Fast Specialized Rotations in Clifford Algebra

Travis Pence[1] **Daisuke Yamada**[1] **Jiaqi Mo**[1] **Chanyoung Moon**[1,2]

Karthikeyan Sankaralingam[1,3] **Vikas Singh**[1]

Abstract

Clifford algebra is becoming increasingly common in machine learning, e.g., in neural operators, equivariant transformers, and structured linear layers. The main appeal is that scalars, vectors, oriented planes, volumes, and higher-grade quantities live together in a single algebraic space. In this setting, rotations are represented by *rotors*, which are parametrized by bivectors, with only $\binom{n}{2}$ degrees of freedom. Most Clifford algebra-based models work only with small algebras, in part because of a bottleneck in the sandwich product, $x \mapsto rxr^\dagger$, a basic operation for applying rotations to multivectors. It is evaluated through geometric products or via a full action matrix, both of which scale poorly with the 2^n-dimensional Clifford basis. We ask whether the $\binom{n}{2}$-parameter description can be carried all the way through application, and show that the answer is yes. The sandwich action decomposes grade-wise into exterior powers of the induced vector rotation. We factor this vector rotation into Givens rotations and apply their lifted actions as sparse updates on pairs of blades. The resulting algorithm never needs to form the rotor coefficients or the $2^n \times 2^n$ action matrix. Empirically, our algorithm achieves $67\times$ wall-clock and $229\times$ memory improvements over existing rotor-sandwich implementation, enabling rotor-based layers at scales previously out of reach.

1 Introduction

Clifford algebra [Lounesto, 2001b, Dorst et al., 2009] is a structured generalization of linear algebra, whose elements, called multivectors [Hestenes and Sobczyk, 2012], unify scalars, vectors, oriented planes, and higher-grade objects into a single graded space. A single multiplication operation, called the *geometric product*, composes these algebraic elements. This product is a central operation, as it subsumes the inner and outer products, generates the action of rotation and reflection groups, captures correlations between fields of different type, and admits factorizations of transformations into low-dimensional algebraic primitives.

Each of the properties identified above has driven a distinct line of work in machine learning recently: from modeling correlated physical fields in PDE solvers [Brandstetter et al., 2023, Alberto Pepe, 2025] to building equivariant transformer architectures across choices of metric [Brehmer et al., 2023, de Haan et al., 2024, Spinner et al., 2024] to constructing steerable convolutional networks [Zhdanov et al., 2024] and parametrizing linear layers from a small number of algebraic primitives [Pence et al., 2025]. Yet nearly all existing results work in algebras where the dimension n is small (e.g., $n = 3$ gives $2^n = 8$ basis elements). Here, the bottleneck is entirely computational: the geometric product, which defines the algebra and underlies nearly every Clifford-based model, costs $O(4^n)$ scalar operations, since it pairs each of the 2^n basis components of a with each of the 2^n basis components of b to produce ab. Thus, larger algebras are avoided in practice, despite their utility and expressiveness, because the compute footprint is often unaffordable.

[1]University of Wisconsin-Madison, Madison, Wisconsin, USA. [2]Korea University, Seoul, Republic of Korea. [3]NVIDIA, Santa Clara, California, USA. Correspondence to: Travis Pence <tnpence@wisc.edu>. We provide the code at `https://github.com/vsingh-group/CliffordCUDA`.

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

The rotor sandwich product. A key application of the geometric product is the *rotor sandwich* $x \mapsto rxr^\dagger$ which generalizes rotations of vectors to arbitrary multivectors. The rotor r is an element of the algebra constructed from a bivector, and the sandwich is its action on a multivector x. It is the key operation in geometric encoder layers [Ruhe et al., 2023b], the mechanism underlying learned rotor embeddings [Sriram et al., 2025], and the building block of the linear-layer parametrization in Pence et al. [2025]. However, each of these papers acknowledges the sandwich's cost as a primary practical limitation. The sandwich can be applied in two ways, each with a drawback. (1) The direct route evaluates two geometric products and thus incurs the full $O(4^n)$ cost on every call. (2) The matrix route precomputes the $2^n \times 2^n$ matrix representation of the rotor; the matrix applies cheaply, but building it from the bivector takes $n-1$ matrix multiplications, and storing it forfeits the $\binom{n}{2}$-parameter bivector that kept the algebra parameter-light in the first place. Hence, there is a natural tradeoff: one is expensive at runtime, the other in memory.

Our contribution. We give an algorithm for the rotor sandwich that is *cheaper in both runtime and memory*. We work directly from the $\binom{n}{2}$ bivector parameters, applying the action of the rotor on a multivector without ever constructing the rotor or its matrix representation. We present the algorithm, analyze its cost, and benchmark it against three implementations from recent Clifford-based work where our algorithm improves wall-clock time and peak GPU memory by up to $67\times$ and $229\times$ respectively. We show how a naive implementation suffers a slowdown in practice and give a constructive proof to avoid the issue. We integrate the algorithm into one of these works, scaling to settings that prior implementations could not reach.

2 Preliminaries of Clifford Algebra

We briefly review some concepts that will be used throughout the paper. A Clifford algebra is built from a vector space by equipping it with a product to track both metric information and oriented subspaces. Let us fix orthonormal generators $e_1, \ldots, e_n$ of $\mathbb{R}^n$ and impose the relation

$$e_i^2 = 1, \qquad e_i e_j = -e_j e_i \quad \text{for } i \neq j. \tag{1}$$

The non-commutative algebra generated by these expressions is called the Euclidean Clifford algebra $\mathrm{Cl}(n)$ (see Lounesto [2001a] and Snygg [2011]). Its multiplication is the *geometric product*. The second expression says that distinct basis directions anticommute, so the order of basis vectors matters. For vectors $u, v \in \mathbb{R}^n$, the geometric product decomposes into symmetric and antisymmetric parts,

$$uv = u \cdot v + u \wedge v, \tag{2}$$

where $u \cdot v$ is the standard inner product and $u \wedge v$ is called the *exterior* (or *wedge*) product, which is the oriented plane spanned by u and v. This way, the geometric product simultaneously captures alignment and orientation via $u \cdot v$ and $u \wedge v$, respectively. For $i \neq j$, we have $e_i \cdot e_j = 0$ and

$$e_i e_j = e_i \wedge e_j = -e_j \wedge e_i. \tag{3}$$

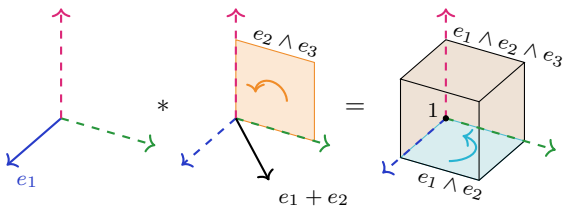

Figure 1: An example of the geometric product in $\mathrm{Cl}(3)$.

Let $I = \{i_1 < \cdots < i_k\} \subseteq \{1, \ldots, n\}$ be an ordered index set. We write

$$e_I \triangleq e_{i_1} \wedge e_{i_2} \wedge \cdots \wedge e_{i_k} = e_{i_1} e_{i_2} \cdots e_{i_k}. \tag{4}$$

Here, e_I is called a *basis blade*: the oriented k-dimensional coordinate subspace spanned by the vectors indexed by I. For example, e_{12} represents the oriented (e_1, e_2)-plane, while e_{123} is the oriented volume. The blades e_I form a basis of $\mathrm{Cl}(n)$, indexed by all subsets $I \subseteq \{1, \ldots, n\}$, giving $\mathrm{Cl}(n)$ dimension 2^n as a real vector space. For instance, $\mathrm{Cl}(3) = \mathrm{span}\{1,\ e_1, e_2, e_3,\ e_{12}, e_{13}, e_{23},\ e_{123}\}$. The geometric product between basis blades is determined by the anticommutation rules. Repeated indices cancel because $e_i^2 = 1$, while swapping distinct generators introduces signs. For example, $e_{12}e_{23} = e_1e_2e_2e_3 = e_1e_3 = e_{13}$, and $e_{12}e_{13} = e_1e_2e_1e_3 = -e_2e_3 = -e_{23}$. The number of generators in e_I is its *grade*, denoted $|I|$. Grade-0 elements are scalars, grade-1 elements are vectors, grade-2 elements are *bivectors*, grade-3 elements are *trivectors*. There are $\binom{n}{k}$ basis blades of grade

k, so the algebra decomposes as the graded vector space

$$\mathrm{Cl}(n) = \bigoplus_{k=0}^{n} \mathrm{Cl}^k(n), \tag{5}$$

where $\mathrm{Cl}^k(n)$ is the span of all grade-k basis blades. A general element of $\mathrm{Cl}(n)$ is called a *multivector*. It can contain components of several grades: $x = \sum_{I \subseteq \{1,\dots,n\}} x_I e_I,\ x_I \in \mathbb{R}$. For example, e_{23} is a basis bivector; $e_{12} + 2e_{23}$ is a bivector; $4 + 3e_1 + (e_{24} - 2e_{12})$ is a multivector with scalar, vector, and bivector components. Note that in practice, a multivector is stored as a flat array of its 2^n real coefficients x_I, one per basis blade. For a multivector x, the *grade-k projection* $\langle x \rangle_k = \sum_{|I|=k} x_I e_I$ extracts its grade-k component. We will also use the *even subalgebra* $\mathrm{Cl}^+(n) = \bigoplus_{k \text{ even}} \mathrm{Cl}^k(n)$, which consists of scalars, bivectors, grade-4 elements, and so on. Rotors, introduced shortly, live in this even subalgebra. The *reversion* operation, denoted †, reverses the order of generators in each blade: $(e_{i_1} e_{i_2} \cdots e_{i_k})^\dagger = e_{i_k} \cdots e_{i_2} e_{i_1}$. For example, $e_i^\dagger = e_i$, while $e_{ij}^\dagger = (e_i e_j)^\dagger = e_j e_i = -e_i e_j = -e_{ij}$ for $i \neq j$. For the orthogonal transformations considered later, reversion plays the role of the transpose: if r is a rotor, then $r^\dagger$ is its inverse.

