on rotational preprocessing. This is especially relevant because, although rotational preprocessing can often be fused or merged into the model (Ashkboos et al., 2024; van Breugel et al., 2025), this is not always possible. In such cases, these transformations must be applied online, adding latency and computational overhead. The results therefore indicate that higher-dimensional vector quantization methods, such as LLVQ, can mitigate or even eliminate the need for expensive rotational preprocessing (e.g., online Hadamards), while still achieving state-of-the-art quantization quality.

5.4. 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 fur-

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**

Table 5. 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).

				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	53.5
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**

ther broadens the analysis to include results previously reported in the literature. Although published baselines such as OmniQ, AQLM, Quip#, and QTIP may slightly differ in training conditions, calibration set composition, and dataset sizes, these comparisons remain highly meaningful, as they contextualize 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 1M tokens. This minimal adaptation reliably improves perplexity, MMLU, and CSR across quantization methods, acting as a lightweight alignment step.

Notably, LLVQ maintains clearly better model performance after quantization compared to the strongest PTQ 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 using 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.

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 Breugel, B., Bondarenko, Y., Whatmough, P., and Nagel, M. Fptquant: Function-preserving transforms for llm quantization. *arXiv preprint arXiv:2506.04985*, 2025.

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.

A. Constructing Λ_{24} from the Extended Golay Code

For efficient search in Λ_{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 $\{8, 12, 16, 24\}$.

Vectors in Λ_{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^{even} or L^{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 $19{,}560$ lattice points, of which the first $1{,}104$ belong to the first class, consisting of vectors with four coordinates equal to 2 and twenty-two equal to 0. The (nearest neighbour) quantization algorithm and dequantization algorithm exploit the hierarchy and completely avoid having to explicitly enumerate lattice points.

Integer-coordinate formulation. The Leech lattice can be defined as the scaled union of integer-valued vectors:

$$L^{\text{int}} = L^{\text{even}} \cup L^{\text{odd}}, \qquad \Lambda_{24} = \frac{1}{\sqrt{8}} L^{\text{int}} \subset \mathbb{R}^{24}. \tag{11}$$

which we call the *even* and *odd* cosets,

$$L^{\text{even}} = \left\{ x \in \mathbb{Z}^{24} \;\middle|\; \begin{array}{l} \text{(i) } x_i \equiv 0 \pmod 2 \\ \text{(ii) } (x/2) \bmod 2 \in \mathcal{G}_{24} \\ \text{(iii) } \sum_i x_i \equiv 0 \pmod 8 \end{array} \right\} \tag{12}$$

$$L^{\text{odd}} = \left\{ x \in \mathbb{Z}^{24} \;\middle|\; \begin{array}{l} \text{(i) } x_i \equiv 1 \pmod 2 \\ \text{(ii) } ((x-\mathbf{1})/2) \bmod 2 \in \mathcal{G}_{24} \\ \text{(iii) } \sum_i x_i \equiv 4 \pmod 8 \end{array} \right\} \tag{13}$$

where the roles of the three constraints are:

(i) *Parity constraint:* forces vector in even or odd coset.

(ii) *Golay constraint:* ensures that the mod–2 reduction of the halved vector matches a codeword 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, unimodular, and has minimum squared norm 4. It is equivalent to the classic description

$$\frac{1}{\sqrt{8}}(2x + y), \qquad x \in \mathbb{Z}^{24},\ y \in \mathcal{G}_{24}, \tag{14}$$

subject to $\sum_i (2x_i + y_i) \equiv 0 \pmod 4$. While this form encodes the above (i)–(iii) conditions implicitly, our expanded formulation exposes them which is essential for the indexing and search procedures that we develop next.

A.1. Class Structure and Leaders

As mentioned, each shell $\text{Shell}(m)$ can be decomposed into different *classes*. A class is the set of all lattice points obtainable from one another by the allowed permutations of coordinates and sign flips.

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

$$\{\, a_1^{p_1},\, a_2^{p_2},\, \ldots,\, a_k^{p_k} \,\}, \qquad \sum_i p_i = 24, \tag{15}$$

where $a_i \in \frac{1}{\sqrt{8}}\mathbb{Z}$ are the distinct coordinate levels and the exponents p_i record their multiplicities. This multiset fully specifies the coordinate composition of vectors in the class.

A.2. Even and Odd Classes

Each class lies entirely in either L^{even} or L^{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

$$F_0(c) = \{\, i : c_i = 0 \,\}, \qquad F_1(c) = \{\, i : c_i = 1 \,\}. \tag{16}$$

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), so odd classes have maximal flexibility in Golay-based placement but essentially no freedom in choosing signs.

