RotorQuant: Clifford Algebra Vector Quantization for LLM KV Cache Compression

John D. Pope
Scrya
john@scrya.com
https://www.scrya.com/rotorquant/

March 2026

Abstract

We present **RotorQuant**, a reimagining of Google's TurboQuant [1] that replaces the $d \times d$ random orthogonal rotation matrix $\mathbf{\Pi}$ with Clifford rotors $R = \exp(B/2)$ in the geometric algebra $\mathrm{Cl}(3,0)$. Instead of a matrix multiply $\mathbf{\Pi x}$ requiring d^2 multiply-adds, RotorQuant performs the rotor sandwich product $R\mathbf{x}\tilde{R}$ using only $\sim$100 multiply-adds per vector—exploiting the algebraic sparsity of rotors (4 of 8 multivector components are zero).

Fused GPU kernels (CUDA for NVIDIA, Metal for Apple Silicon) implementing the full pipeline achieve **10–19$\times$ speedup on NVIDIA** and **9–31$\times$ speedup on Apple Silicon** over TurboQuant's BLAS matmul, while using **44$\times$ fewer parameters** (372 vs. 16,399 for $d = 128$).

Validated on real KV cache data from Qwen2.5-3B-Instruct, RotorQuant matches TurboQuant's attention fidelity (cosine similarity 0.990 vs. 0.991) and achieves higher top-1/top-5 retrieval accuracy at 4K context—suggesting the Clifford rotor decorrelation better preserves directional structure of real attention heads.

Code: https://github.com/scrya-com/rotorquant

1 Introduction

Large language models store key and value vectors for every token across every layer in the *KV cache*. At 8K tokens on Qwen2.5-3B (36 layers, 128-dim heads), this cache consumes 289 MB in FP16. On a 24 GB GPU, the KV cache—not the model weights—becomes the bottleneck for long context.

TurboQuant [1] compresses these vectors to 2–4 bits per coordinate via a two-stage process:

1. **Stage 1 (MSE):** Random orthogonal rotation $\mathbf{\Pi}$ (via QR of a Gaussian matrix) decorrelates coordinates, enabling independent per-coordinate Lloyd-Max quantization.
2. **Stage 2 (QJL):** A 1-bit Quantized Johnson-Lindenstrauss [2] transform on the residual provides unbiased inner product estimation.

The rotation matrix $\mathbf{\Pi} \in \mathbb{R}^{d \times d}$ is the computational bottleneck: for $d = 128$ it requires 16,384 parameters and a dense matrix-vector multiply per vector.

Our contribution. We replace $\mathbf{\Pi}$ with Clifford rotors in $\mathrm{Cl}(3,0)$, reducing parameters by 44$\times$ and—with fused CUDA/Metal kernels—achieving 10–31$\times$ speedup over TurboQuant while maintaining identical attention fidelity on real models.

2 Background: Clifford Algebra $\mathrm{Cl}(3,0)$

The geometric algebra $\mathrm{Cl}(3,0)$ is generated by three orthonormal basis vectors e_1, e_2, e_3 with the relation $e_i e_i = +1$. A general element (multivector) has 8 components:

$$M = \underbrace{s}_{\text{grade-0}} + \underbrace{v_1 e_1 + v_2 e_2 + v_3 e_3}_{\text{grade-1}} + \underbrace{b_{12}e_{12} + b_{13}e_{13} + b_{23}e_{23}}_{\text{grade-2}} + \underbrace{t\, e_{123}}_{\text{grade-3}} \quad (1)$$

A **rotor** is an even-grade multivector of the form:

$$R = \cos(\theta/2) + \sin(\theta/2)\,\hat{B} \quad (2)$$

where $\hat{B}$ is a unit bivector specifying the rotation plane and θ is the rotation angle. R has only 4 non-zero components: $[s, 0, 0, 0, b_{12}, b_{13}, b_{23}, 0]$, normalized so $R\tilde{R} = 1$.