Exterior powers. The second object we need is the exterior power of a linear map. Let $S\colon \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. Its action on vectors induces a canonical action on oriented k-dimensional subspaces. This induced map is the *k-th exterior power* of S, denoted $\bigwedge^k S$, and is defined on basis blades by

$$\bigwedge^k S\,(e_{i_1} \wedge \cdots \wedge e_{i_k}) = (Se_{i_1}) \wedge \cdots \wedge (Se_{i_k}), \tag{6}$$

then extended linearly to all of $\mathrm{Cl}^k(n)$. Intuitively, S moves vectors while $\bigwedge^k S$ moves oriented k-dimensional blades. Although S is an $n \times n$ matrix, its induced action on grade-k elements is represented by a $\binom{n}{k} \times \binom{n}{k}$ matrix. More details are in Hestenes and Sobczyk [2012].

3 Cost of the Rotor Sandwich

In standard linear algebra, a rotation matrix acts only on vectors, but we need a mechanism to do the same for planes, volumes, and higher-grade components. Clifford algebra provides this via the *rotor*. We describe the rotor and show that the standard ways of applying it incur either expensive compute or excessive memory, undermining the properties that make rotors attractive in the first place.

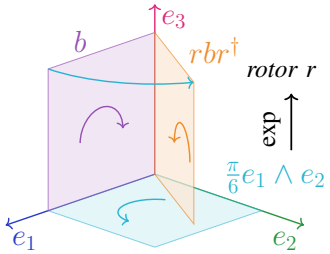

Figure 2: A rotor, parametrized by a bivector, acting on another bivector in $\mathrm{Cl}(3)$ through the sandwich product.

Rotors. A *rotor* extends the notion of a rotation from vectors to arbitrary multivectors. It acts on a multivector $x \in \mathrm{Cl}(n)$ through the *sandwich product*

$$x \mapsto rxr^\dagger. \tag{7}$$

On a vector, the sandwich agrees with the standard rotation; on higher-grade elements, it gives the induced action on planes, volumes, and so on. Hence, a single algebraic operation rotates all grades coherently. Rotors form the *Spin group* [Lounesto, 2001b],

$$\mathrm{Spin}(n) \triangleq \left\{ r \in \mathrm{Cl}^+(n) \mid rr^\dagger = 1 \text{ and } rvr^\dagger \in \mathrm{Cl}^1(n) \text{ for all } v \in \mathrm{Cl}^1(n) \right\},$$

analogous to how $\mathrm{SO}(n)$ is the rotation group on vectors. Precisely, $\mathrm{Spin}(n)$ is a double cover of $\mathrm{SO}(n)$: $\mathrm{Spin}(n)/\{\pm 1\} \cong \mathrm{SO}(n)$. That is, for every $P = (p_{ab}) \in \mathrm{SO}(n)$ there exists $r \in \mathrm{Spin}(n)$, unique up to sign, such that $re_b r^\dagger = \sum_{a=1}^n p_{ab} e_a$ for all $b = 1, \dots, n$. Since r and $-r$ induce the same sandwich, the action $x \mapsto rxr^\dagger$ depends only on P.

Note that as elements of the even subalgebra, rotors have up to 2^{n-1} components. However, rotors admit a parametrization that is more compact still.

A rotor is determined by a bivector. Recall the relationship between rotors and ordinary rotations. The groups $\mathrm{Spin}(n)$ and $\mathrm{SO}(n)$ are distinct, related by a double cover, but they have the *same* Lie algebra. For $\mathrm{SO}(n)$, this Lie algebra is the space of skew-symmetric matrices

$$\mathfrak{so}(n) \triangleq \{B \in \mathbb{R}^{n \times n} \mid B = -B^T\}, \tag{8}$$

which has dimension $\binom{n}{2}$ and is a Lie algebra under the commutator $[U, V] \triangleq UV - VU$. The matrix exponential maps this linear space to rotations, so every rotation matrix can be written as $\exp(B) \in \mathrm{SO}(n)$ for some $B \in \mathfrak{so}(n)$. The Clifford analogue replaces skew-symmetric matrices by bivectors. The bivectors $\mathrm{Cl}^2(n)$ are likewise closed under the Clifford commutator $[a, b] \triangleq ab - ba$ (e.g., $[e_{12}, e_{23}] = 2e_{13}$), and the correspondence $b = \sum_{i<j} B_{ij}\, e_{ij} \mapsto 2B$ is a Lie algebra isomorphism $\mathrm{Cl}^2(n) \cong \mathfrak{so}(n)$. Hence, every rotor can be written as the Clifford exponential of a bivector,

$$r = \exp_{\mathrm{Cl}}(b) = \sum_{i=0}^{\infty} \frac{b^i}{i!}, \qquad b \in \mathrm{Cl}^2(n). \tag{9}$$

The rotor is therefore controlled by only $\binom{n}{2}$ bivector coefficients, matching the degrees of freedom of an ordinary rotation. But we should appreciate the difference: the resulting transformation acts not only on vectors but on *all* grades of $\mathrm{Cl}(n)$. This gives a rotation-like transformation with only $\binom{n}{2}$ parameters acting on the full multivector space. The problem is that the usual ways of applying the sandwich product do *not* preserve this parameter count.

Option A: expand the sandwich as geometric products. The most direct route applies (7) exactly as written: compute rx, then $(rx)r^\dagger$. No matrix representation is built, but the full bilinear structure of the geometric product must be evaluated twice. On basis blades, we have $e_I e_J = \pm e_{I \triangle J}$, where $\triangle$ is symmetric difference and the sign depends on the ordering of generators in I and J. These signs and output indices are stored in a $2^n \times 2^n$ lookup *Cayley table*. Hence, for arbitrary $a, b \in \mathrm{Cl}(n)$, computing ab requires summing contributions from all pairs of basis blades (e_I, e_J) and routing each contribution to the output blade $e_{I \triangle J}$.

The problem. This all-pairs expansion cannot be easily avoided in the blade basis, as the computation enumerates the algebraic interactions between the components of the two operands. The rotor sandwich inherits this exponential interaction pattern through both of its geometric products. As a result, counting scalar products gives 2^{2n} operations for each product, hence 2^{2n+1} for the full sandwich.

Option B: materialize the sandwich as a matrix. For a fixed rotor r, the map $x \mapsto rxr^\dagger$ is linear in x, so it can be represented as a matrix acting on the 2^n coefficients of a multivector. This avoids evaluating two geometric products in Option A, but the matrix encodes the *full* induced action on $\mathrm{Cl}(n)$: it is $2^n \times 2^n$, block diagonal by grade, with total nonzero count

$$\sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n} \sim \frac{2^{2n}}{\sqrt{\pi n}}. \tag{10}$$

The problem. Applying the sandwich is a sparse matrix-vector multiplication, but constructing and storing the matrix requires materializing an object *exponentially larger* than the bivector that defines the rotor. This is especially problematic when the trainable parameters are the bivector coefficients: the action matrix must be reconstructed from those parameters during training, which requires a sequence of $n - 1$ large matrix multiplications [Pence et al., 2025]. Thus, the matrix route incurs the memory and construction cost of the full induced action.

Both options construct too much. The two routes fail in different ways but for the same reason. Option A constructs the rotor as a multivector and expands the sandwich through all blade interactions. Option B materializes the entire linear action on $\mathrm{Cl}(n)$. In both cases, a transformation specified by only $\binom{n}{2}$ bivector parameters is applied only after passing through an exponentially larger object.

> *The object we need is neither the rotor's coefficients nor its action matrix, but the* ***result*** $rxr^\dagger$.

We next show how to compute this result directly from the bivector parameters, with no intermediate exponential object.

4 Applying the Rotor Action Without Constructing it

Our goal is to turn the above observation into an algorithm. Given a bivector $b \in \mathrm{Cl}^2(n)$ and a multivector $x \in \mathrm{Cl}(n)$, we want to compute the rotor sandwich $rxr^\dagger$ where $r = \exp_{\mathrm{Cl}}(b)$. To keep

the working set at the bivector parameter count, we cannot construct either the rotor's coefficients or its full action matrix. To achieve this, we step outside the Clifford product: a rotor induces an ordinary rotation on vectors, and its action on higher grades is obtained by lifting that rotation through exterior powers. We exploit this by factoring the vector rotation into planar rotations and applying their lifted actions directly, without ever forming the lifted matrices. As a preview, the sandwich will be evaluated as

$$\text{bivector} \;\to\; \text{vector rotation} \;\to\; \text{planar factors} \;\to\; \text{sparse blade-pair updates on } x.$$

Step 1. Sandwich is a direct sum of exterior powers. Let $B \in \mathfrak{so}(n)$ be the skew-symmetric matrix corresponding to the bivector b, and define

$$S = \exp(2B) \in \mathrm{SO}(n). \tag{11}$$

Pence et al. [2025] shows the rotor sandwich decomposes grade by grade as the block-diagonal matrix

$$x \mapsto rxr^\dagger \qquad \Longleftrightarrow \qquad x \mapsto \left(\bigoplus_{k=0}^{n} \bigwedge^k S \right) x. \tag{12}$$

Equivalently for all k, the grade-k component transforms as

$$\left\langle rxr^\dagger \right\rangle_k = \left(\bigwedge^k S \right) \langle x \rangle_k. \tag{13}$$

This identity replaces the Clifford-algebra multiplication by an ordinary vector-space rotation S lifted to each grade. The matrix on the right is the same $2^n \times 2^n$ object discussed in Section 3, now resolved into grade blocks $\bigwedge^k S$ of size $\binom{n}{k} \times \binom{n}{k}$. The question is whether the right-hand side of Equation 13 can be computed without materializing $\bigwedge^k S$. Our insight is that *only the dense matrix lives in n dimensions.* The matrix $S = \exp(2B)$ is the exponential of a skew-symmetric matrix, computed in the original n-dimensional vector space rather than the full 2^n-dimensional Clifford algebra. Gallier and Xu [2003] gives the closed form

$$S = I_n + \sum_{i=1}^{\lfloor n/2 \rfloor} \left(\sin 2\theta_i \, B_i + (1 - \cos 2\theta_i) B_i^2 \right), \tag{14}$$

where for each i,

$$B_i = u_i v_i^\top - v_i u_i^\top, \qquad B_i^2 = -(u_i u_i^\top + v_i v_i^\top) \tag{15}$$

is built from a pair of singular vectors $\{u_i, v_i\}$ of B associated with singular value θ_i, and B_i^2 is the negative orthogonal projector onto the corresponding invariant two-plane. This computation lives entirely in $\mathbb{R}^{n \times n}$ and is thus inexpensive compared to computations in the 2^n-dimensional algebra.