A.3. Subclass Counts and Class Cardinality

The cardinality of each class follows directly from combinatorial considerations on 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 $F_1(c)$ (here the q_j are the within-subset multiplicities induced by the placement). Altogether, the class cardinality is

$$n(m) = |\text{Shell}(m)| = A \cdot 2^B \cdot \frac{24!}{\prod_{i=1}^{k} p_i!} \cdot \frac{1}{\prod_j q_j!}. \tag{17}$$

matching the hierarchical indexing used in our algorithms.

A.4. Extending to Spherical Search in $\Lambda_{24}(m)$

In the original single-shell setting of Adoul–Barth, all candidates share the same norm, so Euclidean and angular distances induce the same ordering:

$$\|x - v\|^2 = \|x\|^2 + \|v\|^2 - 2\langle x, v\rangle, \quad \text{(fixed } \|v\|\text{)} \tag{18}$$

Consequently, ranking by $-\langle x, v\rangle$ is equivalent to minimizing $\|x - v\|$. For LLVQ we generalize the search to multiple shells of $\Lambda_{24}(m)$. Across shells, candidate norms vary and the Euclidean vs. angular equivalence no longer holds. We therefore support two scoring modes: *Euclidean distance* (suitable for spherical shaping), and *angular distance* via cosine similarity (suitable for shape–gain). This can be implemented by normalizing both the input and the candidates, $\hat{x} = x/\|x\|$ and $\hat{v} = v/\|v\|$, and maximizing $\langle \hat{x}, \hat{v}\rangle$.

A.5. Dequantizer

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

$$\text{Dequantizer} : \{1, \ldots, N(m)\} \to L^{\text{int}}(m) \subset \mathbb{Z}^{24}. \tag{19}$$

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, the 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 successive modulo and integer-division steps:

$$\begin{aligned} r &= I_{\text{class}} \bmod A, \qquad & I' &= \lfloor I_{\text{class}}/A \rfloor \,, \\ s &= I' \bmod 2^B, \qquad & I'' &= \left\lfloor I'/2^B \right\rfloor \,, \end{aligned} \tag{20}$$

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., as a CUDA kernel).

B. Pseudo-code

B.1. Overall Shape-gain Quantization with Hessian Corrections

Algorithm 1 Overall Shape-gain Quantization with Hessian Corrections

for each layer l in model **do**
  Estimate Hessian matrix $\boldsymbol{H}$ using layer inputs $\boldsymbol{X}$ (subsection E.2)
  Partition each row of weight matrix $W_l \in \mathbb{R}^{N \times D}$ into G partitions: $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_G$ where $G = D/24$.
  for group index $g = 1$ to G **do**
    Extract submatrix $V_g \in \mathbb{R}^{N \times 24}$ containing all $\mathbf{v}_g$ across rows
    for each vector $\mathbf{v}_g^{(i)}$ in V_g **do** (in parallel)
      Compute direction $\hat{\mathbf{v}}_g^{(i)} = \mathbf{v}_g^{(i)} / \|\mathbf{v}_g^{(i)}\|$
      Compute scale $s_g^{(i)} = \|\mathbf{v}_g^{(i)}\|$
      Find nearest Leech lattice point $\mathbf{q}_g^{(i)}$ maximizing $\cos(\theta) = \hat{\mathbf{v}}_g^{(i)\top} \mathbf{q}_g^{(i)}$ (**??**)
      Optionally quantize $s_g^{(i)}$ using scalar quantizer
      Update residual: $\mathbf{v}_{g,\text{res}}^{(i)} = \mathbf{v}_g^{(i)} - s_g^{(i)} \cdot \mathbf{q}_g^{(i)}$
    end for
    In-place adjust W_l using residual V_g and Hessian $\boldsymbol{H}$ (Hessian correction, **??**)
  end for
end for

C. 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. G.

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

$$\mathbf{x} = r\,\mathbf{u}, \qquad r = \|\mathbf{x}\| \in \mathbb{R}, \;\; \mathbf{u} = \mathbf{x}/\|\mathbf{x}\| \in \mathbb{S}^{n-1}. \tag{21}$$

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.

