Geometric Clifford Algebra Networks

David Ruhe [1]   Jayesh K. Gupta [2]   Steven de Keninck [3]   Max Welling [4]   Johannes Brandstetter [4]

Abstract

We propose Geometric Clifford Algebra Networks (GCANs) for modeling dynamical systems. GCANs are based on symmetry group transformations using geometric (Clifford) algebras. We first review the quintessence of modern (plane-based) geometric algebra, which builds on isometries encoded as elements of the $\mathrm{Pin}(p, q, r)$ group. We then propose the concept of group action layers, which linearly combine object transformations using pre-specified group actions. Together with a new activation and normalization scheme, these layers serve as adjustable *geometric templates* that can be refined via gradient descent. Theoretical advantages are strongly reflected in the modeling of three-dimensional rigid body transformations as well as large-scale fluid dynamics simulations, showing significantly improved performance over traditional methods.

1. Introduction

Equipping neural networks with geometric priors has led to many recent successes. For instance, in group equivariant deep learning (Cohen & Welling, 2016; Weiler et al., 2018; Bronstein et al., 2021; Weiler et al., 2021), neural networks are constructed to be equivariant or invariant to group actions applied to the input data.

In this work, we focus on tasks where we expect that the target function is a *geometric transformation* of the input data. Such functions arise ubiquitously in the science of dynamical systems, which is the core experimental domain of this work. Neural surrogates for solving these schemes have been proposed in fluid dynamics (Li et al., 2020b; Kochkov et al., 2021; Lu et al., 2021; Rasp & Thuerey,
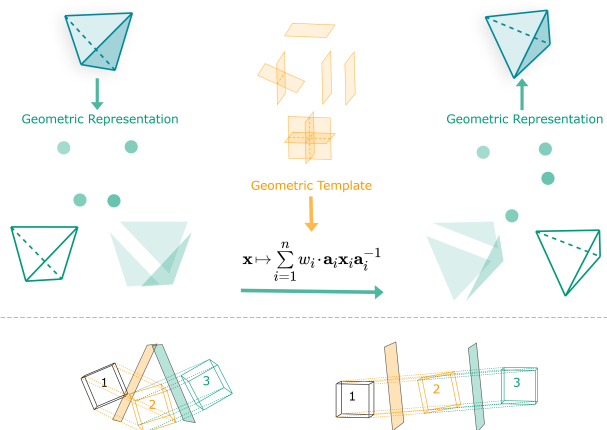

Figure 1. GCANs: Geometric algebra allows us to express data as objects which can simultaneously be interpreted as group elements (top). GCANs parameterize linear combinations of learnable group actions, ensuring that even randomly initialized models form a composition of geometric transformations. GCANs are thus adjustable *geometric templates*. Bottom: geometric transformations can be constructed from compositions of reflections. Two reflections in intersecting planes yield a rotation (left), and two reflections in parallel planes yield a translation (right).

2021; Pathak et al., 2022; Bi et al., 2022; Lam et al., 2022; Nguyen et al., 2023), molecular dynamics (Mardt et al., 2018; Zhong et al., 2019; Greydanus et al., 2019; Mattheakis et al., 2019; Li et al., 2020a), or astrophysics (Tamayo et al., 2016; Cranmer et al., 2021). Typically, the objective is to predict with high precision how a system will evolve based on various initial conditions. This is a challenging task, given that the underlying dynamics can be highly unstable or chaotic.

We introduce Geometric Clifford Algebra Networks (GCANs) as a new approach to incorporate geometry-guided transformations into neural networks using *geometric algebra*. Geometric algebra (Clifford, 1871; Hestenes, 1966; Dorst & Mann, 2002; Mann & Dorst, 2002; Dorst et al., 2009; Artin, 2016) is an algebraic framework based on real Clifford algebras that is particularly well suited to handle and model computational geometry. It has several intriguing properties and advantages over other frameworks, such as classical linear algebra. For example, it naturally and efficiently encodes the transformations and the invariant

[1]Work done during internship at Microsoft Research. [2]Microsoft Autonomous Systems and Robotics Research [3]University of Amsterdam [4]Microsoft Research AI4Science. Correspondence to: David Ruhe <david.ruhe@gmail.com>, Johannes Brandstetter <johannesb@microsoft.com>.

elements of classic geometries. Additionally, in geometric algebra, objects transform covariantly with transformations of space. This means that a single function can transform multiple types of objects, including vectors, points, lines, and planes. Finally, geometric algebra generalizes over dimensions in the sense that transformations and objects are constructed consistently regardless of the dimensionality of the space.

GCANs are built around the concept of *group action layers*, which linearly combine pre-specified group actions to form neural representations. This process can be efficiently implemented using geometric algebra, which encodes both objects and transformations elegantly. By exploiting knowledge of the transformation group actions that govern the underlying dynamics, our approach provides a geometric prior complementary to the symmetry and scale separation principles discussed in Bronstein et al. (2021). In order to ensure the preservation of the vector space of the input representations, we further introduce a specific type of nonlinearity and a new form of normalization. The resulting network can be seen as a composition of geometric transformations, with an uninitialized network serving as a *geometric template* that can be refined through gradient descent learning.

Recently, *Clifford neural layers* (Brandstetter et al., 2022a), which can encode spatial primitives such as scalars and vectors into single entities, were proposed. Most notably, Clifford neural networks can already be seen as geometric templates by allowing Fourier transforms on *multivectors* and geometric products beyond complex numbers or quaternions. However, most Clifford neural layers lack certain geometric guarantees. For example, vector-valued input features might result in multivector-valued quantities that are difficult to interpret geometrically. We will see that specific layers, such as the rotational Clifford convolution layers, are already close to a specific instance of GCAN layers proposed in this work. In fact, the compelling performance of these layers served as motivation for this work.

The theoretical advantages of GCANs are reflected in various dynamical systems modeling tasks. The strong inductive bias of GCANs enhances generalization in low-data regimes and allows for more efficient optimization when there is plenty of data and, therefore, outperforms baselines in both these regimes. We demonstrate these advantages on a rigid body transformation task, where we simulate the motion of Tetris objects in free space. Next, we show excellent performance on two large-scale PDE modeling tasks of fluid dynamic problems, i.e., weather prediction based on the shallow water equations and fluid systems described by incompressible Navier-Stokes equations.

2. Geometric algebra

This section presents a formalization of Euclidean geometry from a geometric algebra perspective, primarily based on Roelfs & De Keninck (2021) and De Keninck & Roelfs (2022). We derive how the $\mathrm{Pin}(n)$ group can model isometries (distance-preserving transformations of metric spaces) and how to use it to obtain geometric templates.

The Pin(n) group. We start the formalization by constructing symmetries (isometries) using *reflections* as our foundation. A reflection is a map from a Euclidean space to itself with a hyperplane as a set of fixed points in which the space gets mirrored. The mathematician Hamilton observed that the composition of two reflections through intersecting planes results in a rotation. This idea is presented in Figure 1 (bottom) and can be generalized to the following theorem.

Theorem 2.1. *Cartan-Dieudonné. Every orthogonal transformation of an n-dimensional space can be decomposed into, at most, n reflections in hyperplanes.*

It is worth noting that isometries composed of an odd number of reflections change the chirality (handedness) of the space, which is often an unwanted property. We refer to such isometries as *improper*. In an n-dimensional space, compositions of reflections construct the $\mathrm{Pin}(n)$ group. A group $(\mathcal{G}, \circ)$ is a non-empty set $\mathcal{G}$ equipped with a binary composition operator $\circ : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ (written here with juxtaposition) that satisfies (i) closure, i.e., for $u, v \in \mathcal{G} : uv \in \mathcal{G}$, (ii) associativity, i.e., for $u, v, w \in \mathcal{G} : (uv)w = u(vw)$, (iii) identity, i.e., there is an element $1 \in \mathcal{G}$, such that for $u \in \mathcal{G} : 1u = u = u1$, and (iv) inverse, i.e., for $u \in \mathcal{G}$ there is an inverse element $u^{-1} \in \mathcal{G}$ such that $uu^{-1} = 1 = u^{-1}u$. Since compositions of reflections (like the ones shown in Figure 1) satisfy all these conditions, they form a group. That is, any element $u \in \mathrm{Pin}(n)$ can be written as composition of k linearly independent reflections: $u = u_1 \ldots u_k$.

Group action: conjugation. A group action on a space is a group homomorphism of the group into the group of transformations of that space. A group can act on itself by the *conjugation* rule, which is a specific map $\mathcal{G} \times \mathcal{G} \to \mathcal{G}$. For $u, v \in \mathrm{Pin}(n)$, we let u act on v via

$$u[v] \mapsto uvu^{-1} , \tag{1}$$

where the group composition is used twice and $uvu^{-1} \in \mathrm{Pin}(n)$. This *sandwich product* tells us how we let a group element act on another (e.g., reflecting a reflection). Intuitively, it mimics what we would do when asked, for example, to write our name upside down: we first rotate the page, write, and rotate the page back. In the following sections, we will see that geometric algebra forms a framework where the objects (e.g., vectors) we want to act on can be interpreted as group elements and vice versa. This allows us to also apply the conjugation rule not only to group elements but also to geometric primitives.

Geometric algebra. Equipped with an understanding of how compositions of reflections yield higher-order orthogonal transformations and how we can let group elements act, we can construct an algebraic implementation of these ideas (Roelfs & De Keninck, 2021) using geometric algebra. Geometric algebra is an emerging tool to model computational geometry (Dorst et al., 2009) based on real Clifford algebras. The geometric product of the algebra allows for intuitive expressions of geometric transformations, making this a natural choice. In an n-dimensional geometric algebra[1] $\mathbb{G}_{pqr}$[2] of $n = p + q + r$, we choose p positive, q negative, and r null basis vectors e_i with

$$e_i e_i \in \{+1, -1, 0\}, \quad e_i e_j = -e_j e_i \quad (i \neq j), \quad (2)$$

where the juxtaposition $e_i e_j$ denotes the algebra's bilinear product. For example, the simplest three-dimensional algebra is $\mathbb{G}_{3,0,0}$, with $p = 3$, $q = 0$, and $r = 0$. A product of k basis vectors is a basis k-blade where the *grade* of the blade is the dimension of the subspace it represents. In this way, vectors with basis components e_i are 1-blades. 2-blades are of the form $e_i e_j$, and so on. In general, an n-dimensional vector space yields 2^n basis blades. The highest grade basis blade $I := e_1 \ldots e_n$ is also known as the *pseudoscalar*. We speak of k-*vectors* by homogeneously combining basis blades of grade k. Vectors, like in linear algebra, are linear combinations of 1-blades. *Bivectors* are formed from 2-blades, etc. A *multivector* $\mathbf{x} \in \mathbb{G}_{p,q,r}$ is a sum of k-vectors, i.e., $\mathbf{x} = [\mathbf{x}]_0 + [\mathbf{x}]_1 + [\mathbf{x}]_2 + \ldots + [\mathbf{x}]_n$, where $[\mathbf{x}]_k$ denotes the k-vector part of $\mathbf{x}$. As an example, $\mathbb{G}_{3,0,0}$ has $2^3 = 8$ basis blades, where a multivector $\mathbf{x}$ is represented via

$$\mathbf{x} = \underbrace{x_0 1}_{\text{Scalar}} + \underbrace{x_1 e_1 + x_2 e_2 + x_3 e_3}_{\text{Vector}}$$
$$+ \underbrace{x_{12} e_{12} + x_{13} e_{13} + x_{23} e_{23}}_{\text{Bivector}} + \underbrace{x_{123} e_{123}}_{\text{Trivector}} . \quad (3)$$

Here, we used $e_{ij} := e_i e_j$. It is worth mentioning that the set of basis blades is closed under multiplication with its elements using Equation (2). The specific choices of algebra (determined by p, q, and r) allows for efficient modeling of many types of geometry.

Geometric product. Multiplication in the algebra is realized via the *geometric product*: a bilinear operation between multivectors. For arbitrary multivectors $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z} \in \mathbb{G}_{pqr}$, and scalar λ, the geometric product has the following properties: (i) closure, i.e., $\mathbf{xy} \in \mathbb{G}_{pqr}$, (ii) associativity, i.e., $(\mathbf{xy})\mathbf{z} = \mathbf{x}(\mathbf{yz})$, (iii) commutative scalar

---

[1]Technically, there are no differences between geometric and (real) Clifford algebra. In fact, Clifford himself chose "geometric algebra". However, it is common practice to use Clifford algebra when primarily interested in mathematical concerns and geometric algebra when interested in geometry.

[2]$\mathbb{G}_{pqr}$ corresponds to $Cl_{p,q,r}(\mathbb{R})$ when using the notation of Brandstetter et al. (2022a).
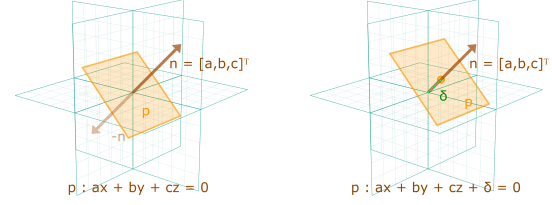

Figure 2. Left, a plane through the origin identified by a normal vector $n = \begin{bmatrix} a & b & c \end{bmatrix}^T$, or equivalently a linear equation $p : ax + by + cz = 0$. Right, a general plane represented by a normal vector n and distance δ, or equivalently by the linear equation $p : ax + by + cz + \delta = 0$.

multiplication, i.e., $\lambda \mathbf{x} = \mathbf{x}\lambda$, (iv) distributivity over addition, i.e., $\mathbf{x}(\mathbf{y} + \mathbf{z}) = \mathbf{xy} + \mathbf{xz}$, and (v) Vectors square to scalars given by a metric norm. The geometric product is in general non-commutative: $\mathbf{xy} \neq \mathbf{yx}$. It can also be applied to (combinations of) lower-grade elements. For example, as shown in Appendix B, for vectors (1-vectors) it exactly results in an inner product part and an antisymmetric part associated with a bivector. In this case, the geometric product directly measures their similarities as well as their differences.

Representing elements of Pin(p, q, r). We can use the geometric algebra $\mathbb{G}_{pqr}$ to represent elements of Pin(p, q, r), where $p + q + r = n$ for an n-dimensional space with p positive, q negative, and r zero dimensions [3]. We saw that the fundamental isometry (from which we build Pin(p, q, r)) is a reflection. To identify reflections, we use the fact that a hyperplane through the origin $p : ax + by + cz + \ldots = 0$ can be mapped onto grade 1 elements of $\mathbb{G}_{pqr}$ via

$$ax + by + cz + \ldots = 0 \leftrightarrow \mathbf{u} := ae_1 + be_2 + ce_3 + \ldots . \quad (4)$$

The grade 1 elements form a vector perpendicular to the surface, i.e., $n := [a, b, c]^\top$ and $[\mathbf{u}]_1 = n$. Note that $\mathbf{u} \in \mathbb{G}_{pqr}$, but the $(k \neq 1)$-blades are simply left zero. This identification is illustrated for the three-dimensional case in Figure 2. Note that, using Equation (4), two unit normals lead to the same geometric plane: one being the negation of the other (also displayed in Figure 2). As such, normalized vectors map 2-to-1 to planes through the origin. Now that we have an algebraic implementation of a hyperplane, it can be shown (Appendix B) that a reflection through that plane amounts to (using geometric products)

$$\mathbf{v} \mapsto -\mathbf{uvu}^{-1} , \quad (5)$$

where $\mathbf{u}, \mathbf{v} \in \mathbb{G}_{p,q,r}$ are vectors (1-vectors in the algebra) and $\mathbf{u}^{-1}$ is the multiplicative inverse such that $\mathbf{uu}^{-1} = 1$.

---

[3]For further theory on representing groups in the geometric algebra, consider drcite
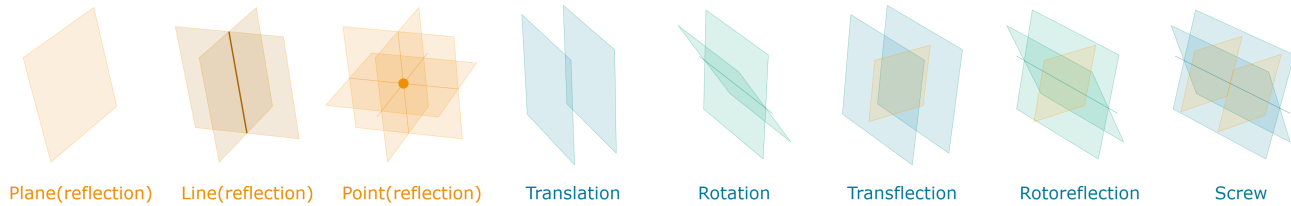
Figure 3. All elements of the Euclidean group can be represented as compositions of reflections in planes. The orange k-blades are compositions of orthogonal planes, and represent simultaneously the points, lines, planes as well as the reflections in these elements. In green: compositions of reflections in arbitrary planes make up all isometric transformations.