Step 2. Decompose S into planar rotations. We use a classical *Givens factorization* of S. Sparse planar-rotation products also appear in multiresolution matrix factorization and its incremental variants [Kondor et al., 2015, Ithapu et al., 2017], where the factors are learned. Here, the factorization is the standard one for any rotation in $\mathrm{SO}(n)$,

$$S = G_{(p_1,q_1)}(\theta_1) G_{(p_2,q_2)}(\theta_2) \cdots G_{(p_N,q_N)}(\theta_N) I_n, \qquad N = \binom{n}{2}. \tag{16}$$

$G_{(p,q)}(\theta)$ is the rotation by angle θ in the plane spanned by e_p and e_q, fixing every other basis vector and acting on coordinates p and q as

$$\begin{pmatrix} v'_p \\ v'_q \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} v_p \\ v_q \end{pmatrix}. \tag{17}$$

The factorization writes the rotation S as a sequence of $\binom{n}{2}$ planar rotations, each touching only two coordinates. We now ask what happens when we lift this factorization to grade k.

Step 3. Lift the factors, not the matrix. Exterior powers respect products. For any $A, B \in \mathrm{GL}(n)$,

$$\bigwedge^k (AB) = \left(\bigwedge^k A \right) \left(\bigwedge^k B \right). \tag{18}$$

Applying this to the Givens factorization (16) gives

$$\bigwedge^k S = \bigwedge^k G_1 \cdot \bigwedge^k G_2 \cdots \bigwedge^k G_N,$$

where $G_t = G_{(p_t,q_t)}(\theta_t)$. Therefore,

$$\left(\bigwedge^k S\right) \langle x \rangle_k = \bigwedge^k G_1 \left(\bigwedge^k G_2 \cdots \left(\bigwedge^k G_N \langle x \rangle_k\right) \cdots\right). \tag{19}$$

This avoids forming the induced map $\bigwedge^k S$, and it remains to apply each lifted factor $\bigwedge^k G_{(p,q)}(\theta)$ without constructing it. Theorem 1 shows this is possible: a planar rotation only couples specific blade pairs.

Theorem 1 (Application of $\bigwedge^k G_{(p,q)}$)**.** *Fix a plane (p, q) with $1 \le p < q \le n$ and a grade k. The action $\bigwedge^k G_{(p,q)}(\theta)\, \langle x \rangle_k$ on any $x \in \mathrm{Cl}(n)$ can be computed without constructing $\bigwedge^k G_{(p,q)}(\theta)$ by independent updates to $\binom{n-2}{k-1}$ disjoint partner pairs (I, J) with $J = (I \setminus \{p\}) \cup \{q\}$. The updates depend on θ only through $\cos\theta$ and $\sin\theta$, together with a fixed sign pattern $\sigma_{I\to J} \in \{\pm 1\}$ over partner pairs.*

Proof sketch. The action depends on each basis blade e_I only through $|I \cap \{p, q\}|$, giving three cases. The cases $|I \cap \{p, q\}| \in \{0, 2\}$ fix e_I, while $|I \cap \{p, q\}| = 1$ couples e_I to its partner e_J on a two-dimensional invariant subspace. The number of partner pairs at grade k is $\binom{n-2}{k-1}$. Full proof in Appendix A.

Step 4. Sparse blade-pair rotations. Composing the lifted Givens factors gives the full grade-k action in (13). By Thm. 1, each factor $\bigwedge^k G_{(p_t,q_t)}(\theta_t)$ decomposes into independent updates over blade-pairs. This construction gives Alg. 1, which applies the rotor action by sweeping through the $\binom{n}{2}$ Givens factors and updating only the partner pairs touched by each rotation plane. The only rotor-dependent quantities are the Givens angles; the partner indices and signs depend only on the algebraic layout – the dimension n, the grade k, and the rotation plane (p_t, q_t). They do not depend on the learned bivector parameters or the input data, and can be precomputed once and reused for different rotors of the same algebra, i.e. across batches and training iterations.

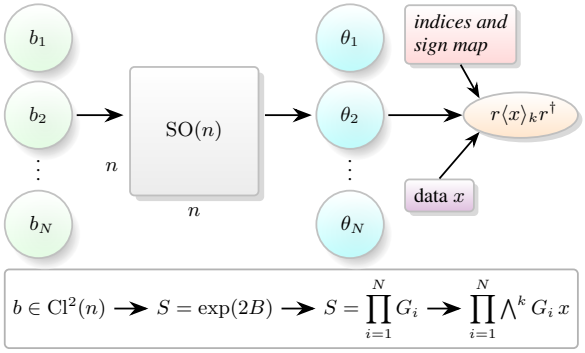

Figure 3: The [bivector → Special Orthogonal Matrix → Givens Decomposition → Wedge-lifted Givens Application] process to compute the rotor sandwich product.

The rotor sandwich can now be applied without forming the rotor or its action matrix. We compute the underlying vector rotation, factor it into Givens rotations, and apply the action of each grade-k lifted factor through sparse blade-pair updates.

Remarks. A Givens factorization of $S \in \mathrm{SO}(n)$ uses $\binom{n}{2}$ rotations to express the $n \times n$ matrix. Lifted through the exterior power, the $2^n \times 2^n$ rotor matrix is now being expressed through those same $\binom{n}{2}$ rotations. The factorization length scales as $\binom{n}{2} \approx n^2/2$, not the algebra dimension 2^n. Writing $D = 2^n$ for the size of a multivector, the total cost is $3n(n-1)2^{n-2} = O(D \log^2 D)$ FLOPs (Appendix A). Since any method must at least read the D input coefficients, the algorithm is optimal up to log factors.

5 From Theory to Implementation

Algorithm 1 accesses many pairs of blades from shared memory. A naive layout of indices creates GPU shared-memory bank (slot) conflicts, slowing down the algorithm. We sketch the problem

Algorithm 1 Apply $\bigwedge^k S$ to $\langle x \rangle_k$

Require: $y = \langle x \rangle_k \in \mathbb{R}^{\binom{n}{k}}$; $\theta_t \in \mathbb{R}$, $i_{k,t}, j_{k,t} \in \{1, \ldots, \binom{n}{k}\}^{\binom{n-2}{k-1}}$, $\sigma_{k,t} \in \{\pm 1\}^{\binom{n-2}{k-1}}$ for $t = 1, \ldots, \binom{n}{2}$

1: **for** $t = 1$ to $\binom{n}{2}$ **do**
2: $\quad a \leftarrow y[i_{k,t}]$
3: $\quad b \leftarrow y[j_{k,t}]$
4: $\quad y[i_{k,t}] \leftarrow \cos\theta_t \cdot a - \sin\theta_t \cdot (\sigma_{k,t} \odot b)$
5: $\quad y[j_{k,t}] \leftarrow \sin\theta_t \cdot (\sigma_{k,t} \odot a) + \cos\theta_t \cdot b$
6: **end for**
7: **return** y

and how to eliminate bank conflicts. For the full background, construction, and formalization, see Appendix B.

The problem. GPU shared memory is partitioned into 32 slots, reading and writing 32 values at full speed only when they sit on 32 distinct slots. The GPU processes 32 partner pairs at a time, so conflict-freeness requires the 32 left blades and the 32 right blades to each land in 32 distinct slots. The canonical layout puts many blades in the same slot, creating conflicts and slowing down the GPU. Which slot a blade lands in is determined by the layout, which we are free to choose.

Balancing slots. We identify each blade e_I with its binary representation $\mathrm{bit}(I) \in \{0,1\}^n$ and restrict the layout to the binary linear family $\pi[I] = M \cdot \mathrm{bit}(I)$, where M is an invertible $n \times n$ binary matrix. The slot holding blade I is given by $A \cdot \mathrm{bit}(I)$, where A is the $5 \times n$ submatrix of M producing the bottom 5 bits of $\pi[I]$. The layout is *balanced* when both the left and right blades of each iteration distribute uniformly across the 32 slots.

Theorem 2 (Slot-balance condition)**.** *The layout $\pi[I] = M \cdot \mathrm{bit}(I)$ is balanced if and only if*

$$\mathrm{rank}\big(A^{(p,q)}\big) = 5$$

for every (p, q) with $0 \le p < q < n$, where $A^{(p,q)}$ is A with columns p, q removed and M invertible.

Given a balanced layout, we build a bipartite multigraph for each rotation round whose vertices are the 32 slots on each side and whose edges are the partner pairs. Balance makes this graph Δ-regular, and König's edge-coloring theorem decomposes it into Δ perfect matchings. Together, this makes every group of 32 partner pairs in the iteration conflict-free. See Appendix B for full details.

