Design and Evaluation of a Geometric Algebra-Based Graph Neural Network for Molecular Property Prediction

Anonymous Full Paper
Submission 18

Abstract

Geometric Algebra (GA) provides a unified framework for representing scalars, vectors, and higher-dimensional geometric elements, along with the geometric product, an operation that mixes information across these components in an equivariant manner. While GA has recently attracted attention in deep learning, its potential for molecular property prediction remains underexplored. We introduce GA-GNN, a novel equivariant graph neural network that extends message passing architectures from separate scalar and vector features to multivector representations, and employs sequences of geometric product layers as the core update mechanism. Evaluated on the QM9 benchmark, GA-GNN achieves competitive performance with the recent state-of-the-art while demonstrating that GA-based representations can simplify architecture design. These results highlight the potential of GA for building expressive equivariant message passing networks for molecular property prediction.

1 Introduction

Equivariant neural networks have emerged as powerful tools for learning from data with geometric structure, such as molecules, by ensuring that learned features transform consistently under translation, rotation and reflection in 3-dimensional space. Current approaches often represent features as scalars, vectors, or higher-order tensors, with message passing architectures designed to respect these symmetries. We propose an alternative based on Geometric Algebra (GA), a unified mathematical framework for representing and manipulating geometric entities. GA extends beyond scalars and vectors to include geometric objects of higher dimensionality such as oriented planes and volumes, all combined in a single object called a multivector. A central operation in GA, the geometric product, mixes information not only within the same representational level (e.g., vector–vector) but also across levels (e.g., scalar–vector, vector–plane) in a principled manner that is equivariant to rotation and reflection. This makes GA a natural candidate for message passing architectures, where information from different geometric orders must be combined efficiently and consistently. While GA has recently attracted interest in deep learning, its potential for molecular property prediction remains underexplored. In this work, we present GA-GNN, a novel equivariant graph neural network for molecular property prediction. The architecture of the model is inspired by elements from PaiNN [1], which we extend to work on multivectors, rather than decoupled scalar and vector features. Its core is based on designing a new update block which uses sequences of geometric product layers as proposed in Clifford Group Equivariant Neural Networks (CGENNs) [2] to compute residual updates. Additionally, the readout layer can be simplified by removing target-specific networks, instead allowing flexible selection of the multivector components relevant to a given target property. We evaluate GA-GNN on the QM9 dataset and study several ablations and architectural variations. To the best of our knowledge, this is the first application of a GA-based model to molecular property prediction on QM9, offering new insights into the use of multivector representations and geometric product operations in this domain. An extended discussion is provided in [3].

2 Background & Related Work

2.1 Molecular Property Prediction

In molecular property prediction, the goal is to learn a function that maps molecular structures to their corresponding properties, which may include chemical, physical, or biological characteristics. In our setting, molecules are represented as graphs embedded in 3D space, where nodes correspond to atoms and edges capture chemical bonds or spatial proximity. Formally, a molecular graph is denoted $G = (V, E)$, where V is the set of atoms (nodes) and $E \subseteq V \times V$ is the set of edges. Each atom $v \in V$ is associated with a spatial position $\mathbf{x}_v \in \mathbb{R}^3$ and an atom type, and each edge $(v, u) \in E$ may be associated with geometric features such as interatomic distance $\|\mathbf{x}_u - \mathbf{x}_v\|$ and relative position (edge vector) $\mathbf{r}_{vu} = \mathbf{x}_u - \mathbf{x}_v$. The neighborhood of a node v is defined as $\mathcal{N}(v) = \{u \in V \mid (v, u) \in E\}$. These representations and features are used as inputs to graph-based deep learning models.

A wide range of deep learning methods have been developed for molecular property prediction. The majority of these approaches are based on message

passing neural networks (MPNN). MPNNs learn node embeddings by iteratively aggregating and updating information from neighboring nodes. At each message passing round $t \in \{1, \dots, T\}$, the embedding of node v is denoted $\mathbf{h}_v^{(t)} \in \mathbb{R}^d$, where d is the feature dimension. Each round consists of a message aggregation step followed by an update, and after T rounds, a readout layer computes the final graph-level output based on the node embeddings:

$$\mathbf{m}_v^{(t+1)} = \bigoplus_{u \in \mathcal{N}(v)} M_t(\mathbf{h}_v^{(t)}, \mathbf{h}_u^{(t)}), \tag{1}$$

$$\mathbf{h}_v^{(t+1)} = U_t(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)}), \tag{2}$$

$$\hat{y} = R(\{\mathbf{h}_v^T \mid v \in V\}). \tag{3}$$

Here M_t is called the *message function*, U_t is called the *update function*, $\bigoplus$ is a permutation invariant aggregation operation (typically the sum), and R is called the *readout function* [4].