D. 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.

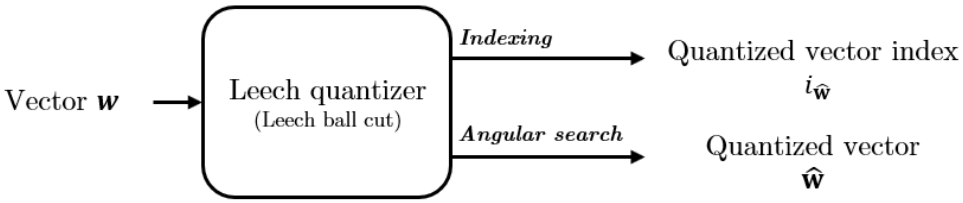

Figure 2. Spherical shaping: ball cut of the Leech lattice with nearest-neighbor selection over $\Lambda_{24} \cap \mathbb{B}(0, R)$.

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.

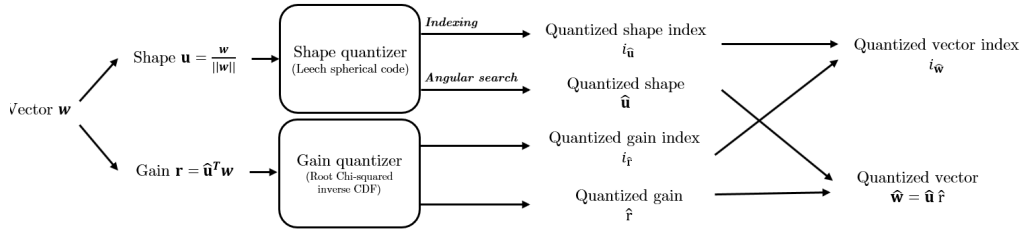

Figure 3. Shape–gain with independent quantization of shape $\mathbf{s}$ and gain g, producing an index in $[1, N(m)]$ and reconstruction $\hat{\mathbf{w}} = \hat{g}\,\hat{\mathbf{s}}$.

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.

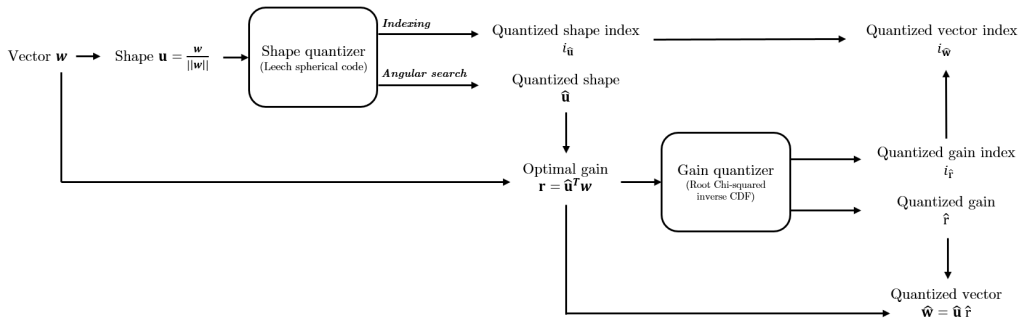

Figure 4. Shape–gain with optimal scales: post-shape gain optimization via a shape-conditioned gain quantizer.

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.

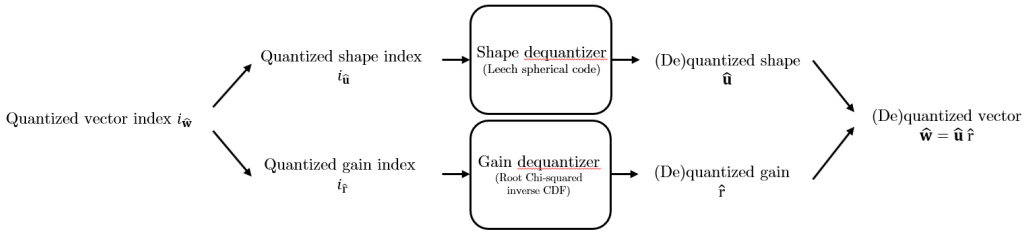

Figure 5. Dequantizer mapping the integer index $i_{\hat{\mathbf{w}}} \in [1, N(m)]$ to the reconstruction $\hat{\mathbf{w}} \in \mathbb{R}^d$.

A detailed comparison between spherical shaping and shape–gain is provided in Appendix G. 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.