6 Experiments

We empirically evaluate Algorithm 1 from two angles. First, we benchmark it against the rotor-sandwich implementations from recent Clifford-based machine learning work. Second, we apply our algorithm to a downstream LLM setting that prior implementations could not reach at scale, where the rotor sandwich was the limiting factor.

6.1 Algorithm Benchmark

FLOPs and memory. As discussed in Section 3, the geometric-product and matrix routes face opposite tradeoffs in FLOPs and memory. Algorithm 1 avoids both: it uses fewer FLOPs than the matrix route and has a smaller representation than the geometric-product route. Table 1 gives the FLOPs and representation sizes.

Benchmark setup. We compare our algorithm against three rotor-sandwich implementations from recent Clifford algebra work, evaluating training and inference speed and peak GPU memory. Throughout, `WLG` is our wedge-lifted Givens algorithm; `CFL` (Clifford Layers) is the geometric-product route from Ruhe et al. [2023b], which builds the rotor r and evaluates $rxr^\dagger$; `CLL` (Composing Linear Layers) is the route from Pence et al. [2025], which builds the rotor matrix and applies it via matrix multiplication; `CARE` (Clifford Algebraic Rotor Embeddings) is the hybrid approach of Sriram et al. [2025], which evaluates $rxr^\dagger$ with outer products and a scatter. We evaluate on $\mathrm{Cl}(n)$ for $n \in \{7, 8, 9, 10, 11, 12, 13\}$.

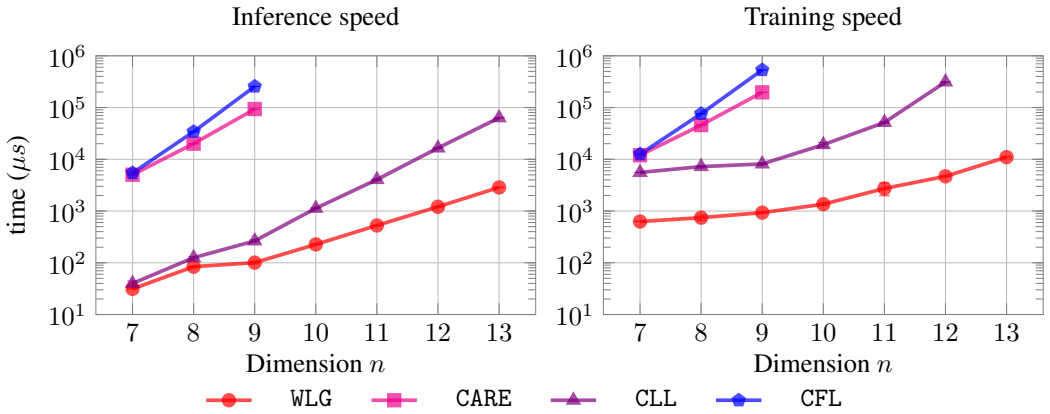

Figure 4: Inference (CUDA-event GPU time) and training (forward and backward, wall-clock) time as a function of n at batch size 8192, where n is the dimension of the Clifford algebra $\mathrm{Cl}(n)$. `WLG` consistently outperforms every baseline at every dimension, with the gap widening as n grows. Against the strongest baseline `CLL` , `WLG` reaches a $14\times$ speedup in inference and a $67\times$ speedup in training at $n = 12$. The geometric-product route `CFL` and the einsum hybrid `CARE` fail to complete within the one-second budget past $n = 9$. Measurements exceeding 1 second are omitted.

Speed and memory. `WLG` outperforms all three baselines on both speed and peak memory across every n. Figure 4 reports inference and training wall-clock time, and Figure 6 reports peak GPU memory. `CFL` and `CARE` are already two orders of magnitude slower than `WLG` at $n = 7$ and fail to complete for $n \geq 10$ within the one-second budget, leaving `CLL` as the only baseline within an order of magnitude of `WLG` on speed. Even at $n = 7$, where the gap is smallest, `WLG` runs inference in $31.3 \pm .36$ μs versus 39.9 ± 4.63 μs for `CLL` and trains in 627 ± 2.57 μs versus $5.57 \pm .01$ ms, an $8.9\times$ gap. By $n = 12$, `WLG` runs inference in 1210 ± 5.64 μs, a $13.7\times$ improvement over `CLL` 's $16.5 \pm .12$ ms. The training gap is even larger with `WLG` taking $4.69 \pm .09$ ms while `CLL` takes $312 \pm .72$ ms, a $66.5\times$ difference.

Method	FLOPs	Memory
Geometric product	2^{2n+1}	2^{n-1}
Matrix	$\frac{2^{2n+1}}{\sqrt{\pi n}}$	$\frac{2^{2n}}{\sqrt{\pi n}}$
Wedge-lifted givens	$n^2 2^{n-1}$	2^{n-2}

Table 1: Approximate FLOPs and component memory of the geometric product, matrix, and wedge-lifted givens application of a rotor from the algebra $\mathrm{Cl}(n)$. The wedge-lifted givens approach has better FLOP and memory scaling. Derivations and exact quantities are given in Appendix A.

The memory picture is similar. `CFL` exceeds the 10GB budget by $n = 10$. `WLG` uses the least memory at every n: at $n = 7$ it uses 0.0061 MB versus 8.6 MB for `CLL` . By $n = 12$, `WLG` uses 0.33 MB while `CLL` uses 75.7 MB, a $229\times$ reduction, and `CARE` uses 383 MB, $1158\times$ more than `WLG` , though its speed disadvantage makes the memory comparison secondary. `WLG` 's speed advantage and absolute memory saving both grow with n, consistent with the scaling in Table 1.

Figure 5 reports `WLG` inference time as the batch size grows from 1 to 16K for $n \in \{7, \ldots, 13\}$. At small batch sizes, time grows slowly because the GPU is memory bound. At larger batch sizes, time approaches linear scaling as the GPU moves into the compute bound regime with the time doubling from batch size 8K to 16K at $n \geq 12$. Even at $n = 13$ and batch size 16K, `WLG` completes in 5.7 ms, about 0.35 μs per multivector.

Can baselines be improved? We note that the baselines have little optimization headroom. `CLL` applies the block-diagonal action as a single batched matmul via `F.linear` and `CARE` uses an `einsum`, both of which dispatch to optimal kernels. `CFL` 's memory usage is high despite its small representation because the geometric product is faster when the all-pairs blade interactions are materialized as intermediates rather than recomputed, and even then, the matrix route remains faster.

6.2 Scaling up an application

Scaling rotor application. We now demonstrate what becomes feasible when rotor application is no longer the bottleneck. `CLL` introduces a rotor-based parametrization of linear layers, motivated by the observation that linear layers dominate parameter count in modern LLMs while admitting a compact

Clifford-algebraic structure [Pence et al., 2025]. A linear map $\mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ is decomposed into rotor sandwich products $\psi_r(x) = rxr^\dagger$ acting on 2^n-dimensional Clifford subspaces of the input, with width c_1 and depth c_2 controlling parallel and sequential rotor maps. Each rotor is parametrized by a bivector with $\binom{n}{2}$ coefficients, so a layer with width c_1 and depth c_2 uses $c_1 c_2 \binom{n}{2}$ parameters in place of $d_{\text{in}} d_{\text{out}}$, where $2^n \leq \min(d_{\text{in}}, d_{\text{out}})$.

`CLL` replaces the query, key, and value projections in just **one to three attention layers** of `LLaMA-3.2 1B` and `3B` [Touvron et al., 2023], `Qwen-2.5 1.5B` [Qwen et al., 2025] and `Fox-1.0 1.6B` [Hu et al., 2025], training each replacement by minimizing the MSE between the rotor layer's output and the original projection's output on hidden states extracted from the pretrained model. When multiple layers are replaced, they are trained sequentially, with each new layer fit on top of the previously replaced ones. Pence et al. [2025] reports that the matrix route to the sandwich is the dominant runtime and memory cost. With our algorithm removing the sandwich product as the limiting factor, we target a setting with far more layers to replace: *mixture-of-experts (MoE) models*.

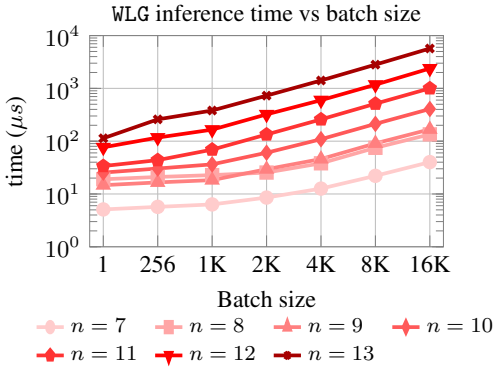

Figure 5: `WLG` inference time as a function of batch size for different $\mathrm{Cl}(n)$. Time grows slowly at small batch sizes and approaches linear scaling at larger batch sizes.

Replacement setup. An MoE layer routes each token to a small subset of expert subnetworks, replacing a single dense feed-forward block with many sparsely activated ones [Shazeer et al., 2017, Fedus et al., 2022]. Here, each expert in the models we consider is a SwiGLU feed-forward block [Shazeer, 2020] parametrized by three linear layers $W_{\text{gate}}, W_{\text{up}} \in \mathbb{R}^{d_e \times d}$ and $W_{\text{down}} \in \mathbb{R}^{d \times d_e}$, where d is the model hidden size and d_e is the expert hidden size. The expert computes

$$\text{Expert}(x) = W_{\text{down}}\big(\text{SiLU}(W_{\text{gate}}x) \odot W_{\text{up}}x\big). \tag{20}$$

A router selects the top-k experts per token from a pool of E experts in each layer. The three projections per expert are independent linear layers, so each MoE layer contains $3E$ candidate replacement targets. We follow the replacement protocol of Pence et al. [2025]. We replace the three projections of all E experts in an MoE layer of `ERNIE-4.5-21B-A3B` [Team, 2025] and `Qwen3-30B-A3B` [Yang et al., 2025].