Early MPNN-based models for molecular property prediction primarily relied on invariant, scalar-valued features such as pairwise interatomic distances, bond angles, and torsion angles [4–8]. More recent *equivariant* MPNNs incorporate vector-valued features that transform consistently under geometric transformations such as rotations and reflections [1, 9–11]. Later generations of models further extends this by using higher-order tensor features, which are updated through operations involving spherical harmonics and tensor products [12–15]. Lastly, non message-passing approaches including transformer models have also achieved promising results [16–18].

2.2 Geometric Algebra

Definition. Let $\{e_1, ..., e_n\}$ be the basis of an n-dimensional vector space V. The *geometric algebra* $\mathbb{G}_{pqr}$ is an algebra generated from the basis vectors e_i in which the following two conditions hold:

1. For all i, the squared basis vectors satisfy:

$$e_i^2 = \begin{cases} +1 & \text{for } i = 1, \dots, p, \\ -1 & \text{for } i = p{+}1, \dots, p{+}q, \\ 0 & \text{for } i = p{+}q{+}1, \dots, p{+}q{+}r, \end{cases} \tag{4}$$

   where the integers $p, q, r \geq 0$ are the number of basis vectors that square to $+1$, -1 and 0 respectively.

2. For $i \neq j$ the basis vectors anti-commute:

$$e_i e_j = -e_j e_i. \tag{5}$$

The total dimension of the space is $n = p + q + r$, and the geometric algebra $\mathbb{G}_{pqr}$ has 2^n basis elements. In the remainder of this section, we focus on the specific case $\mathbb{G}_{3,0,0}$, which is the algebra used in the proposed GA-based GNN architecture. The *geometric product* of two vectors $\mathbf{a}$ and $\mathbf{b} \in V$ is defined as

$$\mathbf{ab} = \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \wedge \mathbf{b}, \tag{6}$$

where $\mathbf{a} \cdot \mathbf{b}$ is the inner product known from traditional vector algebra, and $\mathbf{a} \wedge \mathbf{b}$ is the *outer product*, which is also called the wedge product. The outer product produces an object that represents an oriented plane spanned by $\mathbf{a}$ and $\mathbf{b}$, which is called a *bivector*.

Table 1. Basis blades in $\mathbb{G}_{3,0,0}$ grouped by grade.

Name	Grade	Dimension	Basis blades	Square
Scalar	0	1	1	+1
Vector	1	3	$e_1,\ e_2,\ e_3$	+1
Bivector	2	3	$e_{12},\ e_{23},\ e_{31}$	−1
Trivector	3	1	e_{123}	−1

More generally, the geometric product of k basis vectors produces a *k-blade*. Since the basis vectors are orthogonal, the inner product between them equals zero. As a result, the geometric product between the basis vector reduces to the outer product. Using the shorthand convention $e_{ij} := e_i e_j$, we can form the following blades from the basis vectors:

$$\begin{aligned} e_{12} &:= e_1 e_2 = e_1 \wedge e_2, \\ e_{23} &:= e_2 e_3 = e_2 \wedge e_3, \\ e_{31} &:= e_3 e_1 = e_3 \wedge e_1, \\ e_{123} &:= e_1 e_2 e_3 = e_1 \wedge e_2 \wedge e_3. \end{aligned} \tag{7}$$

The last blade e_{123} is called a *trivector*. In $\mathbb{G}_{(3,0,0)}$ it is also referred to as the *pseudoscalar* as it is one-dimensional and changes sign under reflection. Geometrically it represents an oriented volume that encodes the handedness of space, meaning that the sign indicates whether the orientation is right-handed or left-handed, and the magnitude corresponds to the volume. Each blade has a *grade* equal to the dimension of the subspace it represents, i.e. grade-0 blades are scalars, grade-1 blades are vectors, grade-2 blades are bivectors and grade-3 vectors are trivectors. Table 1 summarizes the geometric algebra $\mathbb{G}_{3,0,0}$.

Linear combinations of blades of different grades are called *multivectors*. In $\mathbb{G}_{3,0,0}$ a multivector A can be written as:

$$\begin{aligned} A = \underbrace{\lambda_0}_{\text{Scalar}} &+ \underbrace{\lambda_1 e_1 + \lambda_2 e_2 + \lambda_3 e_3}_{\text{Vector}} \\ &+ \underbrace{\lambda_4 e_{12} + \lambda_5 e_{23} + \lambda_6 e_{31}}_{\text{Bivector}} \\ &+ \underbrace{\lambda_7 e_{123}}_{\text{Trivector}}. \end{aligned} \tag{8}$$

Hence, A is an 8-dimensional object consisting of scalar, vector, bivector, and trivector components.