The minus sign of Equation (5) comes from the fact that we use two 1-vectors, as explained below. We next use the fact that any $\mathrm{Pin}(p,q,r)$ group element can be written as a composition of k linearly independent reflections. Composition is now rather straightforward: we take Equation (5) and apply the sandwich structure again using geometric products:

$$\mathbf{v} \mapsto \mathbf{u}_2\mathbf{u}_1\, \mathbf{v}\, \mathbf{u}_1^{-1}\mathbf{u}_2^{-1} = (\mathbf{u}_2\mathbf{u}_1)\, \mathbf{v}\, (\mathbf{u}_2\mathbf{u}_1)^{-1}\ , \quad (6)$$

creating a bireflection. We see how we can use the associativity of the geometric product to compose reflections.

As such, we henceforth treat elements of $\mathrm{Pin}(p,q,r)$ as compositions of 1-vectors in the algebra, as opposed to abstract compositions of reflections. In general, for two group elements $\mathbf{u} := \mathbf{u}_1\mathbf{u}_2\ldots\mathbf{u}_k \in \mathrm{Pin}(p,q,r)$ and $\mathbf{v} := \mathbf{v}_1\mathbf{v}_2\ldots\mathbf{v}_l \in \mathrm{Pin}(p,q,r)$, where $\mathbf{u}$ is a k-reflection and $\mathbf{v}$ is an l-reflection, the group action of $\mathbf{u}$ on $\mathbf{v}$ is:

$$\mathbf{v} \mapsto \mathbf{u}[\mathbf{v}] := (-1)^{kl}\mathbf{u}\mathbf{v}\mathbf{u}^{-1}\ , \quad (7)$$

obtaining a similar conjugation rule to Equation (1). The prefactor ensures that we obtain the correct orientation of space. For example, when $\mathbf{u}$ is a reflection ($k = 1$), $\mathbf{u}[\mathbf{u}] = -\mathbf{u}$, meaning that reflecting $\mathbf{u}$ in itself reverses its orientation. Taking the geometric product $\mathbf{u}_1\mathbf{u}_2$ of two vectors (each parameterizing a reflection) with $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{G}_{p,q,r}$ yields bivector components. Considering Hamilton's observation, bivectors thus parameterize rotations. Composing more reflections yields higher-order blades, parameterizing higher-order isometries.

Note that $\mathbf{u}$ and $-\mathbf{u}$ parameterize the same isometry since the sign gets canceled in the sandwich structure of Equation (7). This is visualized in Figure 2, where two vectors parameterize the same plane used to reflect in. We therefore have a 2-to-1 map to any orthogonal transformation, which makes $\mathrm{Pin}(n)$ the double cover of $\mathrm{O}(n)$. I.e., each group element in $\mathrm{O}(n)$ identifies with two elements in $\mathrm{Pin}(n)$. By excluding improper isometries, we obtain the $\mathrm{Spin}(n)$ group, the double cover of the special orthogonal group $\mathrm{SO}(n)$, which is the group of n-dimensional orthogonal transformations excluding reflections.

Projective geometric algebra. We now take a closer look at an instantiation of $\mathrm{Pin}(p,q,r)$: the projective geometric algebra $\mathbb{G}_{3,0,1}$, which is well-suited to model transformations in three-dimensional space. Note that the dimensionality of the algebra is higher than that of the physical space. This is a recurring theme in geometric algebra: higher-dimensional algebras are used to *model* the underlying space. For example, $e_0^2 = \{-1, 0, 1\}$ leads to hyperbolic, projective, and Euclidean geometry, respectively, where we call the fourth 'special' basis vector e_0. The inclusion of the zero element $e_0^2 = 0$ in the algebra allows us to obtain planes that do not pass through the origin of the physical space, which resembles the extra dimension that we are acquainted to when using homogeneous coordinate systems:

$$ax + by + cz + \delta = 0 \leftrightarrow \mathbf{u} = ae_1 + be_2 + ce_3 + \delta e_0 \quad (8)$$

This identification is in Figure 2. Consequently, we are free to construct two *parallel* planes, which, as depicted in Figure 1, can be used to translate. Two intersecting planes still create rotations, but now around a line not necessarily through the origin. Three reflections are improper rotations and reflections, and four reflections lead to screw motions. By including translations, $\mathrm{Pin}(3, 0, 1)$ is the double cover of the three-dimensional Euclidean group $\mathrm{E}(3) = \mathrm{O}(3) \rtimes \mathbb{R}^3$, which is the semi-direct product of $\mathrm{O}(3)$ and the translation group $\mathbb{R}^3$. $\mathrm{E}(3)$ contains all the transformations of three-dimensional Euclidean space that preserve the Euclidean distance between any two points, i.e., translations, rotations, and reflections. As such, we can work with all the rigid motions of Euclidean space by composing reflections (Figure 3). Similarly, $\mathrm{Spin}(3, 0, 1)$, which excludes improper isometries, i.e., those composed of an odd number of reflections, is the double cover of the special Euclidean group $\mathrm{SE}(3) = \mathrm{SO}(3) \rtimes \mathbb{R}^3$, the group of three-dimensional Euclidean isometries excluding reflections.

Representing data as elements of $\mathbf{Pin}(p,q,r)$. We discussed how composing reflections allows us to construct group actions of, e.g., $\mathrm{E}(3)$. However, instead of acting on group elements, we are in practice interested in acting on objects such as vectors, planes, lines, or points. We can naturally construct these by identifying the *invariant sub-*

Reflections	Group element [O(3)]	Invariant subspace†	Algebra element [$\mathbb{G}_{3,0,0}$]
0	Identity	Volume	Scalar
1	Reflection	Plane	Vector
2	Rotation	Line	Bivector
3	Rotoreflection	Point	Trivector

†These subspaces all pass through the origin.

Reflections	Group element [E(3)]	Invariant subspace	Algebra element [$\mathbb{G}_{3,0,1}$]
0	Identity	Volume	Scalar
1	Reflection	Plane	Vector
2	Rotation/translation	Line	Bivector
3	Roto/transflection	Point	Trivector
4	Screw	Origin	Quadvector

Table 1. Overview of elements of $\mathrm{Pin}(3,0,0)$ (top), and $\mathrm{Pin}(3,0,1)$ (bottom). This table relates their group elements, i.e., composition of reflections, to O(3) and E(3) group elements, spatial primitives (identified with invariant subspaces of the transformations), and how they are encoded in geometric algebra. Note that $\mathbb{G}_{3,0,1}$ allows us to encode translations.

spaces (symmetry elements) of the group elements. This was already shown for reflections: they were constructed from planes, which used the 1-vector components of the algebra. Thus, we can relate 1-vectors, planes, and reflections with each other and, most notably, as elements of $\mathrm{Pin}(p,q,r)$. Similarly, a bireflection (Equation (6)) computes a rotation, which preserves a line (see, e.g., Figure 3). Computing the element $\mathbf{u}_1\mathbf{u}_2 \in \mathrm{Pin}(3,0,1)$ using the geometric product yields a bivector. Bivectors (2-vectors) thus parameterize lines, i.e., if our data represents a line, we can use the bivector components of the respective algebra to represent it, and transform it using Equation (7). In this way, we can determine the transformation of any spatial object using conjugation. The group action is now not on G, but rather on X. That is, we consider mappings of the form $\mathcal{G} \times X \to X$ instead of $\mathcal{G} \times \mathcal{G} \to \mathcal{G}$, where X is the space we are interested in. Summarizing, the relationship between $\mathrm{Pin}(p,q,r)$ group elements, spatial primitives, and algebraic elements is shown in Table 1. Specifically, we can identify each row's elements with each other.

3. Geometric algebra neural networks

In the following, we describe the building blocks of GCANs. Crucially, we want to ensure two properties: (i) inputs always map back to their source vector space, and (ii) inputs are transformed by linear combinations of group actions. We call such neural networks *geometric templates*. For CGANs specifically, (i) means the input grades are unchanged or, in other words, k-vectors map to k-vectors, and (ii) means that we use linear combinations of $\mathrm{Pin}(p,q,r)$ actions.

Group action layers. We start by introducing the general concept of a group action layer. Let $\mathcal{G}$ be a group, X a vector space, $\alpha : \mathcal{G} \times X \to X$ a group action, and c a number of input channels. A group action layer then in general form amounts to

$$x \mapsto T_{g,w}(x) := \sum_{i=1}^{c} w_i \cdot \alpha(g_i, x_i) , \qquad (9)$$

where we put $g := (g_i)_{i=1,\dots,c}$ with $g_i \in G$, $x := (x_i)_{i=1,\dots,c}$ with $x_i \in X$, and $w := (w_i)_{i=1,\dots,c}$ with $w_i \in \mathbb{R}$. The *scalar* parameters w determine how the actions are linearly combined ("·" denotes scalar multiplication). This linear combination of group actions ensures that the source object transforms in a geometrically consistent way, regardless of whether it is a point, line, vector, sphere, etc. Both linear and convolutional neural network layers can be constructed from this general notion. As an example, let $X := \mathbb{R}$ and $\mathcal{G} := \mathrm{Aff}(1)$ be the one-dimensional affine group of scaling and addition. Then $g_i = (a_i, b_i) \in \mathcal{G}$ and $\alpha(g_i, x_i) = a_i x_i + b_i$. In this case, we recover the standard linear layers, where

$$T_{g,w}(x) = \sum_{i=1}^{c} w_i \cdot (a_i x_i + b_i) = \sum_{i=1}^{c} w_i' x_i + b' , \quad (10)$$

with $w_i' := w_i a_i$ and $b' := \sum_{i=1}^{c} w_i b_i$.

GCA linear layers. Using geometric algebra, we can now easily encode group transformations and objects into group action layers. In this case,

$$T_{g,w}(\mathbf{x}) = \sum_{i=1}^{c} w_i \cdot \mathbf{a}_i \mathbf{x}_i \mathbf{a}_i^{-1} , \qquad (11)$$

where now $\mathbf{x}_i \in X := \mathbb{G}_{p,q,r}$, and $\mathbf{a}_i \in \mathcal{G} := \mathrm{Spin}(p,q,r)$. The sandwich operation, determined by $\mathbf{a}_i$, represents the group action and can be parameterized. For example, to encode rotations, we parameterize bivector components and
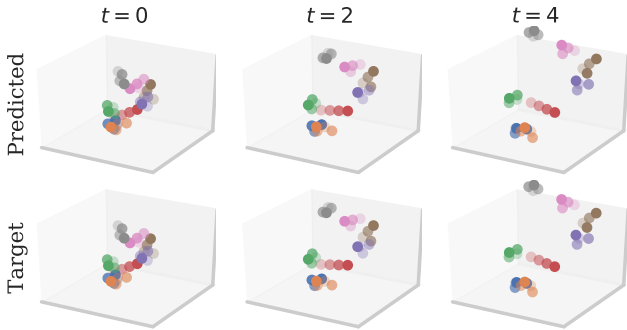
Figure 4. Tetris trajectories. Exemplary predicted (top) and ground-truth (bottom) states. Predictions are obtained by the GCA-GNN model when using 16384 training trajectories.

optimize them using techniques like stochastic gradient descent. This can be viewed as an extension of Clifford neural layers (Brandstetter et al., 2022a) where the geometric product is replaced with the sandwich product, and the transformations are linearly mixed via the scalar parameters w. Equation (11) ensures that when $\mathbf{x}_i$ is a sum of k-vectors, each k-vector transforms as a k-vector, i.e., object types are preserved (see Theorem B.1). For example, when $\mathbf{x}_i$ are 1-vectors, $T_{g,w}(\mathbf{x})$ will be a 1-vector. Clifford (Pearson, 2003), complex (Trabelsi et al., 2017), or quaternion (Parcollet et al., 2020) networks generally do not exhibit this property, meaning that a k-vector would transform into an unstructured multivector[4]. The flexibility of geometric algebra allows the practitioner to determine which group actions to parameterize and how to efficiently represent the data. Furthermore, due to the sparsity of the sandwich product, GCA layers scale better with the algebra dimension than Clifford layers.

Note that Equation (11) computes a "single-channel" transformation, similar to how Equation (10) computes a single-channel value (typically referred to as a neuron). In practice, we extend this to a specified number of output channels by applying the linear transformations in parallel using different weights and transformations.

GCA nonlinearity and normalization. We are interested in nonlinearities that support the idea of geometric templates. Therefore, we propose the following *Multivector Sigmoid Linear Unit* that gates (Ramachandran et al., 2017; Weiler et al., 2018; Sabour et al., 2017) a k-vector by applying

$$[\mathbf{x}]_k \mapsto \mathrm{MSiLU}_k(\mathbf{x}) := \sigma\left(f_k(\mathbf{x})\right) \cdot [\mathbf{x}]_k, \quad (12)$$

where $f_k(\mathbf{x}) : \mathbb{G}_{p,q,r} \to \mathbb{R}$ and σ is the logistic function. We choose this definition to ensure the geometric template: by only scaling, a k-vector remains a k-vector. In this work, we restrict f_k to be a linear combination of the multivector

[4]The complex and Hamilton (quaternion) product both are instances of Clifford's geometric product.

components, i.e., $f_k(\mathbf{x}) := \sum_{i=1}^{2^n} \beta_{k,i} \cdot \mathbf{x}_i$, where $\mathbf{x}_i$ denotes the i-th blade component of $\mathbf{x}$. Also, $\beta_{k,i}$ are either free scalar parameters or fixed to $\beta_{k,i} := 1$ or $\beta_{k,i} := 1/m$, resulting in summation or averaging, respectively. Similarly, we normalize a k-vector by applying a modified version of *group normalization* (Wu & He, 2018), where

$$[\mathbf{x}]_k \mapsto s_k \frac{[\mathbf{x}]_k - \mathbb{E}[[\mathbf{x}]_k]}{\mathbb{E}[\|[\mathbf{x}]_k\|]}. \quad (13)$$

The empirical average of $[\mathbf{x}]_k$ is computed over the number of channels specified by the group size hyperparameter, and rescaled using a learnable scalar s_k through $[\mathbf{x}]_k \mapsto s_k \cdot [\mathbf{x}]_k$, which again only scales the k-vector.

4. Experiments

4.1. Tetris

This experiment shows the ability of GCANs to model complex object trajectories. We subject Tetris objects (Thomas et al., 2018), which are initially located at the origin, to random translations and rotations around their respective centers of mass. The rotations and translations are sampled conditionally, introducing a correlation between the objects. We further apply conditional Gaussian noise to the individual parts of each object. The objects move outward from the origin in an exploding fashion, continuously rotating around their own centers of mass. Given four input time steps, the model's objective is to accurately predict the following four time steps. To do so, it has to infer the positions, velocities, rotation axes, and angular velocities and apply them to future time steps, see Figure 4).

We use $\mathbb{G}_{3,0,1}$, as it is particularly well-suited to model Euclidean rigid motions. The highest-order isometry in this algebra is a screw motion (a simultaneous combination of translation and rotation) constructed using four reflections. Crucially, we do not have to parameterize and compute four reflections explicitly. Instead, we use the fact that four reflections lead to a scalar, six bivectors, and a quadvector component. As such, we parameterize these components and refine them via gradient descent. The points of the Tetris objects are implemented as *trivectors*, representing intersections of three planes in a point. That is, a point x, y, z is implemented in $\mathbb{G}_{3,0,1}$ as $xe_{023} + ye_{013} + ze_{012}$.

GCA-MLPs. Figure 5 summarizes various results of the Tetris experiment, showing test mean squared errors (MSE) summed over the number of predicted time-steps as a function of the number of training trajectories. The baseline MLP networks receive unstructured data in the form of a $3 \times 4 \times 4 \times 8 = 384$-dimensional vector, representing three coordinates per point, four points per object, and eight objects over four input time steps. The *MotorMLP* also takes data in an unstructured manner but regresses directly on a rotation matrix and a translation offset, thereby also enforcing
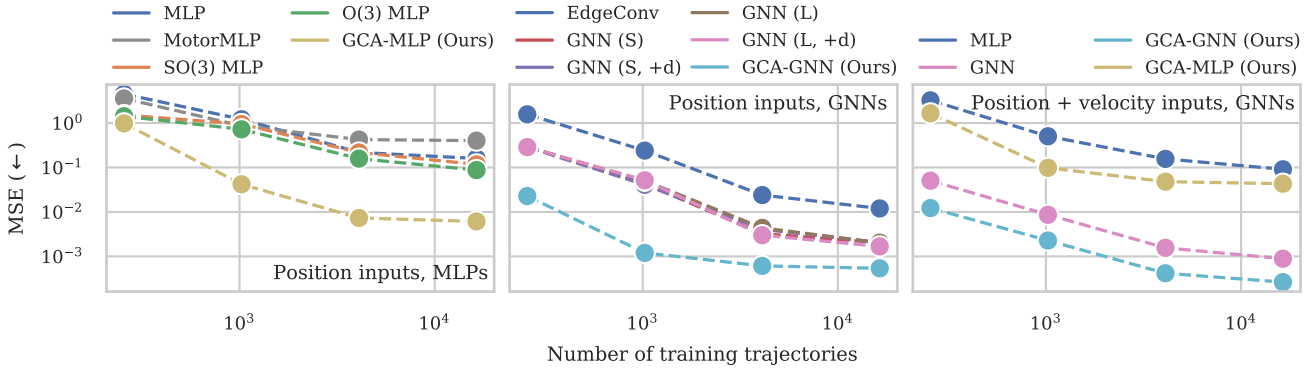
Figure 5. Test MSE results of the Tetris experiment as a function of the number of training trajectories. Left: comparison of different MLP models, center: comparison of different GNN models, right: comparison of the best MLP and GNN models when velocities are included.

a geometric transformation. However, this transformation is indirect, and the learned representations remain entirely unstructured. We also consider two *equivariant* baselines, both EMLP models as presented in Finzi et al. (2021). Indeed, the task has global equivariance properties; the frame of reference does not affect the trajectory's unfolding. However, *local* motions are what makes the task challenging. As such, we see that the O(3) and SO(3) equivariant models only marginally outperform the baseline MLPs. In contrast, our GCA-MLP significantly outperforms the baseline MLPs.