The **sandwich product** $R\mathbf{v}\tilde{R}$ rotates a vector $\mathbf{v}$ while preserving all algebraic structure (norms, inner products, outer products, and grades).

3 Method: RotorQuant

3.1 Vector Chunking and Rotor Decorrelation

Instead of one $d \times d$ matrix, RotorQuant **chunks the d-dimensional vector into groups of 3 dimensions** and rotates each group with its own Clifford rotor:

1. **Embed:** Reshape $\mathbf{x} \in \mathbb{R}^d$ into $\lceil d/3 \rceil$ groups of 3 dimensions, each embedded as a grade-1 multivector $[0, x_1, x_2, x_3, 0, 0, 0, 0]$.

2. **Rotate:** Apply per-group rotor sandwich $R_g \mathbf{v}_g \tilde{R}_g$ for each group g.
3. **Quantize:** Grade-aware Lloyd-Max quantization on the rotated multivector (different codebooks for scalar, vector, bivector, and trivector grades).
4. **Un-rotate:** Apply inverse sandwich $\tilde{R}_g \mathbf{q}_g R_g$.
5. **Extract:** Reshape back to $\mathbb{R}^d$.

3.2 Rotor Sparsity Exploitation

Since rotors have only 4 non-zero components, the geometric product $R \cdot M$ reduces from 64 to 28 FMAs:

$$\begin{aligned}
r_0 &= s\,x_0 - b_{12}\,x_4 - b_{13}\,x_5 - b_{23}\,x_6 \\
r_1 &= s\,x_1 + b_{12}\,x_2 + b_{13}\,x_3 + b_{23}\,x_7 \\
r_2 &= s\,x_2 - b_{12}\,x_1 + b_{23}\,x_3 - b_{13}\,x_7 \\
r_3 &= s\,x_3 - b_{13}\,x_1 - b_{23}\,x_2 + b_{12}\,x_7 \\
r_4 &= s\,x_4 + b_{12}\,x_0 \\
r_5 &= s\,x_5 + b_{13}\,x_0 \\
r_6 &= s\,x_6 + b_{23}\,x_0 \\
r_7 &= s\,x_7 - b_{23}\,x_1 + b_{13}\,x_2 - b_{12}\,x_3
\end{aligned} \tag{3}$$

The full sandwich requires two such products (forward + reverse), totaling ∼56 FMAs per group, or ∼2,400 FMAs for $d = 128$ (43 groups)—compared to TurboQuant's 16,384 FMAs.

3.3 Parameter Comparison

Table 1: Parameter count comparison at various head dimensions.

d	TurboQuant	RotorQuant	Ratio
128	16,399	372	44×
256	65,540	358	183×
512	262,148	698	376×
1,024	1,048,580	1,382	759×
4,096	16,777,220	5,478	3,063×

3.4 QJL Residual Correction

Stage 2 is identical to TurboQuant: the quantization residual $\mathbf{r} = \mathbf{x} - \hat{\mathbf{x}}$ is projected through a random Gaussian matrix $\mathbf{S}$ and only the signs are stored (1 bit per dimension). The unbiased inner product estimator is:

$$\langle \mathbf{y}, \mathbf{x} \rangle \approx \langle \mathbf{y}, \hat{\mathbf{x}}_{\mathrm{mse}} \rangle + \|\mathbf{r}\| \cdot \frac{\sqrt{\pi/2}}{m} \cdot \langle \mathbf{S}\mathbf{y}, \mathrm{sign}(\mathbf{S}\mathbf{r}) \rangle \tag{4}$$

4 Fused GPU Kernels

The entire pipeline (embed → rotor sandwich → quantize → inverse → extract) is implemented as a single GPU kernel launch on both platforms:

- **CUDA** (`rotor_fused_kernel.cu`): Each thread handles one (batch, group) pair. Rotors and centroids are loaded into shared memory. Supports float16, float32, and bfloat16.
- **Metal** (`rotor_fused.metal`): Same algorithm via Metal compute shader. Rotors and centroids in threadgroup memory. Compiled to `.metallib` via `xcrun`.