The k-grade of a multivector is denoted $\langle A \rangle_k$. The geometric product is defined between two multivectors resulting in a new multivector that combines contributions from interactions between the grade-components of each multivector. The geometric product between multivectors in $\mathbb{G}_{(3,0,0)}$ is derived in Appendix A. This enables an organized way to mix information across representational levels of different dimensions all within a single consistent operation. Additionally, the geometric product is an equivariant operation under $O(3)$: Formally, for any orthogonal transformation $g \in O(3)$ and multivectors $A, B \in \mathbb{G}_{3,0,0}$ we have

$$(gA)(gB) = g(AB). \tag{9}$$

Clifford Group Equivariant Neural Networks (CGENNs). Recently several models based on combining GA with deep learning have been proposed, of which many are based on CGENNs [2]. CGENNs represent neurons in neural networks as multivectors and consist of *linear layers* and *geometric product layers* that operate on the multivector representations.

The linear layers operate independently on each grade of the multivectors using separate learnable transformations, and bias terms are included for the scalar components only:

$$\langle Y_i \rangle_k = \sum_j w_{ij}^{(k)} \langle X_j \rangle_k + \begin{cases} b_i, & k = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Here X_j and Y_i are input and output multivectors respectively, and $w_{ij}^{(k)}$ and b_i are learnable weights and biases. In this work we use a simplified linear layer where weights are shared across all blades, $w_{ij}^{(k)} = w_{ij}$. While both approaches preserve $O(3)$ equivariance, the former allows for greater per-grade expressiveness while the latter reduces the number of learnable parameters and emphasizes the idea of treating the multivector as a unified object rather than a collection of separate grades. In addition a bias term can be included for the trivector component; however, this breaks equivariance with respect to reflection.

The geometric product layers take the geometric product between pairs of multivectors and apply separate learnable weights for a total of 20 weights applied to a combination of 64 interaction pairs. Appendix B shows the derivation of the weighted geometric product between multivectors in $\mathbb{G}_{(3,0,0)}$. Additionally, we refer to [2] for the original derivation of CGENN layers.

Recent work has explored the use of CGENN layers in message passing architectures on graphs by redesigning existing architectures such as EGNN, and testing on n-body simulation tasks [19]. Additional work has used CGENNs to perform message passing on simplicial complexes [20], construct Clifford-steerable kernels for convolutional neural networks [21], and design models for 3D molecular generation [22], and protein structure prediction [23].

3 Method

3.1 Architecture

Our architecture builds on the general framework of message-passing neural networks for molecules, taking PaiNN as a starting point of inspiration. Unlike PaiNN, which employs separate scalar and vector channels, we represent node states as multivectors, providing a unified representation across multiple geometric grades. The update block further introduces a novel update scheme based on successive geometric product layers, adapted from CGENN, to compute residual updates.

Initialization. Given an input graph $G = (V, E)$, we initialize each node i with F multivector channels in $\mathbb{G}_{(3,0,0)}$, which we denote:

$$A_i = \mathbf{s}_i + \vec{\mathbf{v}}_i + \vec{\mathbf{b}}_i + \mathbf{t}_i \in \mathbb{R}^{F \times 8}. \tag{11}$$

The four terms correspond to the scalar, vector, bivector, and trivector grades, respectively. Learned embeddings $a_{z_i}, t_{z_i} \in \mathbb{R}^F$ of the atom type z_i associated with the node are used to initialize the scalar and trivector components while vector and bivector components are initialized as zero:

$$\begin{array}{rcll} \mathbf{s}_i^0 & = & \mathbf{a}_{z_i} & \in \mathbb{R}^F, \\ \vec{\mathbf{v}}_i^0 & = & \vec{\mathbf{0}} & \in \mathbb{R}^{F \times 3}, \\ \vec{\mathbf{b}}_i^0 & = & \vec{\mathbf{0}} & \in \mathbb{R}^{F \times 3}, \\ \mathbf{t}_i^0 & = & \mathbf{t}_{z_i} & \in \mathbb{R}^F. \end{array} \tag{12}$$

Message block. For each message passing round $t \in \{1, \ldots, T\}$, the message block computes and aggregates messages from sender nodes $j \in \mathcal{N}(i)$ to receiver nodes i. Figure 1 shows an overview of the message block architecture. Layer sizes (with feature dimension denoted F) are annotated in gray, and we denote elementwise multiplication by $\circ$. Similar to PaiNN, we apply continuous-filter convolutions from SchNet with an additional cosine cutoff function [24] to the pairwise edge distances:

$$\mathcal{W}(\|\mathbf{r}_{ij}\|) = f_{\text{cut}}(\|\mathbf{r}_{ij}\|) \cdot (\mathbf{W}_\psi \, \boldsymbol{\psi}(\|\mathbf{r}_{ij}\|) + \mathbf{b}_\psi), \tag{13}$$

where ψ denotes RBF expansion and f_{cut} is the cosine cutoff function. The sender node's scalar state $\mathbf{s}_j$ is passed through a two-layer MLP $\phi(\mathbf{s}_j)$, and the transformed scalar features and edge features $\mathcal{W}_{ij}$ are combined via elementwise multiplication and split into 5 gates. One for each multivector component, and an additional gate for incorporating normalized edge vectors into the vector message:

$$\mathbf{g}_{ij} = \phi(\mathbf{s}_j) \circ \mathcal{W}_{ij} = \left[\mathbf{g}_{ij}^{(s)}, \mathbf{g}_{ij}^{(v)}, \mathbf{g}_{ij}^{(d)}, \mathbf{g}_{ij}^{(b)}, \mathbf{g}_{ij}^{(t)} \right]. \tag{14}$$