E. Algorithm specification

E.1. Optimal scales under shape-gain

An additional benefit of spherical vector quantization (shape-gain) is that it renders the quantizer scale invariant: $q(s\boldsymbol{w}) = q(\boldsymbol{w})$ for all $s \in \mathbb{R}$, so the scaled grid $q'(\boldsymbol{w}) = \beta \cdot q(\boldsymbol{w}/\beta)$ simplifies to $q'(\boldsymbol{w}) = \beta \cdot q(\boldsymbol{w})$. This removes the need for line search and yields closed-form scale solutions. For the optimal scale that minimizes *weight error* $\|\boldsymbol{w} - \beta\, q(\boldsymbol{w})\|_2^2$, and with $\boldsymbol{q} = q(\boldsymbol{w})$, we obtain

$$\beta_* = \arg\min_{\beta} \|\boldsymbol{w} - \beta\, \boldsymbol{q}\|_2^2 = \frac{\boldsymbol{q}^\top \boldsymbol{w}}{\boldsymbol{q}^\top \boldsymbol{q}}\,, \tag{22}$$

i.e., a 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: $\beta_i^* = \frac{q(\boldsymbol{w}_i)^\top \boldsymbol{w}_i}{q(\boldsymbol{w}_i)^\top q(\boldsymbol{w}_i)}$. Similarly, the optimal scale that minimizes *output activations* can also be found in closed-form. We consider the matrix-vector product $Wx = \sum_{i=1}^{G} \boldsymbol{w}_i x_i$ with G input channels or blocks. We approximate this as $\sum_{i=1}^{G} \beta_i\, q(\boldsymbol{w}_i)\, x_i$, and define $\boldsymbol{A} = [\, q(\boldsymbol{w}_1)x_1, \ldots, q(\boldsymbol{w}_G)x_G \,] \in \mathbb{R}^{B\times G}$ and $\boldsymbol{b} = \boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^B$. The optimal group-wise scale reduces to the least-squares solution:

$$\begin{aligned} \boldsymbol{\beta}_* &= \arg\min_{\boldsymbol{\beta}} \|\boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{\beta}\|_2^2 \\ &= (\boldsymbol{A}^\top \boldsymbol{A})^{-1} \boldsymbol{A}^\top \boldsymbol{W}\boldsymbol{x}\,. \end{aligned} \tag{23}$$

E.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 For a linear layer $\mathbf{y} = \mathbf{W}\mathbf{x}$ with quantized weights $q(\mathbf{W})$ and error $\Delta\mathbf{W} = q(\mathbf{W}) - \mathbf{W}$, it is common (Nagel et al., 2020) to target expected output MSE of each layer as (local) proxy objective:

$$\mathcal{L}_{\text{local}} = \mathbb{E}\big[\|\Delta\mathbf{W}\,\mathbf{x}\|^2\big] \;= \mathbb{E}\big[\mathbf{x}^\top \Delta\mathbf{W}^\top \Delta\mathbf{W}\,\mathbf{x}\big] \tag{24}$$

$$= \mathrm{Tr}\big(\Delta\mathbf{W}\,\mathbf{H}_{\text{in}}\,\Delta\mathbf{W}^\top\big), \quad \mathbf{H}_{\text{in}} = \mathbb{E}[\mathbf{x}\mathbf{x}^\top]. \tag{25}$$

The equalities are exact under the true second moment; in practice we replace $\boldsymbol{H}_{\text{in}}$ by the empirical estimator $\tilde{\boldsymbol{H}}_{\text{in}} = \frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}$ from a finite calibration set $\boldsymbol{X} \in \mathbb{R}^{N\times D_{\text{in}}}$, so the trace expression becomes a Monte Carlo (stochastic) approximation of the population objective, which converges to the exact one as $N \to \infty$ under standard i.i.d. assumptions (law of large numbers). Rows of $\mathbf{W}$ are independent because the quadratic form decomposes as

$$\mathcal{L}_{\text{local}} = \sum_r \Delta\mathbf{w}_r^\top \mathbf{H}_{\text{in}} \Delta\mathbf{w}_r,$$

where $\Delta\mathbf{w}_r$ is the r-th row. For a single row, partition indices into *changed* C and *remaining* R:

$$\mathbf{H}_{\text{in}} = \begin{bmatrix} \mathbf{H}_{CC} & \mathbf{H}_{CR} \\ \mathbf{H}_{RC} & \mathbf{H}_{RR} \end{bmatrix}, \quad \Delta\mathbf{w} = [\Delta\mathbf{w}_C;\, \Delta\mathbf{w}_R].$$