Scale enabled by the algorithm. The prior work targets 1B-3B parameter models, replacing 3 matrices per layer in up to 3 layers, for 9 replacements in total. We scale to 21B and 30B parameters, replacing all $3E$ matrices per layer in half of each model's MoE layers. `ERNIE-4.5-21B-A3B` has $E = 64$ experts per layer and 28 layers, so 14 replaced layers gives $3 \times 64 \times 14 = 2688$ matrices. `Qwen3-30B-A3B` has $E = 128$ experts per layer and 48 layers, so 24 replaced layers gives $3 \times 128 \times 24 = 9216$ matrices, $1024\times$ more rotors than the prior work, made possible by the speed and memory gains in Figures 4 and 6. To verify that quality holds under scaled-up replacement, we measure log-perplexity on `C4` [Dodge et al., 2021] at 50% replacement. The rotor-replaced `Qwen3-30B-A3B` reaches 2.834 versus the baseline's 2.630, and `ERNIE-4.5-21B-A3B` reaches 2.661 versus 2.336. These gaps are comparable to those reported by Pence et al. [2025] at much smaller replacement counts, showing that scaling up by orders of magnitude does not meaningfully degrade quality.

7 Related Works

Clifford algebra in machine learning. Clifford algebra has driven a growing body of work in machine learning across both applied and methodological directions. In the physical sciences, Clifford-based models represent physical fields as multivectors for PDE modeling [Brandstetter et al., 2023, Alberto Pepe, 2025, Pepe et al., 2024a, Szarvas and Zhdanov, 2025], and Lorentz symmetry is built directly into the algebra over four-dimensional spacetime for high-energy physics [Spinner et al., 2024]. Robot manipulation policies use projective geometric algebra to represent rigid motions [Sun et al., 2025], and in computer vision, geometric-algebra representations have been used

for pose regression [Pepe et al., 2024b], 3D line alignment [Pepe et al., 2025b], and optical flow estimation [Pepe et al., 2025a].

Another line develops general-purpose Clifford architectures. Clifford group equivariant networks build $O(n)$- and $E(n)$-equivariant layers via the Clifford group, whose action preserves multivector grading [Ruhe et al., 2023a], while geometric Clifford algebra networks parametrize learnable Pin-group actions on multivectors [Ruhe et al., 2023b]. Geometric algebra transformers add a scalable $E(3)$-equivariant transformer [Brehmer et al., 2023], later generalized across metrics [de Haan et al., 2024]. Other architectures include Clifford-steerable CNNs [Zhdanov et al., 2024], multivector graph networks [Liu et al., 2024], Clifford-equivariant diffusion [Liu et al., 2025], Clifford normalizing flows [Alesiani and Maruyama, 2024], and rotor-based linear layers [Pence et al., 2025]. Theoretical work recasts ReLU training as convex optimization over wedge-product features [Pilanci, 2024], develops randomized algorithms over Clifford algebras [Wang et al., 2024], and shows that tensor product representations are a special case of binding in Clifford algebra [Pence et al., 2026]. Across most of these settings the algebras remain small ($n \leq 4$), since scaling is blocked by the exponential cost of the underlying operations.

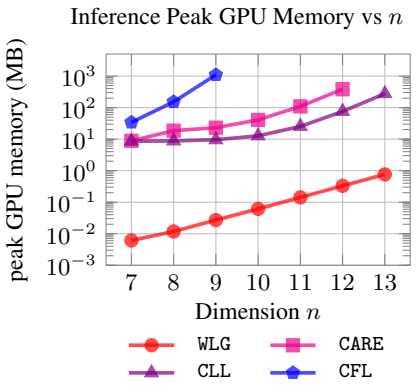

Figure 6: Peak GPU memory during inference at batch size 1 as a function of n. `WLG` uses the least memory at every dimension, and its absolute saving grows with n. Against the strongest speed baseline `CLL`, `WLG` uses $229\times$ less memory at $n = 12$. At higher batch sizes, data rather than weights dominates memory.

Structured linear layers. A parallel line explores structured alternatives to linear maps to reduce parameter and compute cost: circulant [Yu et al., 2014], Toeplitz [Sindhwani et al., 2015], Walsh-Hadamard [Alam et al., 2024], Block-Hadamard [Zeng et al., 2023], butterfly and Monarch factorizations [Dao et al., 2019, 2022], and Clifford rotor parametrizations [Pence et al., 2025], which have also been used for on-chip weight generation in MoE inference [Moon et al., 2027].

8 Limitations

The algorithm implements the rotor sandwich product, a common operation in Clifford algebra. However, the more general geometric product between arbitrary multivectors remains expensive and is not covered by our construction. This is an important direction for future work.

The algorithm requires a custom CUDA kernel to run quickly and avoid memory latency. For other architectures, work would have to be done to implement the algorithm at a low enough level to get the speed gains. The bank-conflict-free construction also relies on groups of 32 threads. While the construction extends to other group sizes, the rank conditions would have to be reworked if the hardware differed enough. Every NVIDIA GPU since CUDA's introduction in 2006 uses 32-thread warps, so this assumption holds across current hardware.

9 Conclusion and Future Work

We presented a wedge-lifted Givens algorithm for the Clifford rotor sandwich that operates directly on the $\binom{n}{2}$ bivector parameters, without forming the rotor coefficients or the $2^n \times 2^n$ action matrix. The algorithm reduces FLOPs and representation size below both the geometric-product route and the matrix route, with the gap widening as n grows. Empirically, the algorithm runs up to $67\times$ faster and uses up to $229\times$ less peak GPU memory than the strongest baselines from recent Clifford-based work. Our algorithmic improvement opens the door to Clifford-based architectures in regimes where larger algebras have so far been impractical. Future work includes extending the construction beyond the Euclidean algebra $\mathrm{Cl}(n)$ to mixed-signature algebras $\mathrm{Cl}(p, q)$. There, the planar factors would include hyperbolic rotations, and because $\mathrm{SO}(p, q)$ is no longer connected, exponentials of bivectors reach only the transformations connected to the identity.

Acknowledgments and Disclosures

Pence, Yamada, and Singh were supported in part by R01AG092220 and Igniting Interdisciplinary Innovation initiative at UW-Madison. Authors declare competing-interests/financial-interests disclosures as follows: Sankaralingam (NVIDIA) and Singh (Google DeepMind).

References

Mohammad Mahmudul Alam, Alexander Oberle, Edward Raff, Stella Biderman, Tim Oates, and James Holt. A walsh hadamard derived linear vector symbolic architecture. *Advances in Neural Information Processing Systems*, 37:2711–2733, 2024.

Mattia Montanari Alberto Pepe, Joan Lasenby. Fengbo: a clifford neural operator pipeline for 3D PDEs in computational fluid dynamics. In *International Conference on Learning Representations*, 2025.

Francesco Alesiani and Takashi Maruyama. Clifford flows. In *Machine Learning and the Physical Sciences Workshop, NeurIPS*, 2024.

Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K. Gupta. Clifford neural layers for pde modeling. *International Conference on Learning Representations*, 2023. doi: 10.48550/arxiv.2209.04934.

Johann Brehmer, Pim de Haan, Sönke Behrends, and Taco Cohen. Geometric algebra transformer. *Advances in Neural Information Processing Systems*, 2023. doi: 10.48550/arxiv.2305.18415.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *International conference on machine learning*, pages 1517–1527. PMLR, 2019.

Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pages 4690–4721. PMLR, 2022.

Pim de Haan, Taco Cohen, and Johann Brehmer. Euclidean, projective, conformal: Choosing a geometric algebra for equivariant transformers. *International Conference on Artificial Intelligence and Statistics*, 2024. doi: 10.48550/arxiv.2311.04744.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 1286–1305, 2021.

Leo Dorst, Daniel Fontijne, and Stephen Mann. *Geometric algebra for computer science (revised edition): An object-oriented approach to geometry*. Morgan Kaufmann, 2009.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Jean Gallier and Dianna Xu. Computing exponentials of skew-symmetric matrices and logarithms of orthogonal matrices. *International Journal of Robotics and Automation*, 18(1):10–20, 2003.

David Hestenes and Garret Sobczyk. *Clifford algebra to geometric calculus: a unified language for mathematics and physics*, volume 5. Springer Science & Business Media, 2012.

John E Hopcroft and Richard M Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

Zijian Hu, Jipeng Zhang, Rui Pan, Zhaozhuo Xu, Shanshan Han, Han Jin, Alay Dilipbhai Shah, Dimitris Stripelis, Yuhang Yao, Salman Avestimehr, Tong Zhang, and Chaoyang He. Fox-1: Open small language model for cloud and edge, 2025. URL `https://arxiv.org/abs/2411.05281`.

Vamsi K Ithapu, Risi Kondor, Sterling C Johnson, and Vikas Singh. The incremental multiresolution matrix factorization algorithm. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2951–2960, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL `https://arxiv.org/abs/1412.6980`.

Risi Kondor, Nedelina Teneva, and Pramod K Mudrakarta. Parallel mmf: a multiresolution approach to matrix computation. *arXiv preprint arXiv:1507.04396*, 2015.

Cong Liu, David Ruhe, and Patrick Forré. Multivector neurons: Better and faster o (n)-equivariant clifford graph neural networks. *arXiv preprint arXiv:2406.04052*, 2024.

Cong Liu, Sharvaree Vadgama, David Ruhe, Erik Bekkers, and Patrick Forré. Clifford group equivariant diffusion models for 3d molecular generation. *arXiv preprint arXiv:2504.15773*, 2025.