GCA-GNNs. Next, we test message-passing graph networks (GNN) (Gilmer et al., 2017; Battaglia et al., 2018) where we encode the position coordinates as nodes and thus provide a strong spatial prior to the learned function. We consider small (S) and large (L) versions, where the small ones have the same number of parameters as the baseline MLPs. We incorporate versions that include the relative distances (+d) between the coordinates in the messages. We further include the dynamic graph convolutional network *EdgeConv* (Wang et al., 2019) in the baselines. Finally, we introduce GCA-GNNs, which apply the same message-passing algorithm as the baseline GNNs, but replace message and node update networks with GCA-MLPs, i.e.,

$$\mathbf{m}_{ij} := \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l), \quad \mathbf{h}_i^{l+1} := \phi_h(\bar{\mathbf{m}}_i, \mathbf{h}_i^l), \quad (14)$$

where $\mathbf{h}_i^l \in \mathbb{G}_{3,0,1}^c$ are the node features at node i, $\mathbf{m}_{ij} \in \mathbb{G}_{3,0,1}^c$ are messages between node i and node j, and $\bar{\mathbf{m}}_i \in \mathbb{G}_{3,0,1}^c$ is the message aggregation for each node. We put $\mathbf{h}_i^0 := \mathbf{x}_i$, where $\mathbf{x}_i$ are the point coordinates embedded in the algebra $\mathbb{G}_{3,0,1}$. The combination of graph structure and geometric transformations (GCA-GNNs) outperforms all the baselines, sometimes by a large margin.

Next, we show how we naturally combine objects in GCANs without sacrificing expressiveness. We do so by including the discretized velocities of the points at all time steps as model inputs, predicting future coordinates and future velocities. In GCANs, we can include the velocities directly

as vector components e_1, e_2, and e_3 alongside the trivector components that encode the object positions, and use the exact same neural network architecture to transform them. Effectively, having more components increases the number of numerical operations used to compute Equation (11). However, the number of parameters stays the same, and weights are now shared between multivector components. Consequently, both attributes then are simultaneously subjected to Euclidean rigid motions throughout the network. In contrast, the baseline models must account for the additional input and output of velocity data, requiring a reduction in the size of the hidden layers to maintain the same number of parameters.

4.2. Fluid mechanics

In the following experiments, we aim to learn large-scale partial differential equation (PDE) surrogates on data obtained from numerical solvers. To be more precise, we aim to learn the mapping from some fields, e.g., velocity or pressure fields, to later points in time. In this work, we investigate PDEs of fluid mechanics problems. Specifically, we focus on the $2 + 1$-dimensional shallow water (Vreugdenhil, 1994) and the $2 + 1$-dimensional incompressible Navier-Stokes equations (Temam, 2001).

Shallow water equations. The shallow water equations describe a thin layer of fluid of constant density in hydrostatic balance, bounded from below by the bottom topography and from above by a free surface. As such, the shallow water equations consist of three coupled PDEs, modeling the temporal propagation of the fluid velocity in x- and y-direction, and the vertical displacement of the free surface, which is used to derive a scalar pressure field. For example, the deep water propagation of a tsunami can be described by the shallow water equations, and so can a simple weather model. We consider a modified implementation of the `SpeedyWeather.jl` (Klöwer et al., 2022) package, obtaining data on a 192×96 periodic grid ($\Delta x = 1.875°$,
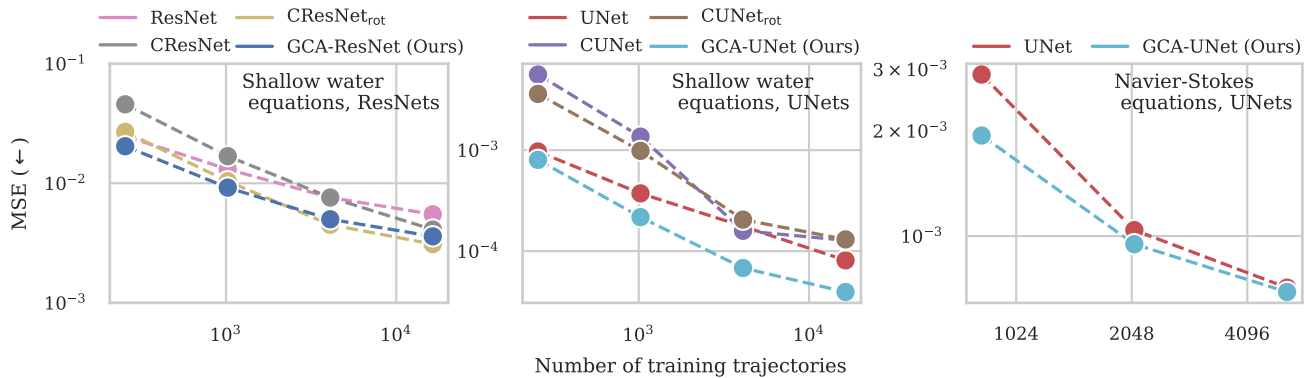
Figure 6. MSE results of the large-scale fluid mechanics experiments as a function of the number of training trajectories. We compare ResNet (left) and UNet (center) models on the shallow water equations. Right: UNet comparison on the Navier-Stokes equations.

$\Delta y = 3.75°$) with temporal resolution of $\Delta t = 6$ h. The task is to predict velocity and pressure patterns 6 hours into the future given four input time steps. Example input and target fields are shown in Figure 7.

GCA-CNNs. When building GCA-CNNs, we use the fact that scalar pressure and vector velocity field are strongly coupled in the underlying shallow water equations. We, thus, consider them as a single entity in a (higher-dimensional 3D) vector space and assume that it transforms under scaling and rotation. We embed data in $\mathbb{G}_{3,0,0}$ as vectors, constructing 2D convolutional GCA layers of the form

$$\mathbf{x} \mapsto \sum_{i=1}^{c \times k \times k} w_i \cdot \mathbf{a}_i \mathbf{x}_i \mathbf{a}_i^{-1} , \qquad (15)$$

where k denotes a pre-specified kernel size, and c is the number of input channels. The group action weights $\mathbf{a}_i$ are equipped with nonzero scalar and bivector components, yielding rotations. In Appendix E.4 we show that in this case, Equation (15) resembles the rotational Clifford layer introduced in Brandstetter et al. (2022a). We compare our methods against residual networks (ResNet) (He et al., 2016), and modern UNet architectures (Ronneberger et al., 2015; Ho et al., 2020), which are considered to be the best-performing models for the task at hand (Gupta & Brandstetter, 2022). We replace their linear layers with layers of the form of Equation (15). Next, we directly replace normalization and nonlinearities with our proposed versions. Altogether, this leads to GCA-ResNet and GCA-UNet architectures. Further, we include the Clifford algebra versions of both models as presented in Brandstetter et al. (2022a). All models are optimized using similar numbers of parameters, i.e., 3 and 58 million parameters for ResNet and UNet architectures, respectively. Hyperparameter choices and further details are summarized in Appendix E.4. The results are shown in Figure 6, reporting the mean-squared error (MSE) loss at a target time step summed over fields.
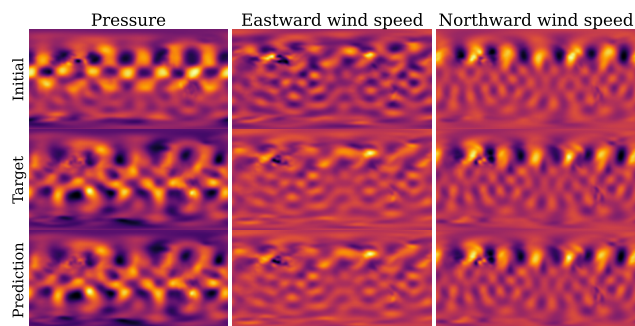

Figure 7. Example input, target, and predicted fields for the shallow water equations. Predictions are obtained by the GCA-UNet model when using 16384 training trajectories.

For ResNets (Figure 6 left), we observe a similar overall picture as reported in Brandstetter et al. (2022a). Our GCA-ResNet and the conceptually similar rotational Clifford ResNet perform best where the overall performance is weak compared to UNet models. The reason is that ResNets as backbone architectures struggle to resolve the local and global processes of PDEs at scale.

For UNets (Figure 6 center), we observe substantial performance gains of our GCA-UNets over baseline architectures, which for larger numbers of training samples is more than a factor of 5. We attribute those performance gains to the strong inductive bias introduced via the geometric template idea. More concretely, for larger backbone architectures such as UNets, which have different resolution, normalization, and residual schemes, it seems crucial that the map from k-vectors to k-vectors through layers and residual blocks is preserved. We thus show that GCA ideas scale to large architectures. An exemplary qualitative result is shown in Figure 7.

Navier-Stokes equations. Finally, we test the scalability of our models on a Navier-Stokes large-scale PDE experi-

ment with a scalar (smoke density) field and a velocity field. The scalar smoke field is *advected* by the vector field, i.e., as the vector field changes, the scalar quantity is transported with it. The equations and simulations are implemented using the ΦFlow (Holl et al., 2020a) package. The grid size for the simulation is 128×128, and the temporal resolution is 1.5s. Similar to the shallow water equations experiment, we embed the scalar and vector components as a single entity in the algebra $\mathbb{G}_{3,0,0}$ that transforms under rotation and scaling. We employ a convolutional layer similar to Equation (15). The results are shown in Figure 6, reporting the mean-squared error (MSE) loss at a target timestep summed over fields. We include the UNet baseline and the respective GCA version. In contrast to the shallow water equations, the coupling between the fields is less pronounced in this experiment. Therefore, our geometric interpretation has a weaker grounding. However, as in the previous experiments, the geometric templates of GCA-UNets prove beneficial.

5. Conclusion

We proposed Geometric Clifford Algebra Networks (GCANs) for representing and manipulating geometric transformations. Based on modern plane-based geometric algebra, GCANs introduce group action layers and the concept of refineable geometric templates. We showed that GCANs excel at modeling rigid body transformations, and that GCANs scale well when applying them to large-scale fluid dynamics simulations.

Limitations and future work. The main limitation of GCANs is that, as observed in Hoffmann et al. (2020) and Brandstetter et al. (2022a), the compute density of (Clifford) algebra operations might lead to slower runtimes and higher memory requirements for gradient computations. Although higher compute density is, in principle, advantageous for hardware accelerators like GPUs, obtaining such benefits can require custom GPU kernels or better modern compiler-based kernel fusion techniques (Tillet et al., 2019; Ansel, 2022). But even without writing custom GPU kernels, we have already managed to reduce training times for convolutional $\mathbb{G}_{3,0,0}$ layers by an order of magnitude to a factor of roughly 1.2 relative to comparable non-GCAN layers using available PyTorch operations. Possible future work comprises extending geometric templates to *geometric object templates*, i.e., transforming whole objects, such as molecules, according to group actions. Further, the covariant transformation of objects in geometric algebra might be exploited to build equivariant architectures (Ruhe et al., 2023). Finally, we aim to leverage the fact that geometric algebra gives us better primitives to deal with object geometry (mesh) transformations and corresponding fluid dynamics around those objects, as seen in, e.g., airfoil computational fluid dynamics (Bonnet et al., 2022).

References

Ansel, J. TorchDynamo, 1 2022. URL https://github.com/pytorch/torchdynamo.

Artin, E. *Geometric algebra*. Courier Dover Publications, 2016.

Batatia, I., Kovács, D. P., Simm, G. N., Ortner, C., and Csányi, G. Mace: Higher order equivariant message passing neural networks for fast and accurate force fields. *arXiv preprint arXiv:2206.07697*, 2022.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Batzner, S., Musaelian, A., Sun, L., Geiger, M., Mailoa, J. P., Kornbluth, M., Molinari, N., Smidt, T. E., and Kozinsky, B. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):1–11, 2022.

Bayro-Corrochano, E. and Buchholz, S. Geometric neural networks. In *International Workshop on Algebraic Frames for the Perception-Action Cycle*, pp. 379–394. Springer, 1997.

Bayro-Corrochano, E. J. Geometric neural computing. *IEEE Transactions on Neural Networks*, 12(5):968–986, 2001.

Behl, A., Paschalidou, D., Donné, S., and Geiger, A. Pointflownet: Learning representations for rigid motion estimation from point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7962–7971, 2019.

Berg, J. and Nyström, K. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

Berg, R. v. d., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Bi, K., Xie, L., Zhang, H., Chen, X., Gu, X., and Tian, Q. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast. *arXiv preprint arXiv:2211.02556*, 2022.

Bivector.net. Bivector.net, 2023. URL https://bivector.net/.

Boelrijk, J., Pirok, B., Ensing, B., and Forré, P. Bayesian optimization of comprehensive two-dimensional liquid chromatography separations. *Journal of Chromatography A*, 1659:462628, 2021.

Bonnet, F., Mazari, J. A., Cinnella, P., and Gallinari, P. AirfRANS: High fidelity computational fluid dynamics dataset for approximating reynolds-averaged navier–stokes solutions. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022. URL https://openreview.net/forum?id=Zp8YmiQ_bDC.

Brandstetter, J., Berg, R. v. d., Welling, M., and Gupta, J. K. Clifford neural layers for PDE modeling. *arXiv preprint arXiv:2209.04934*, 2022a.

Brandstetter, J., Welling, M., and Worrall, D. E. Lie point symmetry data augmentation for neural pde solvers. *arXiv preprint arXiv:2202.07643*, 2022b.

Brandstetter, J., Worrall, D., and Welling, M. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022c.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Buchholz, S. and Sommer, G. On clifford neurons and clifford multi-layer perceptrons. *Neural Networks*, 21(7): 925–935, 2008.

Byravan, A. and Fox, D. Se3-nets: Learning rigid body motion using deep neural networks. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 173–180. IEEE, 2017.

Cao, H., Lu, Y., Lu, C., Pang, B., Liu, G., and Yuille, A. Asap-net: Attention and structure aware point cloud sequence segmentation. *arXiv preprint arXiv:2008.05149*, 2020.

Chakraborty, R., Bouza, J., Manton, J. H., and Vemuri, B. C. Manifoldnet: A deep neural network for manifold-valued data with applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(2):799–810, 2020.

Clifford. Preliminary sketch of biquaternions. *Proceedings of the London Mathematical Society*, s1-4(1):381–395, 1871. doi: https://doi.org/10.1112/plms/s1-4.1.381.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999. PMLR, 2016.

Cranmer, M., Tamayo, D., Rein, H., Battaglia, P., Hadden, S., Armitage, P. J., Ho, S., and Spergel, D. N. A bayesian neural network predicts the dissolution of compact planetary systems. *Proceedings of the National Academy of Sciences*, 118(40):e2026053118, 2021.

De Keninck, S. SIGGRAPH2019 - geometric algebra, 2019. URL https://youtu.be/tX4H_ctggYo.

De Keninck, S. GAME2020 - dual quaternions demystified, 2020. URL https://youtu.be/ichOiuBoBoQ.

De Keninck, S. and Roelfs, M. Normalization, square roots, and the exponential and logarithmic maps in geometric algebras of less than 6d. 2022.

Doran, C., Gull, S. R., Lasenby, A., Lasenby, J., and Fitzgerald, W. *Geometric algebra for physicists*. Cambridge University Press, 2003.

Dorst, L. CGI2020 - pga: Plane-based geometric algebra, 2020. URL https://youtu.be/T7xVTBpHMjA.

Dorst, L. and Mann, S. Geometric algebra: a computational framework for geometrical applications. *IEEE Computer Graphics and Applications*, 22(3):24–31, 2002.