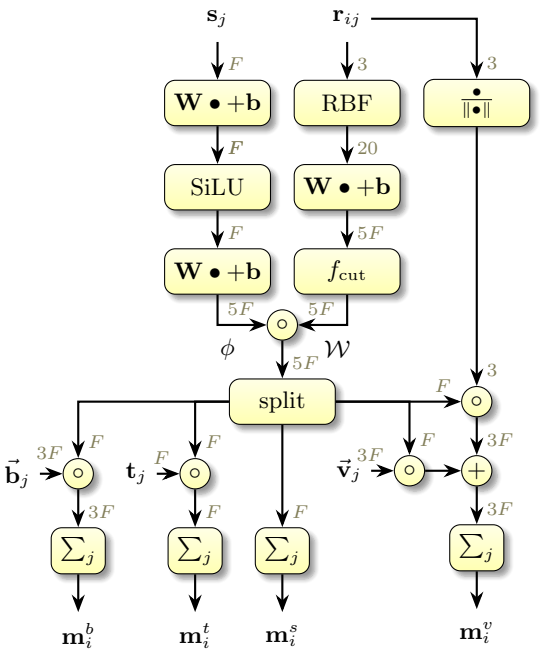

Figure 1. Overview of the message block architecture.

Finally, the messages are computed by aggregating over neighbors in the following way:

$$\mathbf{m}_i^s = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(s)}, \tag{15}$$

$$\mathbf{m}_i^v = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(v)} \circ \vec{\mathbf{v}}_j + \mathbf{g}_{ij}^{(d)} \circ \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|}, \tag{16}$$

$$\mathbf{m}_i^b = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(b)} \circ \vec{\mathbf{b}}_j, \tag{17}$$

$$\mathbf{m}_i^t = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(t)} \circ \mathbf{t}_j. \tag{18}$$

These messages are then added to the corresponding grades of the multivector state for the receiver nodes:

$$\begin{aligned} \mathbf{s}_i &\leftarrow \mathbf{s}_i + \mathbf{m}_i^s, \\ \vec{\mathbf{v}}_i &\leftarrow \vec{\mathbf{v}}_i + \mathbf{m}_i^v, \\ \vec{\mathbf{b}}_i &\leftarrow \vec{\mathbf{b}}_i + \mathbf{m}_i^b, \\ \mathbf{t}_i &\leftarrow \mathbf{t}_i + \mathbf{m}_i^t. \end{aligned} \tag{19}$$

Geometric product layers can also be added to the message block (see Appendix C), but this significantly increases computational cost, scaling with the number of edges rather than nodes, and our experiments with this indicate that the performance of this approach does not justify the overhead.

Update block. The update block processes each node's multivector representation using two linear projections followed by a sequence of geometric product layers and linear layers. Finally, the residual grade-wise update for each multivector is computed by summing over these transformations and modulated by grade-specific gates. Figure 2 shows an overview of the update block architecture. We denote weighted geometric product layers by $\mathrm{GP}(A, B)_w$. We first compute two linear projections of the multivector state A_i: $U_i = \mathbf{U} \cdot A_i$ and $V_i = \mathbf{V} \cdot A_i$. Then, we apply a sequence of weighted geometric products followed by linear layers:

$$Y_{n+1,i} = \mathbf{W}_{n+1} \cdot \mathrm{GP}(X_{n,i}, Y_{n,i})_{w_{n+1}}, \tag{20}$$

for $n = 0, \ldots, N-1$ with $Y_{0,i} = V_i$, and where $X_{n,i}$ is either fixed as U_i, or set to $Y_{n-1,i}$ for $n \geq 2$ to create chained layers. In our main architecture, we use $N = 2$ and keep $X_{n,i} = U_i$, but the formulation supports chaining successive products by setting $X_{n,i} = Y_{n-1,i}$ for $n \geq 2$. To compute grade-specific gates for the residual update, we extract the scalar component from A_i and compute the norm of the vector component from V_i. These are concatenated and passed through atom-type specific two-layer MLPs:

$$\mathbf{a}_i = \mathbf{W}_{2,z_i} \cdot \mathrm{SiLU}\left(\mathbf{W}_{1,z_i}[\mathbf{s}_i, \|\langle V_i \rangle_1\|] + \mathbf{b}_{1,z_i}\right) + \mathbf{b}_{2,z_i}. \tag{21}$$

We split $\mathbf{a}_i \in \mathbb{R}^{F \times 4}$ into four separate gates, and compute residual updates for each grade of the multivector nodes:

$$\Delta\langle A_i \rangle_k = \mathbf{a}_i^{(k)} \circ \left(\langle U_i \rangle_k + \sum_{n=1}^{N} \langle Y_{n,i} \rangle_k \right) \tag{22}$$

Finally, the updated multivector representations for each node is given by adding the residual updates to each grade:

$$\langle A_i \rangle_k \leftarrow \langle A_i \rangle_k + \Delta\langle A_i \rangle_k \tag{23}$$

Readout. The readout layer maps multivector node states to graph-level predictions. We compare two approaches: (1) using PaiNN-style readout networks applied to specific multivector components, and (2) a simplified alternative where relevant components are summed directly across nodes and channels.