Pertti Lounesto. *Clifford Algebras and Spinors*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2 edition, 2001a.

Pertti Lounesto. Clifford algebras and spinors. In *Clifford algebras and their applications in mathematical physics*, pages 25–37. Springer, 2001b.

Chanyoung Moon, Travis Pence, Daisuke Yamada, Jiaqi Mo, Yanxin Zhang, Vikas Singh, and Karthikeyan Sankaralingam. FORGE: Procedural weight generation for sparse MoE inference. In *Proceedings of the 32nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 433–448. ACM, 2027. doi: 10.1145/3845814.3850033.

Travis Pence, Daisuke Yamada, and Vikas Singh. Composing linear layers from irreducibles. In *Advances in Neural Information Processing Systems*, 2025.

Travis Pence, Daisuke Yamada, and Vikas Singh. Recursive binding on a budget: Subspace carving in order-p tensor memories. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, pages 97869–97892. PMLR, 2026. URL `https://proceedings.mlr.press/v306/pence26a.html`.

Alberto Pepe, Sven Buchholz, and Joan Lasenby. Ga-relu: an activation function for geometric algebra networks applied to 2d navier-stokes pdes. In *ICLR 2024 Workshop on AI4DifferentialEquations In Science*, 2024a.

Alberto Pepe, Joan Lasenby, and Sven Buchholz. Cgaposenet+ gcan: A geometric clifford algebra network for geometry-aware camera pose regression. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 6593–6603, 2024b.

Alberto Pepe, Paulo Dos Santos Mendonca, and Joan Lasenby. Geometric inductive priors in diffusion-based optical flow estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 644–654, 2025a.

Alberto Pepe, Yuxin Yao, and Joan Lasenby. Define, refine, align: Correspondence-free 3d line alignment with attentional, equivariant and rotational layers. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 4501–4511, 2025b.

Mert Pilanci. From complexity to clarity: Analytical expressions of deep neural network weights via clifford algebra and convexity. *Transactions on machine learning research*, 2024.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

David Ruhe, Johannes Brandstetter, and Patrick Forré. Clifford group equivariant neural networks. *Advances in Neural Information Processing Systems*, 2023a. doi: 10.48550/arxiv.2305.11141.

David Ruhe, Jayesh K. Gupta, Steven de Keninck, Max Welling, and Johannes Brandstetter. Geometric clifford algebra networks. *International Conference on Machine Learning*, 2023b. doi: 10.48550/arxiv.2302.06594.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *International Conference on Learning Representations*, 2017.

Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. *Advances in Neural Information Processing Systems*, 28, 2015.

John Snygg. *A new approach to differential geometry using Clifford's geometric algebra*. Springer Science & Business Media, 2011.

Jonas Spinner, Victor Bresó, Pim de Haan, Tilman Plehn, Jesse Thaler, and Johann Brehmer. Lorentz-equivariant geometric algebra transformers for high-energy physics. *Advances in Neural Information Processing Systems*, 2024. doi: 10.48550/arxiv.2405.14806.

Sameeksha Sriram, Ayush Paliwal, Alexander S. Ecker, and Chase van de Geijn. Clifford algebraic rotor embeddings: Maybe embeddings should start to care. *NeurIPS Workshop*, 2025. doi: 10.48550/arxiv.2511.11665.

Xiatao Sun, Yuxuan Wang, Shuo Yang, Yinxing Chen, and Daniel Rakita. Hybrid diffusion policies with projective geometric algebra for efficient robot manipulation learning. *arXiv preprint arXiv:2507.05695*, 2025.

Bálint László Szarvas and Maksim Zhdanov. Conditional clifford-steerable cnns with complete kernel basis for pde modeling. *arXiv preprint arXiv:2510.14007*, 2025.

Baidu ERNIE Team. Ernie 4.5 technical report. `https://ernie.baidu.com/blog/publication/ERNIE_Technical_Report.pdf`, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL `https://arxiv.org/abs/2302.13971`.

Yifei Wang, Sungyoon Kim, Paul Chu, Indu Subramaniam, and Mert Pilanci. Randomized geometric algebra methods for convex neural networks. *arXiv preprint arXiv:2406.02806*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL `https://arxiv.org/abs/2505.09388`.

Felix Yu, Sanjiv Kumar, Yunchao Gong, and Shih-Fu Chang. Circulant binary embedding. In *International conference on machine learning*, pages 946–954. PMLR, 2014.

Zhanpeng Zeng, Michael Davies, Pranav Pulijala, Karthikeyan Sankaralingam, and Vikas Singh. Lookupffn: making transformers compute-lite for cpu inference. In *International Conference on Machine Learning*, pages 40707–40718. PMLR, 2023.

Maksim Zhdanov, David Ruhe, Maurice Weiler, Ana Lucic, Johannes Brandstetter, and Patrick Forré. Clifford-steerable convolutional neural networks. In *International Conference on Machine Learning*, 2024. URL `https://arxiv.org/abs/2402.14730`.

Appendix

A Proofs and Complexity

A.1 Proofs

Proof of Theorem 1. Fix $\theta \in \mathbb{R}$ and let $c = \cos\theta$, $s = \sin\theta$. Let $I = \{i_1 < \cdots < i_k\} \subseteq \{1, \ldots, n\}$ index a basis blade e_I of grade k. By definition of the exterior power,

$$\bigwedge^k G_{(p,q)}(e_I) = G_{(p,q)}e_{i_1} \wedge G_{(p,q)}e_{i_2} \wedge \cdots \wedge G_{(p,q)}e_{i_k}.$$

Since $G_{(p,q)}$ acts as the identity on every basis vector outside $\{e_p, e_q\}$, the result depends only on $|I \cap \{p, q\}|$, which takes one of three values.

Case $|I \cap \{p, q\}| = 0$. Every factor $G_{(p,q)}e_{i_j}$ equals e_{i_j}, so $\bigwedge^k G_{(p,q)}(e_I) = e_I$.

Case $|I \cap \{p, q\}| = 2$. The wedge $e_p \wedge e_q$ appears as a factor of e_I. Direct calculation gives

$$G_{(p,q)}e_p \wedge G_{(p,q)}e_q = (ce_p + se_q) \wedge (-se_p + ce_q) = (c^2 + s^2)(e_p \wedge e_q) = e_p \wedge e_q,$$

so this factor is preserved. The remaining factors are fixed by Case 1, and so $\bigwedge^k G_{(p,q)}(e_I) = e_I$.

Case $|I \cap \{p, q\}| = 1$. Without loss of generality take $p \in I$ and $q \notin I$, and define the *partner blade* e_J with $J = (I \setminus \{p\}) \cup \{q\}$. The pair $\{e_I, e_J\}$ spans a two-dimensional subspace, invariant under $\bigwedge^k G_{(p,q)}$, on which the action is

$$\begin{pmatrix} \bigwedge^k G_{(p,q)}(e_I) \\ \bigwedge^k G_{(p,q)}(e_J) \end{pmatrix} = \begin{pmatrix} c & s\sigma \\ -s\sigma & c \end{pmatrix} \begin{pmatrix} e_I \\ e_J \end{pmatrix},$$

where $\sigma = \sigma_{I \to J} \in \{\pm 1\}$ is the sign induced by reordering the wedge factors of $G_{(p,q)}e_{i_1} \wedge \cdots \wedge G_{(p,q)}e_{i_k}$ into the canonical increasing-index form of e_J. The number of blades I with $|I| = k$ and $|I \cap \{p, q\}| = 1$ is $2\binom{n-2}{k-1}$, pairing into $\binom{n-2}{k-1}$ partner pairs. Cases 1 and 2 fix the remaining blades, so $\bigwedge^k G_{(p,q)}$ acts as $\binom{n-2}{k-1}$ independent updates of the partner-pair form above, with all other basis blades fixed. Each update depends on θ only through c and s, together with the sign $\sigma_{I \to J}$, giving the claim. □

A.2 Complexity

Geometric Product. As a rotor r is an element of the even-subalgebra $\mathrm{Cl}^+(n)$, the rotor contains 2^{n-1} components, giving the memory footprint.

A general multivector $x \in \mathrm{Cl}(n)$ has 2^n components. The sandwich product $rxr^\dagger$ is evaluated sequentially as rx followed by $(rx)r^\dagger$. Computing rx maps each of the 2^n components of x against the 2^{n-1} components of r, requiring 2^{2n-1} multiplications and $\approx 2^{2n-1}$ additions, giving 2^{2n} FLOPs. Evaluating $(rx)r^\dagger$ requires the same number of operations. Multiplying by 2 gives 2^{2n+1} FLOPs.

Matrix Application. The matrix $S = \bigoplus_{k=0}^n \bigwedge^k S$ is a block diagonal matrix with block sizes $\binom{n}{k} \times \binom{n}{k}$. The memory footprint is obtained by summing the squares of the block dimensions,

$$\text{Memory} = \sum_{k=0}^{n} \binom{n}{k}^2 = \binom{2n}{n} \approx \frac{2^{2n}}{\sqrt{\pi n}}.$$

A dense $d \times d$ matrix-vector product requires $2d^2 - d$ FLOPs. The total number of FLOPs is obtained by summing over the blocks,

$$\text{FLOPs} \leq \sum_{k=0}^{n} \left[2\binom{n}{k}^2 \right] = 2\binom{2n}{n} \approx \frac{2^{2n+1}}{\sqrt{\pi n}}.$$

Wedge-Lifted Givens. For the kth block of the matrix, there are $\binom{n-2}{k-1}$ associated indices. There are $\binom{n}{2}$ angles given by the Givens decomposition of which we precompute the $\cos$ and $\sin$. This means we have

$$\text{Memory} = 2\binom{n}{2} + \sum_{k=1}^{n-1} \binom{n-2}{k-1} = 2\binom{n}{2} + 2^{n-2} \approx 2^{n-2}.$$

In Algorithm 1, there are 4 multiplications and 2 additions on $\binom{n-2}{k-1}$ elements. Summing over the $\binom{n}{2}$ rotations gives

$$\text{FLOPs} = \sum_{k=1}^{n-1} 6\binom{n-2}{k-1}\binom{n}{2} = 6\binom{n}{2}2^{n-2} = 3n(n-1)2^{n-2} \approx n^2 2^{n-1}.$$

B Eliminating shared-memory bank conflicts

A natural concern with Algorithm 1 is shared-memory bank conflicts: if two lanes in the same warp instruction hit different addresses on the same bank, the access serializes and costs extra cycles. Whether this happens depends on which blades end up in which banks, which we control through the buffer layout. We show that with the right layout and pair ordering, every warp instruction in the rotation loop is conflict-free.