Dorst, L., Fontijne, D., and Mann, S. *Geometric Algebra for Computer Science (Revised Edition): An Object-Oriented Approach to Geometry*. Morgan Kaufmann, 2009.

Feng, Y., Feng, Y., You, H., Zhao, X., and Gao, Y. Meshnet: Mesh neural network for 3d shape representation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pp. 8279–8286, 2019.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Finzi, M., Welling, M., and Wilson, A. G. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In *International Conference on Machine Learning*, pp. 3318–3328. PMLR, 2021.

Gao, H., Sun, L., and Wang, J.-X. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428: 110079, 2021.

Geiger, M. and Smidt, T. e3nn: Euclidean neural networks. *arXiv preprint arXiv:2207.09453*, 2022.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Greydanus, S., Dzamba, M., and Yosinski, J. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

Gupta, J. K. and Brandstetter, J. Towards multi-spatiotemporal-scale generalized PDE modeling. *arXiv preprint arXiv:2209.15616*, 2022.

Hadfield, H., Wieser, E., Arsenovic, A., Kern, R., and The Pygae Team. pygae/clifford, 2022. URL https://doi.org/10.5281/zenodo.1453978.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hestenes, D. *Spacetime Algebra*. Birkhäuser Cham, 1966.

Hestenes, D. *New foundations for classical mechanics*. Springer, 1999.

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

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Hoffmann, J., Schmitt, S., Osindero, S., Simonyan, K., and Elsen, E. Algebranets. *arXiv preprint arXiv:2006.07360*, 2020.

Holl, P., Koltun, V., and Thuerey, N. Learning to control PDEs with differentiable physics. *arXiv preprint arXiv:2001.07457*, 2020a.

Holl, P., Koltun, V., Um, K., and Thuerey, N. phiflow: A differentiable pde solving framework for deep learning via physical simulations. In *NeurIPS Workshop*, volume 2, 2020b.

Hoogeboom, E., Garcia Satorras, V., Tomczak, J., and Welling, M. The convolution exponential and generalized sylvester flows. *Advances in Neural Information Processing Systems*, 33:18249–18260, 2020.

Hoogeboom, E., Satorras, V. G., Vignac, C., and Welling, M. Equivariant diffusion for molecule generation in 3d. In *International Conference on Machine Learning*, pp. 8867–8887. PMLR, 2022.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Kahlow, R. JAX geometric algebra, 2023a. URL https://github.com/RobinKa/jaxga.

Kahlow, R. TensorFlow geometric algebra, 2023b. URL https://doi.org/10.5281/zenodo.3902404.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Klöwer, M., Kimpson, T., White, A., and Giordano, M. milankl/speedyweather.jl: v0.2.1, July 2022. URL https://doi.org/10.5281/zenodo.6788067.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Kofinas, M., Nagaraja, N., and Gavves, E. Roto-translated local coordinate frames for interacting dynamical systems. *Advances in Neural Information Processing Systems*, 34: 6417–6429, 2021.

Koval, I., Schiratti, J.-B., Routier, A., Bacci, M., Colliot, O., Allassonnière, S., Durrleman, S., and Initiative, A. D. N. Statistical learning of spatiotemporal patterns from longitudinal manifold-valued networks. In *Medical Image Computing and Computer Assisted Intervention- MICCAI*

2017: 20th International Conference, Quebec City, QC, Canada, September 11-13, 2017, Proceedings, Part I 20, pp. 451–459. Springer, 2017.

Kucharski, F., Molteni, F., King, M. P., Farneti, R., Kang, I.-S., and Feudale, L. On the need of intermediate complexity general circulation models: A "SPEEDY" example. *Bulletin of the American Meteorological Society*, 94(1):25–30, January 2013. doi: 10.1175/bams-d-11-00238.1. URL https://doi.org/10.1175/bams-d-11-00238.1.

Kuroe, Y. Models of Clifford recurrent neural networks and their dynamics. In *International Joint Conference on Neural Networks*, pp. 1035–1041. IEEE, 2011.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Pritzel, A., Ravuri, S., Ewalds, T., Alet, F., Eaton-Rosen, Z., et al. GraphCast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*, 2022.

Li, S.-H., Dong, C.-X., Zhang, L., and Wang, L. Neural canonical transformation with symplectic flows. *Physical Review X*, 10(2):021020, 2020a.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020b.

Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022.

Lippert, F., Kranstauber, B., Forré, P. D., and van Loon, E. E. Learning to predict spatiotemporal movement dynamics from weather radar networks. *Methods in Ecology and Evolution*, 2022.

Liu, X., Qi, C. R., and Guibas, L. J. Flownet3d: Learning scene flow in 3d point clouds. In *IEEE/CVF conference on computer vision and pattern recognition*, pp. 529–537, 2019a.

Liu, X., Yan, M., and Bohg, J. Meteornet: Deep learning on dynamic 3d point cloud sequences. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9246–9255, 2019b.

Lopes, W. ICACGA2022 - clifford convolutional networks for lymphoblast image classification, 2022. URL https://youtu.be/DR45pK-t8dk.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Macdonald, A. *Linear and Geometric Algebra*. CreateSpace Independent Publishing Platform (Lexington), 2012.

Mann, S. and Dorst, L. Geometric algebra: A computational framework for geometrical applications. 2. *IEEE Computer Graphics and Applications*, 22(4):58–67, 2002.

Mardt, A., Pasquali, L., Wu, H., and Noé, F. Vampnets for deep learning of molecular kinetics. *Nature communications*, 9(1):1–11, 2018.

Mattheakis, M., Protopapas, P., Sondak, D., Di Giovanni, M., and Kaxiras, E. Physical symmetries embedded in neural networks. *arXiv preprint arXiv:1904.08991*, 2019.

Mayr, A., Lehner, S., Mayrhofer, A., Kloss, C., Hochreiter, S., and Brandstetter, J. Boundary graph neural networks for 3d simulations. *arXiv preprint arXiv:2106.11299*, 2021.

Mhammedi, Z., Hellicar, A., Rahman, A., and Bailey, J. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *International Conference on Machine Learning*, pp. 2401–2409. PMLR, 2017.

Milano, F., Loquercio, A., Rosinol, A., Scaramuzza, D., and Carlone, L. Primal-dual mesh convolutional neural networks. *Advances in Neural Information Processing Systems*, 33:952–963, 2020.

Miller, B. K., Geiger, M., Smidt, T. E., and Noé, F. Relevance of rotationally equivariant convolutions for predicting molecular properties. *arXiv preprint arXiv:2008.08461*, 2020.

Miller, B. K., Cole, A., Forré, P., Louppe, G., and Weniger, C. Truncated marginal neural ratio estimation. *Advances in Neural Information Processing Systems*, 34:129–143, 2021.

Molteni, F. Atmospheric simulations using a gcm with simplified physical parametrizations. i: Model climatology and variability in multi-decadal experiments. *Climate Dynamics*, 20(2):175–191, 2003.

Nguyen, T., Brandstetter, J., Kapoor, A., Gupta, J. K., and Grover, A. Climax: A foundation model for weather and climate. *arXiv preprint arXiv:2301.10343*, 2023.

Nozick, V. GAME2020 - geometric neurons, 2020. URL https://youtu.be/KC3c_Mdj1dk.

Pandeva, T. and Forré, P. Multi-view independent component analysis with shared and individual sources. *arXiv preprint arXiv:2210.02083*, 2022.

Parcollet, T., Morchid, M., and Linarès, G. A survey of quaternion neural networks. *Artificial Intelligence Review*, 53(4):2957–2982, 2020.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

Pearson, J. Clifford networks. In *Complex-Valued Neural Networks: Theories and Applications*, pp. 81–106. World Scientific, 2003.

Pearson, J. and Bisset, D. Neural networks in the clifford domain. In *IEEE International Conference on Neural Networks (ICNN)*, volume 3, pp. 1465–1469. IEEE, 1994.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Ramachandran, P., Zoph, B., and Le, Q. V. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Rasp, S. and Thuerey, N. Data-driven medium-range weather prediction with a resnet pretrained on climate simulations: A new model for weatherbench. *Journal of Advances in Modeling Earth Systems*, 13(2): e2020MS002405, 2021.

Roelfs, M. and De Keninck, S. Graded symmetry groups: Plane and simple. *arXiv preprint arXiv:2107.03771*, 2021.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Ruhe, D. and Forré, P. Self-supervised inference in state-space models. *arXiv preprint arXiv:2107.13349*, 2021.

Ruhe, D., Kuiack, M., Rowlinson, A., Wijers, R., and Forré, P. Detecting dispersed radio transients in real time using convolutional neural networks. *Astronomy and Computing*, 38:100512, 2022a.

Ruhe, D., Wong, K., Cranmer, M., and Forré, P. Normalizing flows for hierarchical bayesian analysis: A gravitational wave population study. *arXiv preprint arXiv:2211.09008*, 2022b.

Ruhe, D., Brandstetter, J., and Forré, P. Clifford group equivariant neural networks. *arXiv preprint arXiv:2305.11141*, 2023.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E., Ghasemipour, S. K. S., Ayan, B. K., Mahdavi, S. S., Lopes, R. G., et al. Photorealistic text-to-image diffusion models with deep language understanding. *arXiv preprint arXiv:2205.11487*, 2022.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Satorras, V. G., Hoogeboom, E., and Welling, M. E(n) equivariant graph neural networks. In *International Conference on Machine Learning (ICML)*, pp. 9323–9332. PMLR, 2021.

Smidt, T. E. Euclidean symmetry and equivariance in machine learning. *Trends in Chemistry*, 3(2):82–85, 2021.

Smidt, T. E., Geiger, M., and Miller, B. K. Finding symmetry breaking order parameters with euclidean neural networks. *Physical Review Research*, 3(1):L012002, 2021.

Spellings, M. Geometric algebra attention networks for small point clouds. *arXiv preprint arXiv:2110.02393*, 2021.

sugylacmoe. sudgylacmoe, 2023. URL https://www.youtube.com/@sudgylacmoe.

Tamayo, D., Silburt, A., Valencia, D., Menou, K., Ali-Dib, M., Petrovich, C., Huang, C. X., Rein, H., Van Laerhoven, C., Paradise, A., et al. A machine learns to predict the stability of tightly packed planetary systems. *The Astrophysical Journal Letters*, 832(2):L22, 2016.

Temam, R. *Navier-Stokes equations: theory and numerical analysis*, volume 343. American Mathematical Soc., 2001.

Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

Tillet, P., Kung, H.-T., and Cox, D. Triton: an intermediate language and compiler for tiled neural network computations. In *ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Trabelsi, C., Bilaniuk, O., Serdyuk, D., Subramanian, S., Santos, J. F. S., Mehri, S., Rostamzadeh, N., Bengio, Y., and Pal, C. J. Deep complex networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Trindade, M. A., Rocha, V. N., and Floquet, S. Clifford algebras, quantum neural networks and generalized quantum Fourier transform. *arXiv preprint arXiv:2206.01808*, 2022.

Unke, O., Bogojeski, M., Gastegger, M., Geiger, M., Smidt, T., and Müller, K.-R. Se (3)-equivariant prediction of molecular wavefunctions and electronic densities. *Advances in Neural Information Processing Systems*, 34: 14434–14447, 2021.

Vallejo, J. R. and Bayro-Corrochano, E. Clifford hopfield neural networks. In *IEEE International Joint Conference on Neural Networks (IEEE world congress on computational intelligence)*, pp. 3609–3612. IEEE, 2008.

Vreugdenhil, C. B. *Numerical methods for shallow-water flow*, volume 13. Springer Science & Business Media, 1994.

Wang, J., Chen, Y., Chakraborty, R., and Yu, S. X. Orthogonal convolutional neural networks. In *IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 11505–11515, 2020.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph CNN for learning on point clouds. *ACM Transactions On Graphics (tog)*, 38 (5):1–12, 2019.

Wei, Y., Liu, H., Xie, T., Ke, Q., and Guo, Y. Spatial-temporal transformer for 3d point cloud sequences. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1171–1180, 2022.

Weiler, M., Geiger, M., Welling, M., Boomsma, W., and Cohen, T. S. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *Advances in Neural Information Processing Systems*, 31, 2018.

Weiler, M., Forré, P., Verlinde, E., and Welling, M. Coordinate independent convolutional networks–isometry and gauge equivariant convolutions on riemannian manifolds. *arXiv preprint arXiv:2106.06020*, 2021.

Wu, Y. and He, K. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Xu, B., Wang, N., Chen, T., and Li, M. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

Zang, D., Chen, X., Lei, J., Wang, Z., Zhang, J., Cheng, J., and Tang, K. A multi-channel geometric algebra residual network for traffic data prediction. *IET Intelligent Transport Systems*, 2022.

Zhong, Y. D., Dey, B., and Chakraborty, A. Symplectic ode-net: Learning hamiltonian dynamics with control. *arXiv preprint arXiv:1909.12077*, 2019.

A. Glossary

In Table 2, we provide an overview of notations that are commonly used throughout the paper.

Notation	Meaning
$\mathcal{G}$	Group
$\mathbb{G}_{p,q,r}$	A geometric algebra with p positive dimensions, q negative dimensions and r zero dimensions. q and r are left out of the notation when they equal 0.
$\mathrm{Pin}(p,q,r)$	Pin group with p positive dimensions, q negative dimensions and r zero dimensions.
u, v, w	Abstract $\mathrm{Pin}(p,q,r)$ group elements.
$u_1 u_2 \ldots u_k$	A k-reflection (composition of k reflections) Abstract element of $\mathrm{Pin}(p,q,r)$.
uvu^{-1}	Group conjugation of the form $G \times G \to G$. The group acts on itself.
$\mathbf{x}$	A multivector of $\mathbb{G}_{p,q,r}$.
$[\mathbf{x}]_k$	Selects the k-vector part of $\mathbf{x}$. For example, $[\mathbf{x}]_0$ selects the scalar part, $[\mathbf{x}]_1$ the vector part and $[\mathbf{x}]_2$ the bivector part.
$\mathbf{xy}$	Geometric product between $\mathbf{x}$ and $\mathbf{y}$.
$\lambda\mathbf{x}$	Scalar product of scalar λ with multivector $\mathbf{x}$.
$\mathbf{u}, \mathbf{v}$	$\mathrm{Pin}(p,q,r)$ elements expressed in $\mathbb{G}_{3,0,1}$.
$\mathbf{u}_1, \mathbf{u}_2, \ldots \mathbf{u}_k$	A k-reflection (composition of k reflections using geometric products) expressed in $\mathbb{G}_{3,0,1}$.
$(-1)^{kl} \cdot \mathbf{uvu}^{-1}$	$\mathrm{Pin}(p,q,r)$ group action expressed in elements of $\mathbb{G}_{3,0,1}$.
$\mathbf{u}^{-1}$	Multiplicative inverse (using the geometric product) of $\mathbf{u}$. That is, $\mathbf{uu}^{-1} = 1$.
$\mathbf{a}_i$	$\mathrm{Pin}(p,q,r)$ group element used as group action in our geometric algebra neural layers.
e_i	A basis vector of $\mathbb{G}_{p,q,r}$.
$e_i e_j$	A basis bivector of $\mathbb{G}_{p,q,r}$.
$e_i e_j e_k$	A basis trivector of $\mathbb{G}_{p,q,r}$.
e_0	Fourth (special) basis vector of a geometric algebra modeling three-dimensional space. $e_0^2 = \{-1, 0, 1\}$ leads to hyperbolic, projective and Euclidean geometry, respectively.
I	The pseudoscalar of a geometric algebra.
x, y, z	Axes of a (Euclidean) coordinate system.
$\mathrm{Spin}(n)$	Special $\mathrm{Pin}(n)$ group, excluding improper isometries.
$\mathrm{O}(n),$	n-dimensional orthogonal group.
$\mathrm{SO}(n)$	n-dimensional special orthogonal group.
$\mathrm{E}(n)$	n-dimensional Euclidean group.
$\mathrm{SE}(n)$	n-dimensional special Euclidean group.
X	Vector space.
$\alpha(\cdot, \cdot)$	Group action of our group action linear layers.
g_i	Group element.
x_i	Vector space element.
w_i	Scalar neural network weight.

Table 2. Overview of notations commonly used in the paper.

B. Geometric Algebra

Technically, there are no differences between geometric and (real) Clifford algebra [5]. However, it is common practice to use Clifford algebra when interested in mathematical concerns (e.g., beyond using real numbers), and geometric algebra when interested in geometry. A Clifford algebra is constructed by equipping a vector space with a quadratic form (see Appendix F). The number of positive (p), negative (q), or zero (r) eigenvalues (usually representing dimensions) of the *metric* of the quadratic form determines the signature of the algebra. Specifically, for an n-dimensional real vector space $\mathbb{R}^n$ ($n = p + q + r$) we can choose a basis with

$$
\begin{aligned}
e_i^2 &= +1 & 1 \leq i \leq p, \\
e_i^2 &= -1 & p < i \leq p+q, \\
e_i^2 &= 0 & p+q < i \leq n, \\
e_i e_j &= -e_j e_i & i \neq j.
\end{aligned}
\tag{16}
$$

These identies originate from the fact that the geometric product of two vectors yields a quadratic form and an anti-symmetric wedge product (Appendix F). For two parallel vectors, the wedge product is zero, meaning we only get the scalar quadratic form. For two orthogonal vectors, the scalar part is zero and we only obtain the anti-symmetric part. The metric is usually diagonal with elements in $\{+1, -1, 0\}$, and we can similarly use unit basis vectors to produce all the identities of Equation (16).