In the PaiNN-style setup, scalar properties are predicted by applying an MLP to the scalar part $\mathbf{s}_i$ of each node's multivector state:

$$f(\mathbf{s}_i) = \mathbf{W}_2 \cdot \mathrm{SiLU}(\mathbf{W}_1 \mathbf{s}_i + \mathbf{b}_1) + \mathbf{b}_2, \tag{24}$$

$$\hat{y}_G = \sum_{i \in G} f(\mathbf{s}_i). \tag{25}$$

For the electronic spatial extent $\langle R^2 \rangle$, atom-wise contributions are weighted by squared distances:

$$\hat{y}_G^{\langle R^2 \rangle} = \sum_{i \in G} f(\mathbf{s}_i) \cdot \|\mathbf{x}_i\|^2. \tag{26}$$

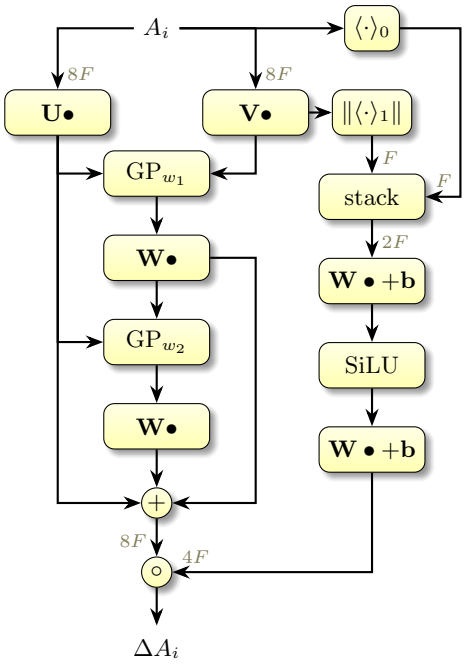

Figure 2. Overview of the update block architecture.

For the dipole moment μ, the prediction is the magnitude of the vector:

$$\vec{\mu} = \sum_{i \in G} \vec{\mu}_{\text{atom}}(\vec{\mathbf{v}}_i) + q_{\text{atom}}(\mathbf{s}_i) \cdot \mathbf{x}_i, \tag{27}$$

where $\vec{\mathbf{v}}_i = \langle A_i \rangle_1$ and $\mathbf{s}_i = \langle A_i \rangle_0$, and both $\vec{\mu}_{\text{atom}}$ and q_{atom} are computed by summing over channels. In the base model, these components are passed through two gated equivariant blocks from PaiNN [1] with atom-type-specific MLPs beforehand.

The simplified readout layer, where we remove the output networks, sums directly across nodes and channels for the relevant grade(s):

$$\hat{y}_G = \sum_{i \in G} \sum_{c=1}^{F} \mathbf{s}_{i,c}, \tag{28}$$

$$\hat{y}_G^{\langle R^2 \rangle} = \sum_{i \in G} \left(\sum_{c=1}^{F} \mathbf{s}_{i,c} \right) \cdot \|\mathbf{x}_i\|^2, \tag{29}$$

$$\vec{\mu} = \sum_{i \in G} \left(\sum_{c=1}^{F} \vec{\mathbf{v}}_{i,c} + \sum_{c=1}^{F} \mathbf{s}_{i,c} \cdot \mathbf{x}_i \right). \tag{30}$$

This approach yields better performance on most targets in our experiments and allows for greater architectural flexibility and generalization as it avoids specialized target-specific readout networks.

3.2 Dataset

We evaluate the proposed architecture on the QM9 dataset [25, 26]. The dataset consists of data for 130,381 small molecules, where atoms can be either carbon, hydrogen, oxygen, nitrogen or fluorine. We use the version of the dataset published by PyTorch Geometric [27, 28]. Preprocessing of the dataset consists of adding edges to the molecular graphs based on a cutoff distance between nodes in 3D space rather than using chemical bonds. We use $r_c = 5.0$ Å. The neighborhood for each node is thus given by:

$$\mathcal{N}(i) = \{ j \in \{1, \ldots, N\} \setminus \{i\} \mid \|\tilde{\mathbf{x}}_j - \tilde{\mathbf{x}}_i\| < r_c \}. \tag{31}$$

Additionally, node coordinates are centralized according to atomic mass. Given a molecule consisting of N atoms, each with atomic number z_i and 3D position $\mathbf{x}_i \in \mathbb{R}^3$ for $i = 1, \ldots, N$, we first compute the center of mass using atomic masses m_i for each atom type:

$$\mathbf{x}_{\text{com}} = \frac{1}{M} \sum_{i=1}^{N} m_i \mathbf{x}_i, \quad \text{where } M = \sum_{i=1}^{N} m_i. \tag{32}$$

All atom positions are then centralized by subtracting the center of mass: $\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}_{\text{com}}$. Finally, for each edge (i, j) in the constructed graph, we compute an edge vector $\mathbf{r}_{ij} = \tilde{\mathbf{x}}_j - \tilde{\mathbf{x}}_i$.