Minimizing the quadratic w.r.t. $\Delta\mathbf{w}_R$ gives the analytic correction:

$$\boxed{\Delta\mathbf{w}_R^\star = -\,\mathbf{L}_{RR}^{-1}\,\mathbf{L}_{RC}\,\Delta\mathbf{w}_C.}$$

where $\boldsymbol{L}_{RR}$ and $\boldsymbol{L}_{RC}$ are lower-triangular matrices by indexing Cholesky of $\boldsymbol{H}$. Hence, the corrections require a single Cholesky decomposition, and a single lower-triangular solve, which has fast implementations. The correction algorithm is similar to LDLQ (Tseng et al., 2024a), generalizes the common GPTQ/OPTQ (Frantar et al., 2022) from scalar to vector quantization, and corrects updates in (Van Baalen et al., 2024) to account for column-mixing of VQ and typos. From a

probabilistic perspective, it can be noted that the correction is equivalent to Gaussian conditioning where the conditional mean of remaining weights R given the changed weights C, and Gaussian elimination. It is well-known that performing this iteratively can efficiently be done in Cholesky: the lower-triangular factor $\mathbf{L}$ from $\mathbf{H}_{\text{in}} = \mathbf{L}\mathbf{L}^\top$ as the inverse Cholesky only has to be computed ones and inverse Cholesky of remaining weights form a submatrix of the original inverse matrix. Batched triangular solves apply corrections for multiple rows efficiently.

Local vs global. Local updates can also be seen as targeting the final loss under an extremely crude Hessian approximation $\mathbf{H} = \mathbf{I} \otimes \mathbf{H}_{\text{in}}$, which ignores cross-row coupling and relating to the output curvature to target the ultimate final loss. Approximations to the global Hessian, as explored in (van der Ouderaa et al., 2023; Tseng et al., 2025) can provide stronger compression performance but require backward passes and are therefore more expensive. Since our focus in this work is on the representation itself, we evaluate all methods under the same local correction scheme to disentangle improvements due to the representation from those due to more powerful correction procedures. Global-Hessian-based corrections are orthogonal to our contribution, and for fair comparison, we restrict attention to the local GPTQ-like (generalized to vectors) updates described above.

E.3. Dimensionality Handling

Although Λ_{24} is 24-dimensional, we quantize vectors $x \in \mathbb{R}^D$ by splitting them into consecutive blocks of length 24:

$$x = (x^{(1)}, \ldots, x^{(B)}), \qquad x^{(j)} \in \mathbb{R}^{24}, \quad B = \lceil D/24 \rceil.$$

If D is not a multiple of 24, the final block is zero-padded. Each block is quantized independently using the Leech-lattice codebook $\Lambda_{24}(m)$, and the full codeword is the Cartesian product of the per-block indices. Thus, the overall scheme is naturally interpreted as a *product code* over 24-dimensional components in the classical sense of product vector quantization (**?**).

F. 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).

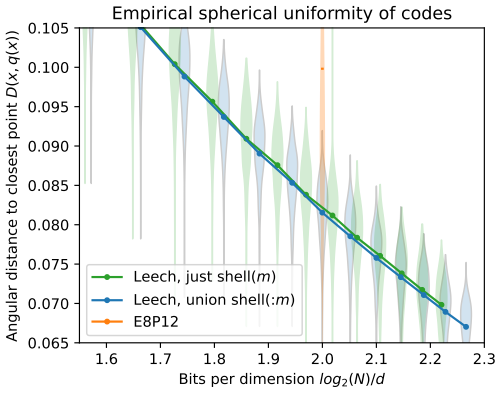

Figure 6. Empirical angular separation vs. rate. For each code, we show the distribution over nearest–neighbor angle against bits per dimension $\log_2(N)/d$. The two Leech-based constructions are traced by varying m (single shell m vs. union $1{:}m$). We compare to E8P12 and find that it lies above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

G. Spherical shaping or shape–gain: which achieves the lower distortion?

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 Λ_{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 Λ_{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_{gain} (e.g., matched to a χ_{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. In this sense, spherical shaping is geometric and monolithic, whereas shape–gain is modular and more flexible.

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_{gain} and the remainder to C_{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_{gain} is a χ_{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 χ-gain bits	2.00000 + 0.00000	2.0	0.085	1.782	89.12
Leech (shape-gain)	norm($\Lambda_{24}(12)$) + 1 χ-gain bits	1.95833 + 0.04167	2.0	0.078	1.843	**92.14**
Leech (shape-gain)	norm($\Lambda_{24}(11)$) + 2 χ-gain bits	1.91667 + 0.08333	2.0	0.080	1.825	91.24
Leech (shape-gain)	norm($\Lambda_{24}(10)$) + 4 χ-gain bits	1.83333 + 0.16667	2.0	0.085	1.780	89.01

Table 6. 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 6 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.** Shape–gain can improve performance over spherical shaping.
- **Key finding.** 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.** 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.