After picking a basis for the underlying vector space $\mathbb{R}^n$, multiplying its components yields higher-order basis elements, called *basis blades*. Through this construction, the *algebraic* basis of the algebraic vector space has 2^n elements. For example, when $n = 3$, the space is spanned by $\{1, e_1, e_2, e_3, e_{12}, e_{13}, e_{23}, e_{123}\}$, where e_{ij} is shorthand for $e_i e_j$. Note that this set is closed under multiplication with elements from itself using the relations Equation (16). The *grade*[6] of a blade is the dimensionality of the subspace it represents. For example, the grades of $\{1, e_1, e_{12}, e_{123}\}$ are 0 (scalar), 1 (vector), 2 (bivector) and 3 (trivector), respectively. The highest grade basis blade $I := e_1 \ldots e_n$ is also known as the *pseudoscalar*. A vector is written as $x_1 e_1 + x_2 e_2 + x_3 e_3$, a bivector as $x_{12} e_{12} + x_{13} e_{13} + x_{23} e_{23}$, and so on. Similarly to how vectors can be interpreted as oriented line segments, bivectors can be interpreted as oriented plane segments and trivectors as oriented cube segments. We can construct k-vectors ($k \leq n$) by homogeneously combining basis blades of grade k. A *multivector* $\mathbf{x} \in \mathbb{G}_{p,q,r}$ is a sum of k-vectors, such that $\mathbf{x} = [\mathbf{x}]_0 + [\mathbf{x}]_1 + \cdots + [\mathbf{x}]_n$, where $[\mathbf{x}]_k$ denotes the k-vector part of $\mathbf{x}$. Combining k-vectors leads to the most general element of the algebra: a *multivector*. In a three-dimensional algebra, this takes the form

$$
\mathbf{x} = \underbrace{x_0 1}_{\text{Scalar}} + \underbrace{x_1 e_1 + x_2 e_2 + x_3 e_3}_{\text{Vector}} + \underbrace{x_{12} e_{12} + x_{13} e_{13} + x_{23} e_{23}}_{\text{Bivector}} + \underbrace{x_{123} e_{123}}_{\text{Trivector}}.
\tag{17}
$$

We usually write $\mathbb{G}_{p,q,r}$ (or $Cl_{p,q,r}$) to indicate what algebra we are using, where we suppress from $\mathbb{G}(\mathbb{R}_{p,q,r})$ the underlying generating vector space argument. Sometimes, the q and r components are left out when equal to zero. The specific choice of algebra (determined by p, q and r) allows for efficient modeling many types of geometry.

Clifford multiplication: the geometric product. Multiplying two elements of the algebra yields the geometric product. It is associative,

$$
(\mathbf{x}\mathbf{y})\mathbf{z} = \mathbf{x}(\mathbf{y}\mathbf{z}),
\tag{18}
$$

left and right distributive,

$$
\mathbf{x}(\mathbf{y} + \mathbf{z}) = \mathbf{x}\mathbf{y} + \mathbf{x}\mathbf{z}
\tag{19}
$$

$$
(\mathbf{x} + \mathbf{y})\mathbf{z} = \mathbf{x}\mathbf{z} + \mathbf{y}\mathbf{z},
\tag{20}
$$

closed under multiplication,

$$
\mathbf{x}\mathbf{y} \in \mathbb{G}_{p,q,r},
\tag{21}
$$

---

[5]In fact, Clifford himself chose "geometric algebra".
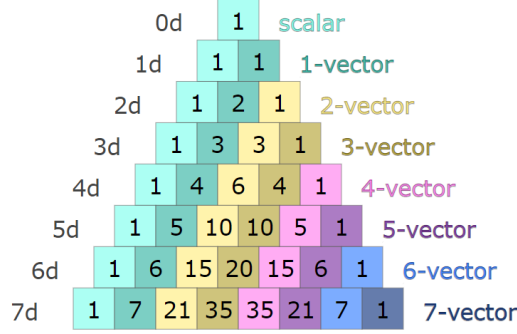[6]Technically, a Clifford algebra is not a graded algebra.

Figure 8. This figure shows the symmetric duality structure of geometric algebras. Up to seven dimensions, we show the number of k-blades up to the pseudoscalar (n-vector) for $n = 0 \ldots 7$. For each grade, we see that we have an equal number $\binom{n}{k} = \binom{n}{n-k}$ of dual blades.

communicative with scalars,

$$\lambda \mathbf{x} = \mathbf{x}\lambda \,. \tag{22}$$

Using the associativity and distributivity laws of the algebra, we get for two multivectors with 1-vector components only, i.e., commonly known "vectors", $\mathbf{u} = u_1 e_1 + u_2 e_2$ and $\mathbf{v} = v_1 e_1 + v_2 e_2$,

$$\mathbf{uv} = (u_1 e_1 + u_2 e_2)(v_1 e_1 + v_2 e_2) \tag{23}$$
$$= \langle \mathbf{u}, \mathbf{v} \rangle + u_1 v_2 e_{12} + u_2 v_1 e_{21} \tag{24}$$
$$= \langle \mathbf{u}, \mathbf{v} \rangle + (u_1 v_2 - u_2 v_1) e_{12} \tag{25}$$
$$= \langle \mathbf{u}, \mathbf{v} \rangle + \mathbf{u} \wedge \mathbf{v} \,, \tag{26}$$

where $\langle \cdot, \cdot \rangle$ and $\cdot \wedge \cdot$ are the quadratic form and wedge product by construction (Appendix F).

Let $g_{ij} := \langle e_i, e_j \rangle$. In general, we can compute the geometric product using its associativity and distributivity laws for two multivectors like

$$\begin{aligned}
\mathbf{xy} = &\left(x_0 y_0 + g_{11} x_1 y_1 + g_{22} x_2 y_2 - g_{11} g_{22} x_{12} y_{12} \right) 1 \\
&+ \left(x_0 y_1 + x_1 y_0 - g_{22} x_2 y_{12} + g_{22} x_{12} y_2 \right) e_1 \\
&+ \left(x_0 y_2 + g_{11} x_1 y_{12} + x_2 y_0 - g_{11} x_{12} y_1 \right) e_2 \\
&+ \left(x_0 y_{12} + x_1 y_2 - x_2 y_1 + x_{12} y_0 \right) e_{12} \,,
\end{aligned} \tag{27}$$

where now $\mathbf{x} = x_0 1 + x_1 e_1 + x_2 e_2 + x_{12} e_{12}$ and $\mathbf{y} = y_0 1 + y_1 e_1 + y_2 e_2 + y_{12} e_{12}$. This is the primary operation of Brandstetter et al. (2022a).

Duality. We can divide a geometric algebra into the vector subspaces that are spanned by each k-vector. As such, we get

$$\mathbb{G}_n = \mathbb{G}_n^0 \oplus \mathbb{G}_n^1 \oplus \cdots \oplus \mathbb{G}_n^n \,. \tag{28}$$

The dimensionality (number of basis blades) of $\mathbb{G}_n^k$ is given by $\binom{n}{k}$. Note that $\binom{n}{k} = \binom{n-k}{k}$. This symmetry shows the duality of the algebra (depicted in Figure 8). Multiplying a multivector $\mathbf{x}$ with the pseudoscalar I yields its dual $\mathbf{x}I$. That is, scalars map to pseudoscalars, vectors to $(n-1)$-vectors, and so forth.

Grade reversion and normalization. Let $\mathbf{u} \in \mathbb{G}_{p,q,r}$ be a k-reflection, i.e., $\mathbf{u} := \mathbf{u}_1 \ldots \mathbf{u}_k$ where $\mathbf{u}_i$ are reflections (implemented as 1-vectors). We define the *grade reversion* operator as

$$\mathbf{u}^\dagger := \mathbf{u}_k \ldots \mathbf{u}_1 \,, \tag{29}$$

which is an involution $\mathbb{G}_{p,q,r} \to \mathbb{G}_{p,q,r}$. In practice, this can be efficiently computed by noting that

$$[\mathbf{u}^\dagger]_k = (-1)^{k(k-1)/2} [\mathbf{u}]_k \,, \tag{30}$$

where $[\mathbf{u}]_k$ selects the k-vector part of $\mathbf{u}$. For example, $[\mathbf{u}]_2$ selects the bivector components of $\mathbf{u}$. Effectively, this simply flips the sign of the components that have $k = 2 \mod 4$ or $k = 3 \mod 4$. In geometric algebras of dimension $n \leq 3$, $\mathbf{u}^\dagger \mathbf{u}$ is a scalar quantity. We can therefore define a norm

$$\|\mathbf{u}\| := \sqrt{|\mathbf{u}^\dagger \mathbf{u}|}. \tag{31}$$

For normalized $\mathbf{u}$, we have

$$\mathbf{u}^\dagger \mathbf{u} = \mathbf{u}^{-1} \mathbf{u} = 1. \tag{32}$$

For example, using $\mathbb{G}_{2,0,0}$ we have for $\mathbf{u} = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} e_{12}$, $\mathbf{u}^\dagger = \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} e_{12}$, $g_{11} = 1$, and $g_{22} = 1$ we get

$$\mathbf{u}\mathbf{u}^\dagger = \left(\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} e_{12} \right) \left(\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} e_{12} \right) \tag{33}$$

$$= \frac{1}{2} - \frac{1}{2} e_{12} + \frac{1}{2} e_{12} - \frac{1}{2} e_{12} e_{12} \tag{34}$$

$$= \frac{1}{2} + \frac{1}{2} e_1^2 e_2^2 \tag{35}$$

$$= 1. \tag{36}$$

Reflections. Using vector algebra, a reflection of $v \in \mathbb{R}^n$ in the hyperplane with normal u is given by

$$v \mapsto v - 2 \frac{\langle u, v \rangle}{\langle u, u \rangle} u, \tag{37}$$

which, intuitively, subtracts from v the projection of v onto u twice. Using the geometric product and embedding the vectors as 1-vectors, we can rewrite this as

$$\mathbf{v} - 2 \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\langle \mathbf{u}, \mathbf{u} \rangle} \mathbf{u} = \mathbf{v} - (\mathbf{v}\mathbf{u} + \mathbf{u}\mathbf{v})\mathbf{u}^{-1} \tag{38}$$

$$= -\mathbf{u}\mathbf{v}\mathbf{u}^{-1}, \tag{39}$$

where we used $\mathbf{u}^{-1} := \frac{\mathbf{u}}{\langle \mathbf{u}, \mathbf{u} \rangle}$ since

$$\mathbf{u}\mathbf{u}^{-1} = \frac{\mathbf{u}^2}{\langle \mathbf{u}, \mathbf{u} \rangle} = 1, \tag{40}$$

and we used the fundamental Clifford identity $\langle \mathbf{u}, \mathbf{v} \rangle = \frac{1}{2}(\mathbf{u}\mathbf{v} + \mathbf{v}\mathbf{u})$ (for two vectors, Appendix F).

The Cartan-Dieudonné theorem tells us that all higher-order orthogonal transformations of an n-dimensional space can be constructed from at most n reflections. As such, we can apply

$$\mathbf{v} \mapsto (-1)^k \mathbf{u}_1 \ldots \mathbf{u}_k \, \mathbf{v} \, (\mathbf{u}_1 \ldots \mathbf{u}_k)^{-1} = \mathbf{u}_1 \ldots \mathbf{u}_k \, \mathbf{v} \, \mathbf{u}_k^{-1} \ldots \mathbf{u}_1^{-1} \tag{41}$$

to compute a k-reflection. We used the fact that for normalized vectors $\mathbf{u}_i$, $(\mathbf{u}_1 \ldots \mathbf{u}_k)^{-1} = (\mathbf{u}_1 \ldots \mathbf{u}_k)^\dagger$, which can simply be computed using Equation (30).

Outermorphism. A linear map $F : \mathbb{G}_n \to \mathbb{G}_n$ is called an *outermorphism* if

1. $F(1) = 1$

2. $F(\mathbb{G}_n^m) \subseteq \mathbb{G}_n^m$

3. $F(\mathbf{x}\mathbf{y}) = F(\mathbf{x})F(\mathbf{y})$.

Property 2 means that such a map is *grade-preserving*. Further,

Theorem B.1. *For every linear map $f : \mathbb{R}^n \to \mathbb{R}^n$ there exists a unique outermorphism $F : \mathbb{G}_n \to \mathbb{G}_n$ such that for $x \in \mathbb{R}^n, F(x) = f(x)$.*

It can be shown (e.g., Macdonald (2012); Hestenes & Sobczyk (2012)) that reflections implemented using geometric products directly extend to outermorphisms. As such, by induction, we get that their compositions are outermorphisms, hence Theorem 2.1 applies and we get grade-preserving isometries operating independently on the geometric algebra subspaces.

Even subalgebras. We can construct *even subalgebras* by considering only the basis blades that have $k = 0 \mod 2$. In certain cases, we can use subalgebras to model higher-dimensional transformations. For example, $\mathbb{G}_{0,1}$ is isomorphic to the complex numbers $\mathbb{C}$, but so are the even grades of $\mathbb{G}_2$. We let the scalar part of $\mathbb{G}_2$ identify with the real part of a complex number, and the bivector (pseudoscalar) part with the imaginary part. The bivector also squares to -1: $e_{12}e_{12} = -e_1^2 e_2^2 = -1$. Similarly, $\mathbb{G}_{0,2}$ is isomorphic to the quaternions $\mathbb{H}$, which are often used to model three-dimensional rotations, but so are the even grade elements of $\mathbb{G}_3$, i.e., the scalar and the three bivectors. This can easily be seen, since bivectors square to -1 in $\mathbb{G}_3$, and so do vectors in $\mathbb{G}_{0,2}$. We usually say that bivectors parameterize rotations.

Advantages of geometric algebra over vector algebra. We discuss some advantages of geometric algebra over classical (vector) linear algebra. First and foremost, geometric algebra is an extension (completion) of vector algebra. For example, cross-products, norms, determinants, matrix multiplications, geometric primitives, projections, areas, and derivatives can all be computed in geometric algebra in a (sometimes more) interpretable way. Second, in vector algebra, we need several different approaches to represent basic geometric objects. When done naively, this can lead to ambiguities. For example, one uses vectors for both directions and points. In geometric algebra, one can naturally represent these objects as invariant subspaces. Third, in vector algebra, we also need to implement transformations differently depending on the object. In contrast, in geometric algebra, we parameterize transformations of space, and the objects transform covariantly, regardless of what they are. Fourth, matrices are dense and hard to interpret. That is, it is not straightforward to quickly see whether a certain matrix is, e.g., a rotation or reflection. In geometric algebra, by just observing what components are parameterized, we can directly see what transformation a certain multivector parameterizes. Finally, geometric algebra generalizes across dimensions. That is, a computational geometry computer script that works for, e.g., two dimensions also works for three and higher dimensions. In other words, we do not need to call different functions depending on the dimensionality of the problem. This is not usually the case for classical approaches.

C. References

In this section, we first discuss related scientific work and then provide further references to other matters involving (incorporating machine learning in) geometric algebra.

C.1. Related Work

We discuss work related to incorporating geometric priors in neural networks for dynamical systems, using orthogonal transformations or isometries in neural networks, Clifford or geometric algebras in deep learning.

Geometric priors in dynamical systems. The use of machine learning, especially deep learning, has proven to be highly effective in tackling complex scientific problems (Li et al., 2020b; Lam et al., 2022; Jumper et al., 2021; Sanchez-Gonzalez et al., 2020; Mayr et al., 2021; Thomas et al., 2018; Miller et al., 2021; Lippert et al., 2022; Boelrijk et al., 2021; Pandeva & Forré, 2022; Hoogeboom et al., 2022; Brandstetter et al., 2022c;b; Ruhe et al., 2022a;b; Smidt, 2021; Smidt et al., 2021; Batzner et al., 2022). Many of these settings involve the dynamical evolution of a system. These systems play out in spaces equipped with geometries, making geometry an essential aspect of modeling. Geometric deep learning focusing on equivariance (Bronstein et al., 2017; 2021; Cohen & Welling, 2016; Weiler et al., 2018; Finzi et al., 2021; Geiger & Smidt, 2022) or learning on manifolds (Feng et al., 2019; Chakraborty et al., 2020; Milano et al., 2020; Koval et al., 2017) forms a rich subfield enabling in which models can be parameterized such that they are either invariant of equivariant to group actions applied to the input features. Methods arising from this philosophy have successfully been applied to scientific settings (Batatia et al., 2022; Miller et al., 2020; Unke et al., 2021; Satorras et al., 2021). For example, the celebrated AlphaFold protein folding architecture (Jumper et al., 2021) uses an E(3)-equivariant attention mechanism. There has been less focus on incorporating geometric priors in a similar sense to the current work: biasing a model towards (Euclidean) rigid motions, often found in dynamical systems. Examples of inductive biases in dynamical systems are given by Ruhe & Forré (2021), who bias a particle's trajectory to transformations given by underlying prior physics knowledge of the system,

and Kofinas et al. (2021), who provide local coordinate frames per particle in Euclidean space to induce roto-translation invariance to the geometric graph of the dynamical system.