In QM9 each molecule has 19 regression targets. In this work we focus on evaluating the architecture on four of these:

- **Target 0:** Dipole moment (μ)
- **Target 1:** Isotropic polarizability (α)
- **Target 2:** Highest occupied molecular orbital energy (ϵ_{HOMO})
- **Target 5:** Electronic spatial extent ($\langle R^2 \rangle$)

3.3 Training details

We use the same hyperparameters as PaiNN [1]. All experiments use the AdamW optimizer [29] with weight decay $\lambda = 0.01$. We use MSE as the loss function for all targets, except for α which uses MAE loss. If the validation loss plateaus, the learning rate is decayed by a factor of 0.5, with patience 5, and we use early stopping with patience 30. For learning rate decay and early stopping we use exponential smoothing of the validation loss with factor 0.9. Lastly, for the ϵ_{HOMO} and α targets we normalize the target values before training and de-normalize when evaluating on the test set. The full set of hyperparameters are listed in Appendix D. The code for conducting the experiments can be found at Anonymous GitHub Repository.

4 Results

4.1 Architecture selection study

To explore the space of design choices, we first evaluated several architectural variations of the base

Table 2. MAE on 4 QM9 target properties for each addition/ablation compared to the base model. Bold results have the lowest error. Detailed descriptions of each experiment can be found in Appendix E.

	ϵ_{HOMO} meV	μ D	R^2 α_0^2	α α_0^3
Addition				
Sender/receiver GP	26.5374	0.0127	0.1057	0.0525
Sender/copy GP	26.2839	0.0122	0.0843	0.0538
3 GP in update block	24.5228	0.0127	0.0711	0.0525
Grade-wise linear layers	24.7583	0.0130	**0.0661**	0.0526
Ablation				
Removal of second GP	23.4886	NaN	0.0868	**0.0479**
Non-weighted GPs	24.4641	0.0128	0.0806	0.0529
No output networks	28.0492	**0.0119**	0.0697	0.0506
Trivectors initialized as 0	**23.4371**	0.0123	0.0878	0.0508
Shared update MLP	25.1560	0.0130	0.0772	0.0543
Base architecture	24.3626	0.0126	0.0731	0.0532

model (Table 2). To limit the computational cost, these studies were carried out using only $F = 64$ channels. A detailed description of each variation and ablation can be found in Appendix E.

Results show that no single variant performs best across all targets, suggesting that different architectural choices benefit different molecular properties. However, the differences are generally modest, and since each variant was evaluated from a single training run, the results should not be over-interpreted.

Among the message block variants, both the geometric product between sender and receiver pairs and between sender nodes and linear projections of themselves perform worse than the base model on most targets. The latter does slightly improve μ, but worsens all other targets. These results indicate that using the geometric product to mix between grades within a node to update its state is more effective than using it to combine features across neighboring nodes.

Adding a third geometric product in the update block shows mixed results, slightly improving performance on $\langle R^2 \rangle$ and α, but slightly worsening ϵ_{HOMO} and μ. This suggests diminishing returns from stacking additional geometric products in this setting. Conversely, while ablating the second GP layer improves ϵ_{HOMO} and α, it leads to training instability on μ, and worse performance on $\langle R^2 \rangle$.

Replacing the shared linear layers with grade-wise linear layers, while theoretically more expressive, leads to degraded performance on ϵ_{HOMO} and μ, and comes at higher computational cost. For α it yields a slight improvement, and for $\langle R^2 \rangle$ it achieves the best results.

Removing the learnable weights from the geometric product layers leads to a consistent, though modest, degradation in accuracy. This suggests that most of the benefit comes from the structure of the geometric product itself, with the weights serving to refine the computation. Hence, in resource-constrained settings, the weights can possibly be omitted to reduce complexity, with only a small drop in performance.

Most notably, removing the gated equivariant blocks (for μ prediction) and readout MLP (for scalar targets and $\langle R^2 \rangle$) in the output layer improves performance on 3 out of 4 targets. It achieves the lowest error on μ and improves $\langle R^2 \rangle$ and α, but does harm performance on ϵ_{HOMO}.

Initializing trivectors as zero instead of using learned embeddings of the atom type leads to the best overall result for ϵ_{HOMO}, and improves performance slightly on μ and α, but worsens $\langle R^2 \rangle$.