B.1 Notation and conventions

GPU concepts. A CUDA kernel launches a grid of *thread blocks*; each block is partitioned into *warps* of 32 threads, the unit of SIMT execution. Within a warp the 32 threads are called *lanes*. A single instruction issued by a warp is a *warp instruction*; it executes (logically) on all 32 lanes simultaneously. Each block has a chunk of fast on-chip memory called *shared memory* (SMEM). On NVIDIA Ampere, Hopper, and Blackwell, this memory is divided into 32 hardware *banks*, four bytes wide. The instructions `LDS` and `STS` load and store, respectively, from SMEM. When a warp instruction accesses SMEM, the access ideally completes in one *wavefront*; if two lanes target different addresses on the same bank, the access is serialized into multiple wavefronts (a *bank conflict*), each replay costing extra cycles.

The finite field $\mathbb{F}_2$. $\mathbb{F}_2 = \{0, 1\}$ is the field where addition is XOR ($\oplus$) and multiplication is AND. Vectors over $\mathbb{F}_2$ are bit-strings; matrix-vector multiplication $A \cdot v$ over $\mathbb{F}_2$ replaces each row's dot product with a chain of XORs of selected entries. A binary M is called $\mathbb{F}_2$-linear if a map $v \mapsto M \cdot v$ respects the linear structure of $\mathbb{F}_2^n$.

Algorithmic notation. Recall that $I \subseteq \{1, \ldots, n\}$ indexes a basis blade $e_I \in \mathrm{Cl}(n)$. We identify each blade e_I with its bit-vector $\mathrm{bit}(I) \in \{0,1\}^n$, where bit i is 1 iff $i \in I$. In this section $\mathrm{bit}(I)$ is treated as an element of $\mathbb{F}_2^n$. We refer to each $\bigoplus_k \bigwedge^k G_{(p_r,q_r)}$ as *rotation round* r. For a fixed bivector $e_p \wedge e_q$, the round pairs each blade I that has $p \in I$ and $q \notin I$ with the *partner blade* $J = (I \setminus \{p\}) \cup \{q\}$. Each partner pair (I, J) contributes one 2D rotation. There are

$$\mathrm{ppr} = \sum_{k=1}^{n-1} \binom{n-2}{k-1} = 2^{n-2}$$

partner pairs per round; the kernel processes the same ppr partner pairs for every round.

B.2 Setting

The shared-memory layout we control. Each input multivector lives in a shared-memory buffer `my_buf` of $\dim = 2^n$ elements during processing. The buffer is laid out so that the value of blade I lives at SMEM slot $\pi[I]$, where

$$\pi : \{0, \ldots, \dim - 1\} \to \{0, \ldots, \dim - 1\}$$

is a permutation we get to choose. The bank that holds blade I is then $\pi[I] \bmod 32$. Choosing π well (and then choosing the order in which the kernel processes pairs) is the subject of step A.

The conflict-freeness goal. The kernel processes the ppr pairs of each round in groups of 32, where one group of 32 pairs is handled by one warp instruction. Within that warp instruction the warp issues four SMEM accesses:

- 32 simultaneous `LDS` of the left blade values $\{\texttt{my_buf}[\pi[I_t]]\}_t$,
- 32 simultaneous `LDS` of the right blade values $\{\texttt{my_buf}[\pi[J_t]]\}_t$,
- 32 simultaneous `STS` of the new left values,
- 32 simultaneous `STS` of the new right values.

Conflict-free requires, for each of these four phases, the 32 lane accesses to hit 32 distinct banks. Since the load and store addresses inside one group of 32 are the same, this reduces to: the group's 32 left-banks must be distinct, and its 32 right-banks must be distinct.

So we need to (i) choose π, and (ii) order the ppr pairs of each round so that consecutive groups of 32 each form a *perfect matching* between the 32 left-banks and the 32 right-banks. We split this into Step A (choose π) and Step B (order the pairs).

Recap. To restate the goal concretely. A single warp instruction processes 32 partner pairs. For the warp's left-side LDS to complete in one wavefront, the 32 left blades $I_1, \ldots, I_{32}$ in that group must satisfy $\pi[I_t] \bmod 32$ all distinct (32 distinct banks), and likewise for the 32 right blades $J_1, \ldots, J_{32}$. We have $\binom{n}{2}$ rotation rounds, each contributing ppr partner pairs, and the kernel processes each round's pairs in $\mathrm{ppr}/32$ consecutive warp instructions. The two levers are (i) the permutation π, which determines which bank each blade lives in, and (ii) the order in which the ppr pairs of each round are processed, which determines which 32 pairs end up in each warp instruction.

Note that reordering alone is not enough. If π assigns more than $\mathrm{ppr}/32$ partner pairs of some round to the same bank, then pigeonhole forces at least two of those pairs into the same warp instruction, a conflict, regardless of how the pairs are ordered. We therefore need π to spread each round's partner pairs evenly across the 32 banks first, and only then does the reordering have a chance of being conflict-free. Step A chooses π and Step B chooses the ordering.

B.3 Step A — choosing π to balance banks

Bank-balance cost. Let π be the candidate. Define for each round $r \in \left\{1, \ldots, \binom{n}{2}\right\}$, side $\in \{\text{left}, \text{right}\}$, and bank $b \in \{0, \ldots, 31\}$,

$$\mathrm{count}_\pi(r, \text{left}, b) = |\{(I, J) \in \text{round } r : \mathrm{bank}(I) = b\}|,$$
$$\mathrm{count}_\pi(r, \text{right}, b) = |\{(I, J) \in \text{round } r : \mathrm{bank}(J) = b\}|.$$

In the perfectly balanced case, each bank holds the same number of partner pairs, namely $\mathrm{ppr}/32$. The static cost measures the deviation from this target:

$$\mathrm{Cost}(\pi) = \sum_{r,\, \text{side},\, b} \max\left(0, \mathrm{count}_\pi(r, \text{side}, b) - \mathrm{ppr}/32\right).$$

$\mathrm{Cost}(\pi) = 0$ iff every (round, side, bank) has *exactly* $\mathrm{ppr}/32$ pairs.

We initially tried an LP-SAT formulation over the full permutation space, but this naive approach was unable to reach $\mathrm{Cost}(\pi) = 0$ due to the size of the problem. Instead, we have to use the bit-wise relationship between the paired indices.

(O1) Constant XOR mask within a round. If round r corresponds to bivector $e_p \wedge e_q$, then for every pair (I, J) in round r,

$$\mathrm{bit}(J) = \mathrm{bit}(I) \oplus \mathrm{mask}_r, \qquad \mathrm{mask}_r = 2^p + 2^q.$$

This is because $J = (I \setminus \{p\}) \cup \{q\}$ removes p from I and adds q. In bit-vector terms, bit p flips from 1 to 0 and bit q flips from 0 to 1, with all other bits unchanged. Flipping bits p and q is exactly XORing with the mask $2^p + 2^q$, which depends only on the round, not on the pair.

(O2) Left-blade set is an affine subspace of $\mathbb{F}_2^n$. The set of left-blades in round r, those with $p \in I$, $q \notin I$, is the affine subspace

$$\{I : \mathrm{bit}(I)_p = 1, \mathrm{bit}(I)_q = 0\}$$

of dimension $n-2$. As I ranges over this subspace, $\mathrm{bit}(I)$ takes the 2^{n-2} values of the form $e_p +$ (any vector supported on the $n-2$ free bits), where $e_p \in \mathbb{F}_2^n$ is the p-th standard basis vector.

Restriction to $\mathbb{F}_2$-linear permutations. Consider only permutations of the form

$$\pi[I] = M \cdot \mathrm{bit}(I),$$

where M is an $n \times n$ matrix over $\mathbb{F}_2$, the matrix-vector product is taken over $\mathbb{F}_2$, and the resulting n-bit vector is read as an integer in $\{0, \ldots, \dim - 1\}$. Invertibility of M over $\mathbb{F}_2$ guarantees that $\mathrm{bit}(I) \mapsto M \cdot \mathrm{bit}(I)$ is a bijection on $\mathbb{F}_2^n$, hence π is a valid permutation.

Let A be the $5 \times n$ submatrix of M whose rows produce the bottom 5 bits of $\pi[I]$. The bank index of blade I under this π is the bottom 5 bits of $\pi[I]$:

$$\mathrm{bank}(I) = \pi[I] \bmod 32 = A \cdot \mathrm{bit}(I).$$

The product $A \cdot \mathrm{bit}(I)$ is taken over $\mathbb{F}_2$ and produces a 5-bit vector.