Regarding geometric priors in dynamical systems, we discuss works related to point clouds and neural partial differential equation surrogates. Current leading work is provided by Liu et al. (2019b), who build on Qi et al. (2017). The work constructs spatiotemporal neighborhoods for each point in a point cloud. Local features are thereby grouped and aggregated to capture the dynamics of the whole sequence. Cao et al. (2020) similarly propose a spatiotemporal grouping called Spatio-Temporal Correlation. Further, Byravan & Fox (2017); Behl et al. (2019) are seminal works for modeling point cloud sequences. There are several critical differences between the current work an the above-mentioned. First, our approach is general in the sense that we can apply actions from several groups (e.g., O(n), SO(n) for all n, E(3), SE(3)), depending on the application. Second, we can naturally operate on objects of different vector spaces: (standard) vectors, points, lines, planes, and so on. And, most notably, we can operate on *grouped* versions of these, transforming them in a coupled manner. Third, we provide network layers that preserves the vector space of these objects. Fourth, geometric algebra provides a flexible framework that can be extended to different kinds of geometry, such as *conformal* geometry. Finally, some of these do not explicitly provide a geometric prior, or their application setting is quite different (e.g., estimating Euclidean motion from images). Works like Liu et al. (2019a) and its extension (that includes a geometric prior in the form of a "geometric loss function") operate on voxalized grids, whereas we operate in this work directly on the objects. Wei et al. (2022) introduce a spatiotemporal transformer that can enhance the resolution of the sequences.

Relatively less work has been published on incorporating geometry in neural PDE surrogates. Berg & Nyström (2018) propose to include partial derivatives with respect to space variables in the backpropagation algorithm. Gao et al. (2021) propose `PhyGeoNet`, a convolutional network architecture supported by coordinate transformations between a regular domain and the irregular physical domain. Li et al. (2022) similarly uses a coordinate transformation from irregular domains to a regular latent domain, which is now fully learned. All of the above methods differ from the current approach by not incorporating geometric templates, the main contribution of the current work. Brandstetter et al. (2022a) propose a weaker version of geometric templates, which was discussed in detail throughout the current work.

Orthogonality in neural networks. Next, we discuss more generally where orthogonal transformations or isometries have been used in deep learning. For time series, Mhammedi et al. (2017); Wang et al. (2020) propose orthogonal recurrent neural networks through Householder reflections and Cayley transforms, respectively. The main contribution of orthogonality is a solution to exploding or vanishing gradients, and not necessarily incorporating geometry. Wang et al. (2020) propose orthogonal filters for convolutional neural networks, also to enable training stability.

Further, orthogonality enables easy invertibility and cheap Jacobian determinants for normalizing flows (Berg et al., 2018; Hoogeboom et al., 2020).

Clifford and geometric algebras in neural networks. Incorporating Clifford algebras in neural networks is a rich subfield that started already with Pearson & Bisset (1994); Pearson (2003). Most of such approaches incorporate "Clifford neurons" (Vallejo & Bayro-Corrochano, 2008; Buchholz & Sommer, 2008) as an extension of real-valued neural networks. For an overview of many of these works, consider Brandstetter et al. (2022a). More recent applications of Clifford algebras in neural networks are Kuroe (2011); Zang et al. (2022); Trindade et al. (2022). Most works that incorporate geometric algebra in neural networks are actually (in this sense) Clifford neural networks (Bayro-Corrochano & Buchholz, 1997; Bayro-Corrochano, 2001). In the current work, we are more rigorous in building geometry into the models. In particular, Bayro-Corrochano (2001) (to our knowledge) were the first to propose taking linear combinations of geometric products with learnable scalar weights. This contrasts similarly to previously mentioned works in the sense that the multivector grades completely get mixed. Finally, Spellings (2021) also use geometric algebra in a more principled way but use it to construct equivariant neural networks that can operate on geometric primitives other than scalars.

C.2. Resources

Excellent introductory books to geometric algebra for computer science or physics include Hestenes (1999); Doran et al. (2003); Dorst et al. (2009); Macdonald (2012).

De Keninck (2019; 2020); Dorst (2020) provide outstanding introductory lectures to plane-based geometry and (projective and conformal) geometric algebra. Further, Nozick (2020) give a talk about geometric neurons, and Lopes (2022) discuss Clifford CNNs for lymphoblast image classification.

To get started with Clifford or geometric algebra in Python, Hadfield et al. (2022) is the go-to package. Further, Kahlow

(2023b) provides an implementation of many of these concepts in `TensorFlow`. Kahlow (2023a) is its followup, where the same procedures are implemented in `JAX`.

Other great references include Bivector.net (2023) and sugylacmoe (2023).

D. Implementation Details

In this section we discuss how we practically implemented our models and some practicalities like weight initialization. To be complete, we repeat the definition of the group action linear layer using geometric algebra:

$$T_{g,w}(\mathbf{x}) = \sum_{i=1}^{c} w_i \cdot \mathbf{a}_i \mathbf{x}_i \mathbf{a}_i^{-1}, \tag{42}$$

with now $\mathbf{x}_i \in X := \mathbb{G}_{p,q,r}$, and $\mathbf{a}_i \in \mathcal{G} := \mathrm{Pin}(p,q,r)$.

D.1. Initialization

As shown in Equation (10), group action layers generalize scalar neural layers. Therefore, we can get similar variance-preserving properties by using initialization schemes for $w = (w_1, \ldots, w_c)$ from He et al. (2015); Glorot & Bengio (2010), especially when considering that the group actions considered in this work are isometries. Indeed, upon initialization, $\mathbf{a}_i$ are normalized, such that they preserve distances. However, by freely optimizing them, we can get transformations that do scale the input by a small amount (to be precise, the squared norm of $\mathbf{a}_i$). In a sense, we thus get an overparameterization of the scaling since w_i also accounts for that. However, we empirically observe that having free scalars explicitly parameterizing the linear combination of Equation (11) can yield stabler learning. Further experimentation is needed to determine exactly why and when this is the case.

D.2. GCA-MLPs

GCA linear layers. To parameterize a GCA-MLP, we first specify in which algebra $\mathbb{G}_{p,q,r}$ we work and which blade components of the action $\mathbf{a}_i \in \mathbb{G}_{p,q,r}$ we want to parameterize. Further, similar to regular MLPs, we specify the number of input and output channels per layer. The geometric algebra linear layer Equation (11) then first embeds data in the algebra. For example, a vector in $\mathbb{R}^n$ gets embedded as 1-vector components in $\mathbb{G}_{p,q,r}$, leaving the rest of the blades to zero. The sandwich $\mathbf{a}_i \mathbf{x}_i \mathbf{a}_i^{-1}$ consists of two geometric products, which can efficiently be implemented as matrix multiplications. After this, we take a linear combination with the scalar weights w_i, which is straightforward to implement.

MSiLU. In this work we use linear combinations of basis blades $(f_k(\mathbf{x}))$ in our nonlinearity

$$[\mathbf{x}]_k \mapsto \mathrm{MSiLU}_k (\mathbf{x}) := \sigma (f_k(\mathbf{x})) \cdot [\mathbf{x}]_k , \tag{43}$$

where $f_k(\mathbf{x}) : \mathbb{G}_{p,q,r} \to \mathbb{R}$ and σ is the logistic function. This can be efficiently implemented as a linear layer in all modern deep learning frameworks.

Normalization. Finally, our normalization layer

$$[\mathbf{x}]_k \mapsto \frac{[\mathbf{x}]_k - \mathbb{E}[[\mathbf{x}]_k]}{\mathbb{E}[\|[\mathbf{x}]_k\|]} . \tag{44}$$

can be computed in a straightforward manner by computing an empirical average, and dividing by the average norm which is computed as described in Appendix B.

Architectures. For both the baseline MLP and GCA-MLP we use two hidden layers.

D.3. GCA-GNNs

We use the message propagation rules of Gilmer et al. (2017); Battaglia et al. (2018). Using multivectors, this amounts to

$$
\begin{aligned}
\mathbf{m}_{ij} &= \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l) \\
\bar{\mathbf{m}}_i &= \sum_{j \in \mathcal{N}(i)} \mathbf{m}_{ij} \\
\mathbf{h}_i^{l+1} &= \phi_h(\mathbf{h}_i^l, \mathbf{m}_i),
\end{aligned}
\tag{45}
$$

where $\mathbf{h}_i^l \in \mathbb{G}_{p,q,r}^c$ are the node features at node i, $\mathbf{m}_{ij} \in \mathbb{G}_{p,q,r}^c$ are messages between node i and node j, and $\bar{\mathbf{m}}_i \in \mathbb{G}_{p,q,r}^c$ is the message aggregation for each node. ϕ_c and ϕ_h are GCA-MLPs. Therefore, the whole GCA-GNN is a geometric template. For the Tetris experiment, we use a fully connected graph since all the nodes are correlated. Further, we use PyTorch Geometric (Fey & Lenssen, 2019) to implement the message passing algorithm.

Architectures. The baseline GNNs use the message passing updates as proposed by Gilmer et al. (2017); Battaglia et al. (2018). We use four message passing layers, where the message and update networks ϕ_e, ϕ_h (the non-GCA version) are implemented with scalar linear layers and LeakyReLU (Xu et al., 2015) activation functions. The graphs are fully connected over 32 nodes (8 objects, each consisting of four point coordinates). That allows for 12 input features (four input time-steps with 3 position values) for each node. We further have embedding MLPs and output MLPs that map from the 12 input features to a number of hidden features and back.

The GCA-GNNs replace all the baseline modules with there GCA counterparts. By coupling the x, y, z coordinates in single multivector entities, we now only have 4 input features. Again, to account for the additional parameters that the group actions introduce, we reduce the number of hidden channels.

D.4. GCA-ResNets

Let $y \in \mathbb{Y} \subset \mathbb{Z}^2$ be a two-dimensional spatial coordinate in the domain $\mathbb{Y}$. A convolution operation[7] convolves $x(y) : \mathbb{Y} \to \mathbb{R}^{k \times k \times c}$, with a filter map where we extract a $k \times k$ "patch" around the coordinate y with c channels:

$$
x \mapsto \sum_{i=1}^{c \times k \times k} w_i \cdot x_i,
\tag{46}
$$

where we suppress the y arguments. We repeat this procedure using different weights depending on the specific output channel.

Now, we discuss the geometric algebra analog of the above. Let $\mathbf{x}(y) : \mathbb{Y} \to \mathbb{G}_{p,q,r}^{k \times k \times c}$ be a multivector-valued feature map where we extract a $k \times k$ patch around the coordinate y with c channels. A group action convolution is then computed by

$$
\mathbf{x} \mapsto \sum_{i=1}^{c \times k \times k} w_i \cdot \mathbf{a}_i \mathbf{x}_i \mathbf{a}_i^{-1},
\tag{47}
$$

where we suppressed the spatial argument y. It is important to note that in Equation (47), the weights $w_i \in \mathbb{R}$, but the weights $\mathbf{a}_i \in \mathbb{G}_{p,q,r}$. That is the geometric transformations given by the sandwich product $\mathbf{a}_i \mathbf{x}_i \mathbf{a}_i^{-1}$ are linearly mixed by w_i afterwards.

Architectures. We use the same ResNet implementations as Brandstetter et al. (2022a). That is, we have 8 residual blocks, each consisting of two convolutional layers with 3×3 kernels (unit strides), shortcut connections, group normalization, and GELU activation functions (Hendrycks & Gimpel, 2016). Further, there are two embedding and two output layers. The input time steps and fields are all encoded as input channels. For example, for 4 input time steps and 3 input feature fields, we get 12 input channels. Since the output space is the same as the input space, we do not need downsampling layers. The CGA-ResNets directly replace each convolutional layer with a CGA layer, each GELU activation with MSiLU, and normalization with our normalization layer. To keep the weights similar, we have to reduce the number of feature channels (see Appendix E.2).

---

[7]In deep learning, a convolution operation in the forward pass is implemented as cross-correlation.

D.5. GCA-UNets

Gupta & Brandstetter (2022) show excellent performance of modern UNets on large-scale PDE surrogate tasks. We use similar architectures. Specifically, we encode the time steps and input feature fields as input channels to the model. We use convolutional embedding and projection layers to upsample (downsample) the input (output) channels. Throughout the network, we use 3×3 convolutions unit strides. We then use a channel multiplier structure of $(1, 2, 2, 2)$ which, similar to, e.g., Saharia et al. (2022), shifts the parameters from the high-resolution blocks to the low-resolution blocks, increasing model capacity without encountering egregious memory and computation costs. At each resolution, we have a residual block and up- or downsampling layer. A residual block consists of two convolutions, activations (GELU), normalizations (Group Normalization), and a skip connection. The bottleneck layer has two such residual layers.

For GCA-UNets, Clifford UNets, and rotational Clifford UNets, we, identically to the ResNet case, directly replace the layers, activations, and normalizations by the ones proposed in this paper and in Brandstetter et al. (2022a).

E. Experiments

E.1. Computational Resources

For the Tetris experiments, we used 1×40 GB NVIDIA A100 machines. The average training time for these experiments was 4 hours. The GCA-GNN took roughly 24 hours. However, as indicated before, several quick wins can be found to significantly reduce this. For the fluid dynamics experiments, we used $2 \times 4 \times 16$ GB NVIDIA V100 machines. The average training time was 24 hours. The GCANs, in this case, did not perform significantly worse than the baselines. On the other hand, the Clifford neural networks, due to their expensive normalization procedure, took significantly longer to optimize.

E.2. Tetris Experiment

Data generation. We take the shapes as provided by Thomas et al. (2018) and center them at the origin. For every Tetris shape, we randomly sample a rotation axis with a maximum angle of $0.05 \cdot 2\pi$ and translation directions with a maximum offset of 0.5. This leads to a rotation matrix $\mathbf{P}$ and translation vector $\mathbf{t}$. The position at time step t ($t = 0 \ldots 8$) can then be computed by applying $\mathbf{P}^t$ and $t \cdot \mathbf{t}$ to an initial coordinate. The discretized velocities are calculated by taking differences between the consecutive time steps. We further add slight deformations to the structures by adding small Gaussian noise.

Objective and loss function. After constructing a dataset of N_{train} such trajectories, we predict, given four input time steps, the next four input timesteps and compare them against the ground-truth trajectories for all objects. We define a loss function

$$\mathcal{L}_{\text{MSE}} := \frac{1}{N_{\text{locations}}} \sum_{t=1}^{N_t} \sum_{y=1}^{N_{\text{locations}}} \sum_{p=1}^{N_p} (x_{typ} - \hat{x}_{typ})^2 \tag{48}$$

that expresses for all train datapoints and for all time steps the discrepancy between the predicted three-dimensional locations and the ground-truth ones. When we predict positions, $N_p = 3$. When we also predict the velocities, we include those as well (i.e., then p sums to 6). In our experiments we have $N_t = 4$ predicted output time steps and $N_{\text{locations}} = 32$. We average this loss over 1024 validation and test trajectories.

GCAN implementation. In this experiment, we consider the geometric algebra $\mathbb{G}_{3,0,1}$. The highest-order proper isometry is a screw motion (a simultaneous rotation and translation), which is implemented using the components $\mathbf{a} := a_0 1 + a_{01} e_{01} + a_{02} e_{02} + a_{03} e_{03} + a_{12} e_{12} + a_{13} e_{13} + a_{23} e_{23} + a_{0123} e_{0123}$, where the coefficients are free parameters to be optimized. Since points in geometric algebra are defined by intersection of three planes. As such, we encode the x, y, z-coordinates of a point as $\mathbf{x} := x e_{012} + y e_{013} + z e_{023}$ and transform them using group actions $\mathbf{a}$. The e_{123} (the dual of e_0) parameterizes a distance from the origin, as such we get $\mathbf{x} := x e_{012} + y e_{013} + z e_{023} + \delta e_{123}$, where we leave δ in each layer as a free parameter. The full input representation to the GCA-MLP has $N_{\text{locations}} \cdot N_t$ channels. In contrast: a naive MLP has $N_{\text{locations}} \cdot N_t \cdot N_p$ input channels. The GCA-MLP then transforms this input using geometric algebra linear layers.