Finally, using a shared update MLP across atom types harms performance across the board, confirming the value of atom-type-specific gates for the residual grade updates.

4.2 Main Evaluation

Based on these findings, we select the best performing architecture for each target and conduct the main evaluation. For ϵ_{HOMO} we initialize the trivector as zero instead of using atom type embeddings, and we only use one geometric product in the update block. For μ and $\langle R^2 \rangle$ we compute the final predictions without output networks (using Eq. 30 and 29, respectively). For $\langle R^2 \rangle$, we additionally test both with and without grade-wise linear layers and find that the effect on performance is minimal (MAE of 0.063 vs. 0.064). For α we also only use one geometric product in the update block. Table 3 compares GA-GNN to state-of-the-art baselines across the selected QM9 targets. Additionally, Figure 3 shows a visual comparison. Results for baselines are from [30, 31], and results for GA-GNN are averaged over 3 random data splits. We increased the feature dimension to $F = 128$, except for $\langle R^2 \rangle$, where $F = 64$ performed best.

The model achieves particularly strong results on dipole moment and electronic spatial extent, ranking third among the compared models. We suspect that GA-GNN performs best on the dipole prediction due to the vector nature of the dipole moment, which aligns well with the multivector representation. Across the four targets, GA-GNN ranks fifth on average, surpassed only by GotenNet [31]. Results for each split are included in Appendix F.

5 Discussion

This work demonstrates that geometric algebra provides a powerful foundation for designing expressive and stable message-passing networks for molecular property prediction. By embedding nodes as multivectors and incorporating weighted geometric products into message passing, GA-GNN achieves strong performance on several QM9 targets. The

Table 3. Average MAE of GA-GNN across three random splits, compared with state-of-the-art models, as reported in the literature, on selected QM9 targets. Models are ordered by their average rank across the targets. The lowest errors are shown in bold, and results within 10% of the best are underlined.

	ϵ_{HOMO} meV	μ D	R^2 α_0^2	α α_0^3	Avg. rank
Cormorant [12]	34	0.038	0.961	0.085	17.50
LieConv [9]	30	0.032	0.800	0.084	16.25
NMP [4]	43	0.030	0.180	0.092	14.75
SchNet [5]	41	0.033	0.073	0.235	14.50
SEGNN [13]	24	0.023	0.660	0.060	12.75
EGNN [10]	29	0.029	0.106	0.071	12.00
MGCN [6]	42	0.056	0.110	**0.030**	11.25
NoisyNodes [32]	20	0.025	0.700	0.052	11.25
DimeNet++ [7]	25	0.030	0.331	0.044	11.00
SphereNet [8]	23	0.026	0.292	0.046	10.50
GNS-TAT+NN [33]	17	0.021	0.650	0.047	9.50
MACE [14]	22	0.015	0.210	0.038	7.75
PaiNN [1]	28	0.012	0.066	0.045	7.50
EQGAT [11]	20	0.011	0.382	0.035	6.25
Equiformer [18]	15	0.011	0.251	0.046	6.00
TorchMD-NET [17]	20	0.011	0.033	0.059	5.75
Equiformer V2 [34]	**14**	0.010	0.186	0.050	5.75
GotenNet B [31]	15	**0.007**	**0.027**	0.032	**1.50**
GA-GNN	21	0.011	0.063	0.045	5.00