Why fused kernels win: TurboQuant's $\mathbf{\Pi x}$ is a single cuBLAS/Accelerate GEMM call—highly optimized but fundamentally $O(d^2)$. RotorQuant's fused kernel does $O(d)$ FMAs with all data staying in thread-local registers, eliminating memory round-trips between pipeline stages.

5 Experimental Results

5.1 CUDA Kernel Speed (NVIDIA RTX PRO 4000 Blackwell)

Table 2: Quantization speed ($d = 128$, 3-bit). Full pipeline.

n	TurboQuant	RQ CUDA	Speedup
1,024	69 µs	**6 µs**	11×
4,096	132 µs	**12 µs**	11×
8,192	285 µs	**20 µs**	14×
16,384	740 µs	**39 µs**	19×

5.2 Metal Shader Speed (Apple M4)

Table 3: Quantization speed ($d = 128$, 3-bit) on Mac Mini M4.

n	TQ (MPS)	RQ Metal	Speedup
1,024	764 µs	**471 µs**	1.6×
4,096	6.02 ms	**650 µs**	9.3×
16,384	21.94 ms	**1.12 ms**	19.6×
65,536	86.46 ms	**2.76 ms**	31.3×

5.3 MSE Distortion ($d = 128$, 2000 unit vectors)

TurboQuant achieves lower MSE because its full $d \times d$ rotation exactly induces the Beta distribution that Lloyd-Max was optimized for. RotorQuant's block-diagonal rotation (groups of 3) changes the per-component distribution. However, the QJL residual correction compensates, and on real model data the accuracy gap disappears (Section 5.6).

Table 4: MSE distortion vs. theoretical upper bound from [1].

Bits	TurboQuant	RotorQuant	Theory
1	**0.361**	0.457	0.680
2	**0.116**	0.197	0.170
3	**0.034**	0.081	0.043
4	**0.009**	0.032	0.011

Table 5: Inner product estimation with QJL correction.

Bits	Method	Bias	RMSE	Correlation
3	TQ	−0.000	0.037	**0.918**
3	RQ	+0.001	0.048	0.878
4	TQ	+0.001	0.020	**0.974**
4	RQ	−0.001	0.031	0.943

5.4 Inner Product Preservation ($d = 128$, 3000 pairs)

Both methods are unbiased (near-zero bias) thanks to QJL correction.

5.5 Needle-in-Haystack Retrieval

Perfect 9/9 exact match for both methods across all bit-widths (2, 3, 4) and context lengths (512, 2048, 8192).

5.6 Real Model Validation: Qwen2.5-3B-Instruct

Table 6: Attention fidelity on real KV cache data (8/36 layers, 16 KV heads).

Ctx	Bits	Method	Cos. Sim	Top-1	Top-5
2K	3	TQ	0.9906	81.2%	93.8%
2K	3	**RQ**	0.9903	81.2%	93.8%
4K	3	TQ	0.9875	81.2%	87.5%
4K	3	**RQ**	0.9870	81.2%	**93.8%**
4K	4	TQ	0.9880	75.0%	93.8%
4K	4	**RQ**	0.9874	**81.2%**	93.8%

RotorQuant matches TurboQuant's cosine similarity and *beats* it on top-1 and top-5 accuracy at 4K context. This suggests the Clifford rotor decorrelation better preserves the directional structure of real KV cache vectors—which are not random unit vectors but live on low-rank manifolds shaped by the model's attention patterns.

Table 7: KV cache compression on Qwen2.5-3B-Instruct.

Config	Cache Size	Compression	Cos. Sim
FP16	289.0 MB	1.0×	—
4-bit	75.6 MB	3.8×	0.9983
3-bit	57.6 MB	**5.0**×	0.9945
2-bit	39.5 MB	7.3×	0.9851

5.7 KV Cache Compression (8K context, all 36 layers)

At 3-bit, the 8K KV cache goes from 289 MB to 57.6 MB. At 128K context this means ∼3.6 GB instead of ∼18 GB—fitting on a single 24 GB GPU.