The GCA-GNN additionally encodes the positions as graph nodes and therefore only has N_t input channels. Each message and each node update network of the message passing layers is implemented as a GCA-MLP as described above.

When we include the velocities then $\mathbf{x} := v_x e_1 + v_y e_2 + v_z e_3 + x e_{0123} + y e_{013} + z e_{023}$. Importantly, we can use the same weights $\mathbf{a}$ to transform this multivector. However, to incorporate the knowledge of the additional velocities into the neural

network weights, retraining is required.

Model selection and optimization. We trained for $2^{17} = 131072$ steps of gradient descent for all data regimes as reported in the main paper. We did not run hyperparameter searches for these models and set them up using reasonable default architectures and settings. Specifically, for the MLP baseline we use 2 layers of 384 hidden features resulting in 444K parameters. We implement the O(3) and SO(3) baseline following Finzi et al. (2021), where input and output representation are $8 \times 4 \times 4T_1$ (T_1 is the vector representation). The O(3), SO(3) and MotorMLP baselines have 256 channels each to obtain equal parameters. The GCA-MLP has 128 channels to equal the parameter count. The GNN baselines use four layers of message passing with 136 (small models, resulting in 444K parameters) or 192 (large models, resulting in 893K parameters) hidden features. The EdgeConv baseline uses 256 hidden features, resulting in 795K parameters. The GCA-GNN equals the parameters of the small GNNs, using four hidden layers of 64 hidden features.

For the velocity experiment, the MLP baseline uses 2 hidden layers with 248 features to obtain the same parameter count as the large GNN baseline, i.e., 893K. The GNN baseline has four message passing layers with 192 hidden features, resulting in 900K parameters. The GCA-MLP and GCA-GNN use the same number of hidden features as in the positions experiment, i.e., 444K.

For all models we use the Adam (Kingma & Ba, 2014) optimizer with the default parameter settings (i.e., a learning rate of 10^{-3}). We did no further extensive hyperparameter or architecture searches for these models and kept them to reasonable default settings.

E.3. Extended results

In Table 3 we present numerically the results on the Tetris experiment for MLP-style models (also presented in Figure 5).

Training trajectories	256	1024	4096	16384
MLP	4.5061	1.2180	0.2184	0.1596
Motor-MLP	3.5732	0.8655	0.4249	0.4020
SO(3)-MLP	1.5153	0.9504	0.2172	0.1176
O(3)-MLP	1.4232	0.7224	0.1584	0.0876
GCA-MLP (Ours)	**0.9852**	**0.0420**	**0.00732**	**0.0061**

Table 3. Mean squared error of MLP-style models on the tetris experiment.

In Table 4 we present numerically the results on the Tetris experiment for GNN-style models (also presented in Figure 5).

Training trajectories	256	1024	4096	16384
EdgeConv	1.572	0.2406	0.0240	0.0123
GNN (S)	0.3012	0.0432	0.0032	0.0020
GNN (S, +d)	0.2887	0.0408	0.0040	0.0019
GNN (L)	0.2879	0.0504	0.0043	0.0017
GNN (L, +d)	0.2793	0.0516	0.0030	0.0016
GCA-GNN (Ours)	**0.2403**	**0.0012**	$\mathbf{6.1 \cdot 10^{-4}}$	$\mathbf{5.4 \cdot 10^{-4}}$

Table 4. Mean squared error of GNN-style models on the tetris experiment.

In Table 5 we present numerically the results on the Tetris experiment where we include input and output velocities (also presented in Figure 5).

Training trajectories	256	1024	4096	16384
MLP	3.2403	0.5040	0.1560	0.0912
GCA-MLP	1.6560	0.0984	0.0418	0.0432
GNN	0.0504	0.0086	0.0015	$8.8 \cdot 10^{-4}$
GCA-GNN (Ours)	**0.0122**	**0.0022**	$\mathbf{4.1 \cdot 10^{-4}}$	$\mathbf{2.6 \cdot 10^{-4}}$

Table 5. Mean squared error of GNN-style models on the tetris experiment when including velocity inputs and outputs.

E.4. Speedyweather

Shallow water equations. The shallow water equations describe a thin layer of fluid of constant density in hydrostatic balance, bounded from below by the bottom topography and from above by a free surface. For example, the deep water propagation of a tsunami can be described by the shallow water equations, and so can a simple weather model. The shallow water equations read:

$$
\begin{aligned}
\frac{\partial v_x}{\partial t} + v_x \frac{\partial v_x}{\partial x} + v_y \frac{\partial v_x}{\partial y} + g \frac{\partial \eta}{\partial x} &= 0 \,, \\
\frac{\partial v_y}{\partial t} + v_x \frac{\partial v_y}{\partial x} + v_y \frac{\partial v_y}{\partial y} + g \frac{\partial \eta}{\partial y} &= 0 \,, \\
\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial x}\left[(\eta + h)v_x\right] + \frac{\partial}{\partial y}\left[(\eta + h)v_y\right] &= 0 \,,
\end{aligned}
\tag{49}
$$

where v_x is the velocity in the x-direction, or zonal velocity, v_y is the velocity in the y-direction, or meridional velocity, g is the acceleration due to gravity, $\eta(x, y)$ is the vertical displacement of free surface, which subsequently is used to derive pressure fields; $h(x, y)$ is the topography of the earth's surface.

Simulation. We consider a modified implementation of the `SpeedyWeather.jl`[8](Klöwer et al., 2022) package, obtaining data on a 192×96 periodic grid ($\Delta x = 1.875°$, $\Delta y = 3.75°$). This package uses the shallow water equations (Vreugdenhil, 1994), a specific instance of a Navier-Stokes fluid dynamics system, to model global weather patterns. The temporal resolution is $\Delta t = 6$ h, meaning we predict these patterns six hours into the future. SpeedyWeather internally uses a leapfrog time scheme with a Robert and William's filter to dampen the computational modes and achieve 3rd oder accuracy. `SpeedyWeather.jl` is based on the atmospheric general circulation model `SPEEDY` in Fortran (Molteni, 2003; Kucharski et al., 2013).

Objective. Given four input time steps, we predict the next time step and optimize the loss function

$$
\mathcal{L}_{\text{MSE}} := \frac{1}{N_y} \sum_{t=1}^{N_t} \sum_{n=1}^{N_{\text{fields}}} \sum_{y \in \mathbb{Y}} (x_{tny} - \hat{x}_{tny})^2,
\tag{50}
$$

where $\mathbb{Y} \subset \mathbb{Z}^2$ is the (discretized) spatial domain of the PDE, N_t is the number of prediction time steps, and N_{fields} is the number of field components. For the shallow water equations we have two velocity components and one scalar component. Here, $\hat{x}_{n,t,y}$ is the predicted value at output time step t for field component n and at spatial coordinate y. $x_{n,t,y}$ is its ground-truth counterpart. We consider only one-step ahead loss, where $N_t = 1$, as empirically it has been found that this naturally generalizes to better rollout trajectories (Brandstetter et al., 2022a; Gupta & Brandstetter, 2022).

GCAN implementation. It can be shown that the $\mathbb{G}_{3,0,0}$ geometric algebra linear layer $T_{g,w}(\mathbf{x}) = \sum_{i=1}^{c} w_i \cdot \mathbf{a}_i \mathbf{x}_i \mathbf{a}_i^{-1}$ reduces to $T_{g,w}(\mathbf{x}) = \sum_{i=1}^{c} w_i \cdot \mathbf{R}_i \mathbf{x}_i$.

$$
\begin{aligned}
\mathbf{ax} &= (a_0 + a_{12}e_{12} + a_{13}e_{13} + a_{23}e_{23})(x_1 e_1 + x_2 e_2 + x_3 e_3) \\
&= a_0 x_1 e_1 + a_0 x_2 e_2 + a_0 x_3 e_3 \\
&\quad - a_{12} x_1 e_2 + a_{12} x_2 e_1 + a_{12} x_3 e_{123} \\
&\quad - a_{13} x_1 e_3 - a_{13} x_2 e_{123} + a_{13} x_3 e_1 \\
&\quad + a_{23} x_1 e_{123} - a_{23} x_2 e_3 + a_{23} x_3 e_2,
\end{aligned}
\tag{51}
$$

---

[8] https://github.com/milankl/SpeedyWeather.jl

which reduces to

$$
\begin{aligned}
\mathbf{ax} = {} & (a_0 x_1 + a_{12} x_2 + a_{13} x_3) e_1 \\
& + (a_0 x_2 - a_{12} x_1 + a_{23} x_3) e_2 \\
& + (a_0 x_3 - a_{13} x_1 - a_{23} x_2) e_3 \\
& + (a_{12} x_3 - a_{13} x_2 + a_{23} x_1) e_{123}
\end{aligned}
\tag{52}
$$

Using grade reversion (Appendix B), we get $\mathbf{a}^{-1} = a_0 - a_{12} e_{12} - a_{13} e_{13} - a_{23} e_{23}$. Then,

$$
\begin{aligned}
\mathbf{axa}^{-1} = {} & (a_0^2 x_1 + a_0 a_{12} x_2 + a_0 a_{13} x_3) e_1 \\
& + (a_0^2 x_2 - a_0 a_{12} x_1 + a_0 a_{23} x_3) e_2 \\
& + (a_0^2 x_3 - a_0 a_{13} x_1 - a_0 a_{23} x_2) e_3 \\
& + (a_0 a_{12} x_3 - a_0 a_{13} x_2 + a_0 a_{23} x_1) e_{123} \\
& + (-a_{12} a_0 x_1 - a_{12}^2 x_2 - a_{12} a_{13} x_3) e_2 \\
& + (a_{12} a_0 x_2 - a_{12}^2 x_1 + a_{12} a_{23} x_3) e_1 \\
& + (-a_{12} a_0 x_3 + a_{12} a_{13} x_1 + a_{12} a_{23} x_2) e_{123} \\
& + (a_{12}^2 x_3 - a_{12} a_{13} x_2 + a_{12} a_{23} x_1) e_3 \\
& + (-a_{13} a_0 x_1 - a_{13} a_{12} x_2 - a_{13}^2 x_3) e_3 \\
& + (a_{13} a_0 x_2 - a_{13} a_{12} x_1 + a_{13} a_{23} x_3) e_{123} \\
& + (a_{13} a_0 x_3 - a_{13}^2 x_1 - a_{13} a_{23} x_2) e_1 \\
& + (-a_{13} a_{12} x_3 + a_{13}^2 x_2 - a_{13} a_{23} x_1) e_2 \\
& + (-a_{23} a_0 x_1 - a_{23} a_{12} x_2 - a_{23} a_{13} x_3) e_{123} \\
& + (-a_{23} a_0 x_2 + a_{23} a_{12} x_1 - a_{23}^2 x_3) e_3 \\
& + (a_{23} a_0 x_3 - a_{23} a_{13} x_1 - a_{23}^2 x_2) e_2 \\
& + (a_{23} a_{12} x_3 - a_{23} a_{13} x_2 + a_{23}^2 x_1) e_1,
\end{aligned}
\tag{53}
$$

which again reduces to

$$
\begin{aligned}
\mathbf{axa}^{-1} = {} & (a_0^2 x_1 + 2 a_0 a_{12} x_2 + 2 a_0 a_{13} x_3 - a_{12}^2 x_1 + 2 a_{12} a_{23} x_3 - a_{13}^2 x_1 - 2 a_{23} a_{13} x_2 + a_{23}^2 x_1) e_1 \\
& + (a_0^2 x_2 - 2 a_0 a_{12} x_1 + 2 a_0 a_{23} x_3 - a_{12}^2 x_2 - 2 a_{12} a_{13} x_3 + a_{13}^2 x_2 - 2 a_{13} a_{23} x_1 - a_{23}^2 x_2) e_2 \\
& + (a_0^2 x_3 - 2 a_0 a_{13} x_1 - 2 a_0 a_{23} x_2 + a_{12}^2 x_3 - 2 a_{12} a_{13} x_2 + 2 a_{12} a_{23} x_1 - a_{13}^2 - a_{23}^2 x_3) e_3
\end{aligned}
\tag{54}
$$

where we see that the trivector components cancel! Collecting the terms, we can define

$$
\mathbf{R} := \begin{bmatrix}
a_0^2 - a_{12}^2 - a_{13}^2 + a_{23}^2 & 2 a_0 a_{12} - 2 a_{23} a_{13} & 2 a_0 a_{13} + 2 a_{12} a_{23} \\
-2 a_0 a_{12} - 2 a_{13} a_{23} & a_0^2 - a_{12}^2 + a_{13}^2 - a_{23}^2 & 2 a_0 a_{23} - 2 a_{12} a_{13} \\
-2 a_0 a_{13} + 2 a_{12} a_{23} & -2 a_0 a_{23} - 2 a_{12} a_{13} & a_{12}^2 + a_0^2 - a_{13}^2 - a_{23}^2
\end{bmatrix}
\tag{55}
$$

Here, $\mathbf{R}_i$ is a rotation matrix that acts on the vector components of $\mathbf{x}_i$. This is a much more efficient implementation than the sandwich operation, and resembles the rotational layer of Brandstetter et al. (2022a) but does not implement a scalar part (which would go against the concept of a group action layer).

Model selection and optimization. For ResNet architectures we consider 128 and 144 feature channels, the latter matching the number of parameters of the GCA-ResNets. Out of these two, the best performing model was reported in the paper. These channels are kept constant (apart from embedding and decoding layers) throughout the network. The Clifford ResNets and GCA ResNet consider 64 channels. These architectures then have roughly 3M parameters. The UNet models consider 64 and 70 base channels, the latter matching the parameters of the GCA-UNet (58M). The Clifford and GCA counterparts have 32 channels. At 448 training trajectories, we tested all models across two learning rates ($2 \cdot 10^{-4}$ and $5 \cdot 10^{-4}$), and different normalization schemes. Normalization turned out to be beneficial at all times. We further closely followed the hyperparameter settings as reported in Gupta & Brandstetter (2022)[9]. Further, we tested for the GCA models whether a

---

[9] https://microsoft.github.io/pdearena/

learned linear combination, a simple summation, or averaging in the MSiLU layers works best. Summation and a learned linear combination turned out to be the most promising.

We used the Adam optimizer with the best performing learning rate, and use cosine annealing (Loshchilov & Hutter, 2016) with linear warmup. We trained at all data-regimes for 50 epochs.

Extended results.

In Table 6 we present the mean squared error values for ResNet style models that are also presented in Figure 6.

Training trajectories	112	224	448	892
ResNet	0.0248	0.01310	0.0076	0.0055
CResNet	0.0457	0.01685	0.0076	0.0041
CResNet$_{rot}$	0.0269	0.01040	**0.0045**	**0.0031**
GCA-ResNet (Ours)	**0.0204**	**0.00920**	0.0050	0.0036

Table 6. Mean squared error of ResNet-style models on the shallow water equations experiment.

In Table 7 we present the mean squared error values for UNet-style models that are also presented in Figure 6.

Training trajectories	112	224	448	892
CUNet	0.0056	0.0013	$1.59 \cdot 10^{-4}$	$1.28 \cdot 10^{-4}$
CUNet$_{rot}$	0.0036	$9.83 \cdot 10^{-4}$	$2.05 \cdot 10^{-4}$	$1.31 \cdot 10^{-4}$
UNet	$9.69 \cdot 10^{-4}$	$3.73 \cdot 10^{-4}$	$1.76 \cdot 10^{-4}$	$8.10 \cdot 10^{-5}$
GCA-UNet (Ours)	$\mathbf{8.01 \cdot 10^{-4}}$	$\mathbf{2.17 \cdot 10^{-4}}$	$\mathbf{6.85 \cdot 10^{-5}}$	$\mathbf{3.95 \cdot 10^{-5}}$

Table 7. Mean squared error of UNet-style models on the shallow water equations experiment.

Finally, we show an examplary predicted and ground-truth trajectory for our GCA-UNet in Figure 9 and for the baseline UNet in Figure 10.

Runtimes.

	F (s/it)	B (s/it)	F + B (s/it)	Parameters (M)	Compute
ResNet	0.10	0.17	0.27	3	2×4 NVIDIA V100
CResNet	0.63	0.95	1.59	3	2×4 NVIDIA V100
CResNet$_{rot}$	0.62	0.94	1.56	3	2×4 NVIDIA V100
GCA-ResNet	0.13	0.19	0.33	3	2×4 NVIDIA V100
UNet	0.25	0.40	0.65	58	2×4 NVIDIA V100
CUNet	0.64	0.97	1.61	58	2×4 NVIDIA V100
CUNet$_{rot}$	0.63	0.95	1.59	58	2×4 NVIDIA V100
GCA-UNet	0.42	0.61	1.03	58	2×4 NVIDIA V100

Table 8. Overview of the estimated forward and backward runtimes of the implemented models.