XOR-shift property. For any pair (I, J) in round r, by (O1) and the linearity of A over $\mathbb{F}_2$:

$$\mathrm{bank}(J) = A \cdot \mathrm{bit}(J) = A \cdot (\mathrm{bit}(I) \oplus \mathrm{mask}_r) = \mathrm{bank}(I) \oplus \underbrace{A \cdot \mathrm{mask}_r}_{=:\delta_r \in \mathbb{F}_2^5}.$$

Crucially, δ_r depends *only on the round*, not on the individual pair. Within a single round, every pair's right-bank is its left-bank XORed with one fixed 5-bit constant. So $\mathrm{count}_\pi(r, \mathrm{right}, \cdot)$ is the function $\mathrm{count}_\pi(r, \mathrm{left}, \cdot)$ with its argument shifted by δ_r: the right-side counts are a permutation of the left-side counts. If the left-side counts are uniform (every bank holds $\mathrm{ppr}/32$ pairs), so are the right-side counts.

Rank condition. By (O2), as I ranges over round r's left-blade set, $\mathrm{bit}(I)$ takes the form $e_p + I'$ where I' ranges over $\mathbb{F}_2^{n-2}$ (the free bits, with the p- and q-bits zeroed). Therefore

$$\mathrm{bank}(I) = A \cdot \mathrm{bit}(I) = A \cdot e_p + A^{(p,q)} \cdot I' = (\text{constant}) + A^{(p,q)} \cdot I',$$

where $A^{(p,q)}$ is the $5 \times (n-2)$ matrix obtained by *deleting columns p and q from A*.

The image of an $\mathbb{F}_2$-linear map is uniformly distributed over its codomain iff the map is surjective. Therefore, $\mathrm{bank}(I)$ is uniformly distributed iff $A^{(p,q)}$ has rank 5 over $\mathbb{F}_2$, giving the following theorem.

Theorem (Bank-balance condition for $\mathbb{F}_2$-linear π, Formalized)**.** *Let $n \geq 9$. For an invertible $M \in \mathbb{F}_2^{n \times n}$, the permutation $\pi[I] = M \cdot \mathrm{bit}(I)$ satisfies $\mathrm{Cost}(\pi) = 0$ iff*

$$\mathrm{rank}\left(A^{(p,q)}\right) = 5$$

for every (p, q) with $0 \leq p < q < n$, where A is the $5 \times n$ submatrix of M producing the bottom 5 bits of $\pi[I]$.

This gives $\binom{n}{2}$ rank conditions on the $5 \times n$ matrix A, in addition to the requirement that the full M be invertible.

Why this is tractable. We do not enumerate the $\mathbb{F}_2$-linear family, but rather randomly sample random M and check the rank conditions. A random $5 \times (n-2)$ binary matrix has rank 5 over $\mathbb{F}_2$ with probability $\prod_{i=0}^{4}(1 - 2^{i-(n-2)})$. The $\binom{n}{2}$ rank conditions of Theorem B.3 are not independent, but we find that random sampling still works. Empirically, we observed $85, 7, 4, 21$, and 3 random samples to find an M that satisfies the rank conditions at $n = 9, 10, 11, 12$, and 13 respectively.

Output of Step A. Sampling random M until both (i) M is invertible over $\mathbb{F}_2$, and (ii) A satisfies the rank-5-on-every-$(n-2)$-column condition, gives a permutation π with $\mathrm{Cost}(\pi) = 0$.

B.4 Step B — perfect matching of pairs to warp instructions

Step A spreads each round's partner pairs evenly across the 32 banks, but it does not yet say that the 32 SMEM accesses in any *single warp instruction* land on 32 distinct banks. This additional question is answered by Step B.

Bipartite multigraph. For each round r, build a bipartite multigraph G_r:

- left vertex set $\{0, 1, \ldots, 31\}$ (the 32 left-banks),
- right vertex set $\{0, 1, \ldots, 31\}$ (the 32 right-banks),
- for each partner pair $(I, J) \in$ round r, an edge from $\mathrm{bank}(I)$ to $\mathrm{bank}(J)$. Multi-edges between the same pair of vertices are allowed.

Recall the goal: order the ppr pairs of round r so that consecutive groups of 32 are conflict-free. In the language of G_r, this means partitioning $E(G_r)$ into $\mathrm{ppr}/32$ groups of 32 edges, each a perfect matching of G_r — equivalently, a proper edge-coloring of G_r with $\mathrm{ppr}/32$ colors whose color classes are perfect matchings.

Does such a coloring exist? Let $\Delta = \mathrm{ppr}/32$. After Step A with $\mathrm{Cost}(\pi) = 0$, we have that (1) every left vertex of G_r has degree Δ and (2) every right vertex of G_r has degree Δ. Thus, G_r is Δ-regular bipartite.

Fact 1 (König's edge-coloring theorem, regular bipartite case)**.** *Every bipartite multigraph G has chromatic index $\chi'(G) = \Delta(G)$, the maximum vertex degree. For a Δ-regular bipartite multigraph, the proper edge-coloring uses exactly Δ colors, and each color class is a perfect matching.*

So G_r decomposes into Δ edge-disjoint perfect matchings, exactly Δ groups of 32 conflict-free pairs.

Constructive algorithm. The standard way to construct the König decomposition is to repeatedly extract a perfect matching from the residual multigraph using Hopcroft–Karp [Hopcroft and Karp, 1973] and remove its edges. The residual graph remains regular bipartite at every step, so the next perfect matching exists and after Δ iterations the graph is empty.

Output of Step B. An ordering on the pairs of each round such that, when grouped into consecutive groups of 32, every group is a perfect matching of G_r. The kernel reads those groups in order, meaning each warp instruction sees 32 distinct left-banks and 32 distinct right-banks.

B.5 Combined effect

After Step A ($\mathbb{F}_2$-linear π) followed by Step B (König-decomposition), every warp instruction in the rotation loop is bank-conflict-free. We verified this empirically with Nsight Compute. Conflicts remain in the load and store passes that move data between global memory and SMEM, but these are negligible relative to the rotation loop as they only happen at the start and end of the workload.

C Hyperparameters and Experimental Details

C.1 Speed and Memory Benchmarks

Implementations. We compare four implementations of the rotor sandwich: `WLG` (our wedge-lifted Givens), `CARE` (einsum geometric product factored as outer product, sign multiply, and XOR scatter-add), `CLL` (matrix), and `CFL` (geometric product). All four implementations are verified to compute the same sandwich at $\sim 10^{-6}$ absolute tolerance for $n \in \{7, 8, 9, 10\}$.

Measurement protocol. For speed measurements in Figure 4, we use batch size 8192 and sweep $n \in \{7, \ldots, 13\}$. Each configuration runs 25 warmup forward passes followed by 10 trials of 10 timed iterations each, and we report the median across trials with the standard deviation. For memory measurements, we use batch size 1 to show the representation size difference and report peak memory via `torch.cuda.max_memory_allocated` after a single forward pass, or after a full forward and backward pass for training measurements. We skip configurations above the n at which they first

exceed one second per iteration on speed or 10 GB of peak memory. `CARE` and `CFL` are skipped from $n = 10$ onward on speed, `CLL` is skipped at $n = 13$ on training speed, and `CFL` is skipped at $n \in \{10, \ldots, 13\}$ on memory. We use 25 warmup runs, 10 iterations, and 10 trials for each.

Compute. All benchmarks run on a single NVIDIA A100 (40 GB HBM, fp32). The full speed benchmark takes $\sim$40 minutes. The memory benchmark completes in under a minute.

C.2 MoE Layer Replacement

Models and dataset. We used `Qwen/Qwen3-30B-A3B` (48 layers, 128 experts/layer) and `baidu/ERNIE-4.5-21B-A3B-PT` (28 layers, 64 experts/layer) in `bfloat16`. Replacement-module training data and perplexity evaluation use `allenai/c4`.

Replacement-module training. We follow the training method of Pence et al. [2025]. For each expert and projection (gate, up, down), we train one rotor module independently against (x, y) pairs by minimizing MSE in fp32. We use Adam [Kingma and Ba, 2017] with learning rate 0.05, a cosine schedule, no weight decay, batch size 256, and one epoch. The rotor configuration has two hidden layers, PReLU activations, normalization enabled, a single rotor, and fixed permutations enabled. We replace two layers per stage. After each stage, we attach LoRA adapters with rank 8, $\alpha = 32$, and dropout 0.05 to the expert projections and the router on all layers not scheduled for replacement, and fine-tune for one epoch with AdamW at learning rate 2×10^{-5}, a cosine schedule, sequence length 512, and bfloat16 precision. Adapters accumulate across stages.

Compute. These experiments run on 2$\times$ NVIDIA A100-PCIE-40GB for roughly 200 hours. Each replacement job uses 2 GPUs.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: We claim that the algorithm we present has much better scaling properties than existing implementations. We do a FLOPs analysis and show empirical speed-up and memory reduction against recent existing implementations of the sandwich product in Section 6.

   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: The limitations are discussed in Section 8.

   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?

Answer: [Yes]

Justification: Section 4 walks through the steps to derive Algorithm 1 and Appendix A shows sparsity results. We also walk through proofs in Appendix B.

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.

4. **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: We discuss the experimental setup in Section 6 and additional setup and hyperparameters in Appendix C.

   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.

5. **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: [Yes]

Justification: We include the instructions and codebase in the supplemental material.

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.

6. **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: Yes, these are given in Appendix C.

   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.

7. **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: [Yes]

   Justification: We provide std for the speed benchmark. The memory benchmark is static.

   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.

8. **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: We provide these in Appendix C.

   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).

9. **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: This paper conforms, in every respect, with the NeurIPS Code of Ethics.

   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).

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: There is no risk.

    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).

11. **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: There is no risk.

    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.

12. **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: All datasets, models, and pre-existing code bases used are cited.

    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.

13. **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 release 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.

14. **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: There is no crowdsourcing or research with human subjects.

    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.

15. **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: There is no human participants.

    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.

16. **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: None used for research.

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.