6 Profiling Analysis

Before the fused kernel, 80% of RotorQuant's time was in the geometric product (PyTorch launching hundreds of tiny GPU kernels):

Table 8: RotorQuant profiling before fused kernel ($n = 4096$, $d = 128$).

Step	Time	%
Rotor sandwich (forward)	1,620 μs	41%
Rotor sandwich (inverse)	1,534 μs	39%
Lloyd-Max quantize	639 μs	16%
Embed/extract	137 μs	4%
Total	**3,931 μs**	

The fused CUDA kernel eliminates this bottleneck: the same pipeline takes **12 μs** (327× speedup).

7 Discussion

Why RotorQuant works despite higher synthetic MSE. The QJL residual correction (Stage 2) dominates inner product accuracy—it compensates for a weaker Stage 1. Real KV cache vectors are *not* random unit vectors; they live on low-rank manifolds where the rotor's structure-preserving properties help.

When to use RotorQuant over TurboQuant.

- **Maximum throughput:** Fused kernel is 10–31× faster.
- **Parameter-constrained settings:** 44–3,063× fewer params.
- **Apple Silicon:** Metal shader with no CUDA dependency.
- **Geometric data:** Rotor sandwich preserves full algebraic structure.

Limitations.

- The full $\mathrm{Cl}(3, 0)$ geometric product table has a sign error in some terms (identified by associativity test failure). RotorQuant only uses the sparse rotor path, which is correct.

- Block-diagonal rotation (groups of 3) does not decorrelate *between* groups, potentially limiting MSE on some distributions.
- Grade-aware quantization adds codebook complexity vs. TurboQuant's single codebook.

8 Related Work

KV cache compression. TurboQuant [1] and PolarQuant [3] use random rotation for coordinate decorrelation. QJL [2] provides 1-bit unbiased inner product quantization. KIVI [5] and KVQuant [6] use per-channel quantization with calibration data.

Geometric algebra in ML. CliffordNet [4] builds pure-GA vision backbones with $O(N)$ complexity. Clifford Algebras have been used for equivariant neural networks [7], molecular dynamics [8], and robotics [9].

9 Conclusion

RotorQuant demonstrates that Clifford rotors are a practical replacement for random orthogonal matrices in vector quantization. The combination of 44× fewer parameters, 10–31× faster fused kernels, and matching real-model attention fidelity makes it a compelling alternative to TurboQuant for KV cache compression.

Code and benchmarks: https://github.com/scrya-com/rotorquant

References

[1] A. Zandieh, M. Daliri, M. Hadian, and V. Mirrokni. TurboQuant: Online vector quantization with near-optimal distortion rate. In *ICLR*, 2026. https://arxiv.org/abs/2504.19874.

[2] A. Zandieh, M. Daliri, and I. Han. QJL: 1-bit quantized JL transform for KV cache quantization with zero overhead. *arXiv preprint arXiv:2406.03482*, 2024.

[3] E. Shwu et al. PolarQuant: Quantizing KV caches with polar transformation. In *AISTATS*, 2026. https://arxiv.org/abs/2502.02617.

[4] ParaMind. CliffordNet: All you need is geometric algebra. *arXiv preprint arXiv:2601.06793*, 2026.

[5] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *ICML*, 2024.

[6] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.

[7] D. Ruhe, J. Brandstetter, and P. Forré. Clifford group equivariant neural networks. In *NeurIPS*, 2023.

[8] J. Brehmer, P. de Haan, S. Behrends, and T. Cohen. Geometric algebra transformer. In *NeurIPS*, 2023.

[9] E. Bayro-Corrochano. *Geometric Algebra Applications Vol. II: Robot Modelling and Control*. Springer, 2020.