E.5. Navier-Stokes

The incompressible Navier-Stokes equations are built upon momentum and mass conservation of fluids. For the velocity flow field v, the incompressible Navier-Stokes equations read

$$\frac{\partial v}{\partial t} = -v \cdot \nabla v + \mu \nabla^2 v - \nabla p + f \,, \tag{56}$$

$$\nabla \cdot v = 0 \,, \tag{57}$$

where $v \cdot \nabla v$ is the convection of the fluid, $\mu \nabla^2 v$ the diffusion controlled via the viscosity parameter ν of the fluid, ∇p the internal pressure, and f an external buoyancy force. Convection is the rate of change of a vector field along a vector
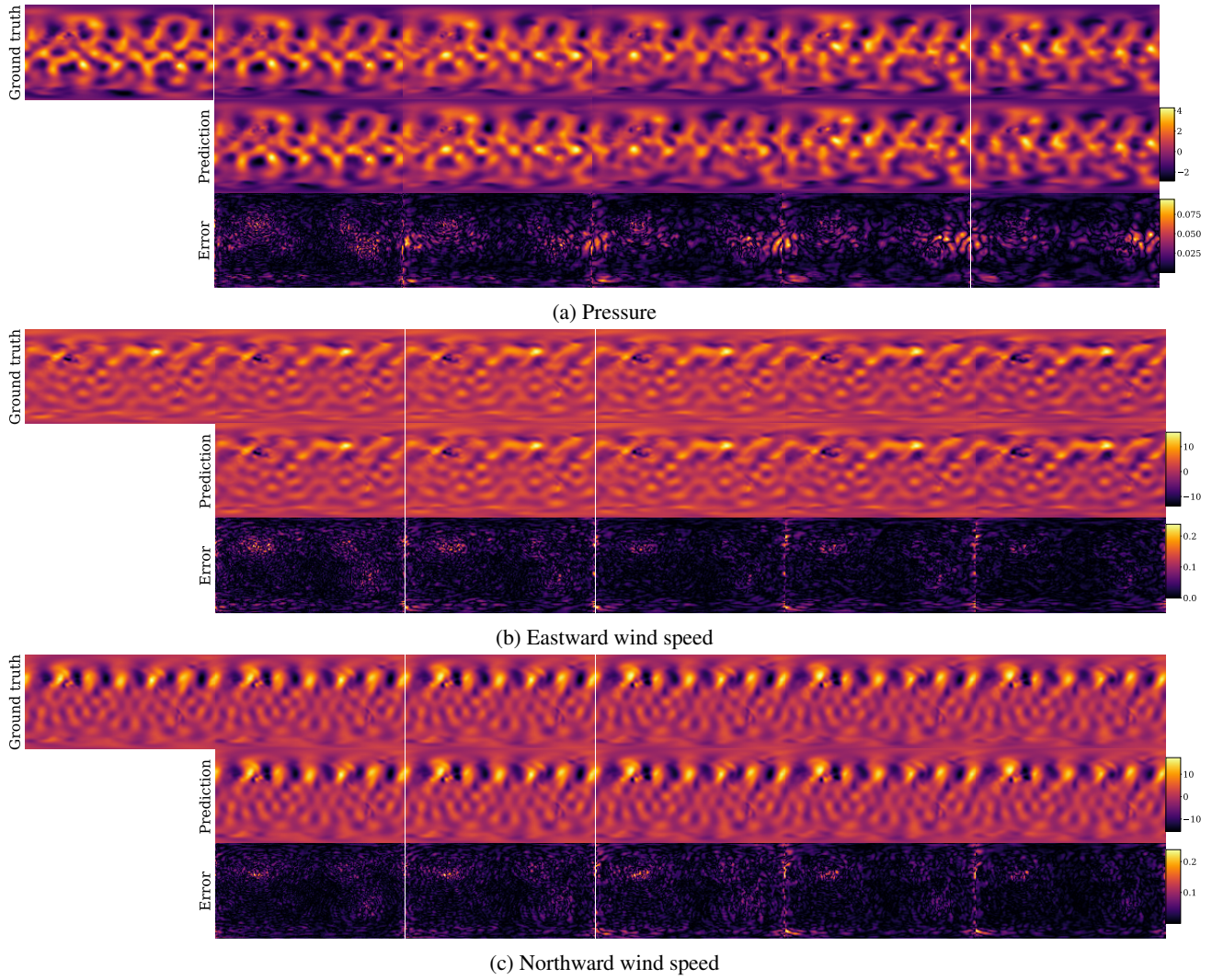
(a) Pressure

(b) Eastward wind speed

(c) Northward wind speed

Figure 9. Example rollout from GCA-UNet on the 6 hour shallow water equations.

(a) Pressure

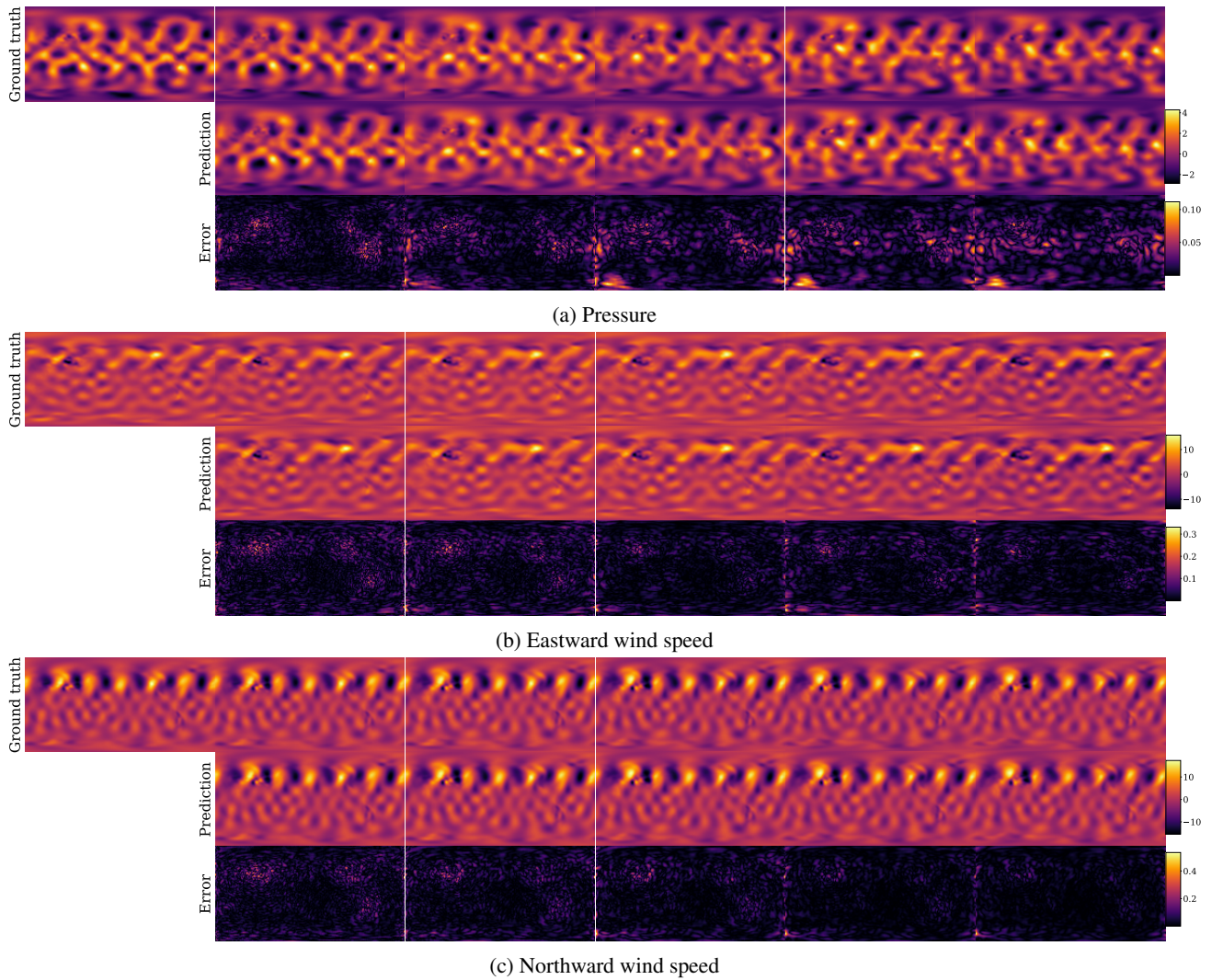(b) Eastward wind speed

(c) Northward wind speed

Figure 10. Example rollout from UNet on the 6 hour shallow water equations. Note that the bottom left corner of the pressure field visibly differs from our GCA-UNet rollout in Figure 9.

field (in this case along itself), and diffusion is the net movement form higher valued regions to lower concentration regions. Additional to the velocity field $v(x)$, we introduce a scalar field $s(x)$ representing a scalar quantity, such as particle concentration or smoke density in our case, that is being *advected*, i.e., transported along the velocity field.

We implement the 2D Navier-Stokes equation using $\Phi\mathrm{Flow}^{10}$ (Holl et al., 2020b). Solutions are obtained by solving for the pressure field, and subsequently subtracting the gradients of the pressure field.

We obtain data on a closed domain with Dirichlet boundary conditions ($v = 0$) for the vector (velocity), and Neumann boundaries $\frac{\partial s}{\partial x} = 0$ for the scalar field. The grid has a spatial resolution of 128×128 ($\Delta x = 0.25$, $\Delta y = 0.25$), and temporal resolution of $\Delta t = 1.5s$. The viscosity is set to $\nu = 0.01$. The scalar field is initialized with random Gaussian noise fluctuations, and the velocity field is initialized to 0. We run the simulation for $21.0s$ and sample every $1.5s$. Trajectories contain scalar smoke density, and vector velocity fields at 14 different time points.

GCAN implementation. We use the algebra $\mathbb{G}_{3,0,0}$ and therefore use the same implementation as for the shallow water experiment.

Objective. Similar to the shallow water equations, we have

$$\mathcal{L}_{\mathrm{MSE}} := \frac{1}{N_y} \sum_{t=1}^{N_t} \sum_{n=1}^{N_{\mathrm{fields}}} \sum_{y \in \mathbb{Y}} (x_{tny} - \hat{x}_{tny})^2. \tag{58}$$

Model selection and optimization. We used the best-performing models in the shallow water equations: the GCA-UNet and UNet and compare them at different number of training trajectories: 832, 2080, and 5200. For the UNet, we tested a version with 64 input channels and 72 input channels, which matches the parameter of the GCA counterpart (58M). We searched across learning rates of $2 \cdot 10^{-4}$ and $5 \cdot 10^{-4}$, or all architectures including the baseline UNets. For the GCA-UNet, we further tested normalization schemes and activation schemes, obtaining similar results to the shallow water experiment.

We used the Adam optimizer with best performing learning rate, and cosine annealing (Loshchilov & Hutter, 2016) with linear warmup. We trained at all data-regimes for 50 epochs.

Extended results. We present in Table 9 the numerical results of our Navier-Stokes experiment also presented in the main paper.

Training trajectories	832	2080	5200
UNet	0.00290	0.001040	$9.71 \cdot 10^{-4}$
GCA-UNet (Ours)	**0.00194**	$\mathbf{9.48 \cdot 10^{-4}}$	$\mathbf{9.21 \cdot 10^{-4}}$

Table 9. Mean squared error of UNet models on the Navier-Stokes experiment.

F. Clifford Algebra

This section provides a pedagogical insight into how a Clifford algebra and its multiplication rules are constructed in a basis-independent way. Consider a vector space V whose characteristic is not 2. A Clifford algebra over V has an distributive and associative bilinear product with the construction

$$v^2 - \langle v, v \rangle 1 = 0, \tag{59}$$

for $v \in V$. Here, $\langle \cdot, \cdot \rangle$ denotes a symmetric quadratic form, and 1 is the multiplicative identity. In other words, multiplication of a vector with itself identifies with the quadratic form. This yields the identity

$$(u + v)^2 = \langle u + v, u + v \rangle, \tag{60}$$

from which the fundamental Clifford identity (geometric product) arises:

$$uv = 2\langle u, v \rangle - vu. \tag{61}$$

---

^{10}https://github.com/tum-pbs/PhiFlow

As such, we see that

$$\frac{1}{2}(uv + vu) = \langle u, v \rangle. \tag{62}$$

Further, we can construct an antisymmetric part

$$u \wedge v := \frac{1}{2}(uv - vu), \tag{63}$$

with

$$u \wedge v = -v \wedge u. \tag{64}$$

This is commonly referred to as *wedge product*. As such, we see that we get the identity

$$uv = \langle u, v \rangle + u \wedge v, \tag{65}$$

which is referred to as the *geometric product* and directly follows from the fundamental Clifford identity. In Appendix B we show how the geometric product is computed in practice after choosing a basis.

G. Pseudocode

Algorithm 1 Pseudocode for obtaining Clifford kernels to compute geometric products. Here, $\mathbf{a} \in \mathbb{G}_{p,q,r}^{c_{\text{out}} \times c_{\text{in}}}$, where c_{in} is the number of input and c_{out} the number of output channels of the layer. A Cayley table $M \in \{-1, 0, 1\}^{n \times n \times n}$ ($n = p + q + r$) is provided by the algebra and its signature to effectively compute geometric products.

Function `GetCliffordKernel`(M: *Cayley Table*, $\mathbf{a}$: *Action*):
    $K_{jvkw}^{\text{left}} \leftarrow \sum_i M_{ijk} \mathbf{a}_{vwi}$
    $K_{jviw}^{\text{right}} \leftarrow \sum_k M_{ijk} \mathbf{a}_{vwk}$
    return $K^{\text{left}}, K^{\text{right}}$

Algorithm 2 Pseudocode for the geometric algebra conjugate linear layer.

Function `ConjugateLinear`(x: *Input*, $\mathbf{a}$: *Actions*, w: *Weights*, A: *GeometricAlgebra*):
    $M \leftarrow$ `CayleyTable`(A)
    $\mathbf{a}^{-1} \leftarrow$ `Reverse`($\mathbf{a}$)
    $\mathbf{x} \leftarrow$ `Embed`(A, x)
    $K^{\text{left}}, \_ \leftarrow$ `GetCliffordKernel`($M, \mathbf{a}$)
    $\_, K^{\text{right}} \leftarrow$ `GetCliffordKernel`($M, \mathbf{a}^{-1}$)
    $\mathbf{x} \leftarrow$ `GroupActionLinear`($w, K^{\text{left}}, K^{\text{right}}, \mathbf{x}$)
    $x \leftarrow$ `Retrieve`($A, \mathbf{x}$)
    return x

Algorithm 3 Pseudocode for the multivector sigmoid linear units.

Function MSiLU (*x: Input, w: Weights, A: GeometricAlgebra, Agg: Aggregation*) :

  $\mathbf{x} \leftarrow$ Embed(A, x)

  for $[\mathbf{x}]_k$ *in* $\mathbf{x}$ **do**

    if *Agg = "linear"* **then**

      $[\mathbf{x}]_k \leftarrow \sigma\left(\text{Linear}(\mathbf{x}, w_k)\right) \cdot [\mathbf{x}]_k$

    end

    if *Agg = "sum"* **then**

      $[\mathbf{x}]_k \leftarrow \sigma\left(\text{Sum}(\mathbf{x})\right) \cdot [\mathbf{x}]_k$

    end

    if *Agg = "mean"* **then**

      $[\mathbf{x}]_k \leftarrow \sigma\left(\text{Mean}(\mathbf{x})\right) \cdot [\mathbf{x}]_k$

    end

  end

  $x \leftarrow$ Retrieve$(A, \mathbf{x})$

  return x

Algorithm 4 Pseudocode for the geometric algebra normalization layer.

Function GCANormalize (*x: Input, s: Rescaling, A: GeometricAlgebra*) :

  $\mathbf{x} \leftarrow$ Embed(A, x)

  for $[\mathbf{x}]_k$ *in* $\mathbf{x}$ **do**

    $[\mathbf{x}]_k \leftarrow s_k \cdot ([\mathbf{x}]_k - \text{ChannelsAverage}([\mathbf{x}]_k)) / \text{ChannelsAverage}(\|[\mathbf{x}]_k\|)$

  end

  $x \leftarrow$ Retrieve$(A, \mathbf{x})$

  return x

Algorithm 5 Pseudocode for the $\mathbb{G}_{3,0,0}$ conjugate linear layer.

Function ConjugateLinear (*x: Input, $\mathbf{a}$: Actions, w: Weights, A: GeometricAlgebra*) :

  $\mathbf{x} \leftarrow$ Embed(A, x)

  $\mathbf{R} \leftarrow$ GetRotationalKernel$(\mathbf{a})$

  $\mathbf{x} \leftarrow$ RotationalConvolution$(w, \mathbf{R}, \mathbf{x})$

  $x \leftarrow$ Retrieve$(A, \mathbf{x})$

  return x