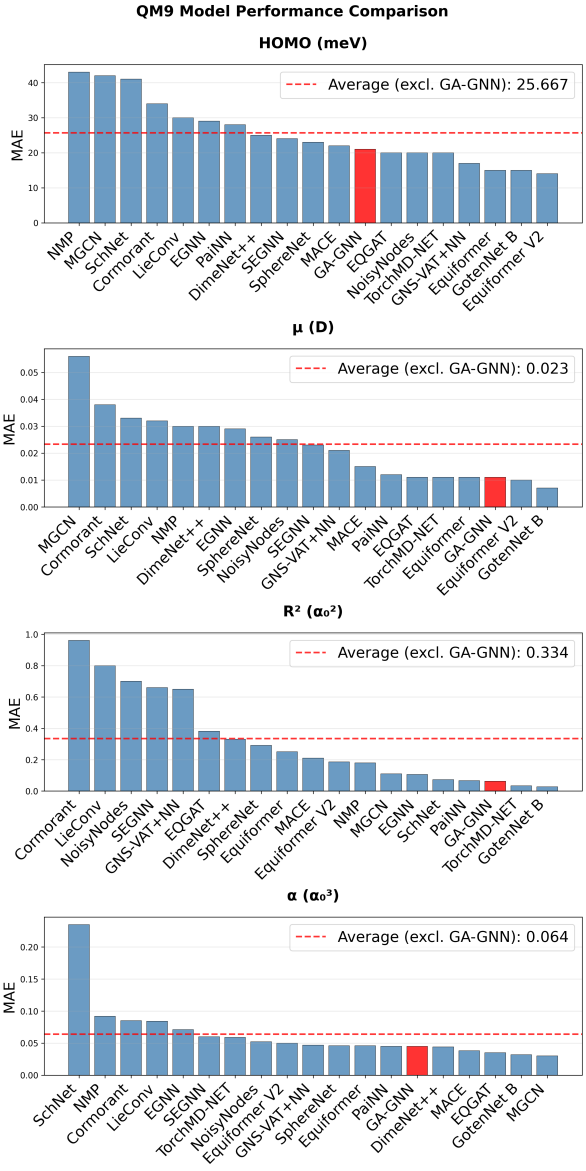

Figure 3. Comparison of models on the QM9 dataset across selected properties.

architecture is flexible: the number of geometric product layers can be scaled up or down depending on the task, and different multivector grades can be utilized for different properties without requiring task-specific readout layers or entirely new architectures. In this way, the combination of multivector embeddings and weighted geometric products enables the model to handle targets of varying geometric nature within a unified framework, requiring only minimal adjustments across targets.

At the same time, our ablation and variation studies show that, despite the flexibility of the multivector framework, architectural choices such as geometric product configurations affect different targets in different ways. Since these results are based on single training runs and the observed differences are relatively modest, they should be interpreted with caution. Nevertheless, they indicate that while GA-GNN offers a unified framework, optimal performance across targets remain sensitive to specific design choices. This variation is consistent with molecular property prediction more broadly, where different architectures tend to excel at capturing different aspects of molecular structure.

In addition to these observations, a practical downside of the approach is the relatively high computational cost and parameter count, mainly due to the weighted geometric products and atom-type–specific update MLPs. The ablation results, however, suggest that lighter variants, for example by removing the geometric product weights, can retain strong performance, providing avenues for reducing complexity.

Overall, our findings highlight both the feasibility and potential of a GA-based approach to designing GNNs for molecular property prediction, while pointing to many opportunities for further exploration. A broader evaluation on all QM9 targets, additional tasks such as n-body simulations [10], or domains with bivector- or trivector-valued outputs, would provide a more complete picture of the model's capabilities. New readout designs that better align with the multivector structure, rather than removing the output network entirely, may also yield improvements. We have also not explored the use of multivector normalization layers from CGENN [2] or hybrid approaches such as scalar/multivector parallel paths from Clifford-EGNNs [19]. Additionally, experimenting with other message block designs such as attention-based architectures (as in EQGAT [11]) may further enrich the framework. Finally, exploring

alternative GA spaces (e.g. $\mathbb{G}_{3,0,1}$), would expand the multivector representation from 8 to 16 components, which could provide additional expressive power and capture richer geometric structures.

6 Conclusion

We introduced GA-GNN, an equivariant graph neural network that extends the message-passing framework to multivector representations and employs geometric product layers from CGENN for structured feature interactions. Evaluated on the QM9 benchmark, GA-GNN achieves competitive performance with recent state-of-the-art models, demonstrating the feasibility and potential of GA-based representations for molecular property prediction. Our experiments highlight effective design choices for incorporating geometric product layers into message passing, as well as the use of shared linear layers. These findings open several directions for future work, including a broader evaluation of the approach and continued exploration through architectural refinements.

References

[1] K. T. Schütt, O. T. Unke, and M. Gastegger. "Equivariant message passing for the prediction of tensorial properties and molecular spectra". In: *CoRR* abs/2102.03150 (2021). arXiv: 2102.03150. URL: https://arxiv.org/abs/2102.03150.

[2] D. Ruhe, J. Brandstetter, and P. Forré. "Clifford Group Equivariant Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine. Vol. 36. Curran Associates, Inc., 2023, pp. 62922–62990. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/c6e0125e14ea3d1a3de3c33fd2d49fc4-Paper-Conference.pdf.

[3] Anonymous. "Master's Thesis. Omitted for double-blind review". MA thesis. 2025.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. "Neural Message Passing for Quantum Chemistry". In: *CoRR* abs/1704.01212 (2017). arXiv: 1704.01212. URL: http://arxiv.org/abs/1704.01212.

[5] K. T. Schütt, P.-J. Kindermans, H. E. Sauceda, S. Chmiela, A. Tkatchenko, and K.-R. Müller. *SchNet: A continuous-filter convolutional neural network for modeling quantum interactions*. 2017. arXiv: 1706.08566 [stat.ML]. URL: https://arxiv.org/abs/1706.08566.

[6] C. Lu, Q. Liu, C. Wang, Z. Huang, P. Lin, and L. He. *Molecular Property Prediction: A Multilevel Quantum Interactions Modeling Perspective*. 2019. arXiv: 1906.11081 [physics.comp-ph]. URL: https://arxiv.org/abs/1906.11081.

[7] J. Klicpera, S. Giri, J. T. Margraf, and S. Günnemann. "Fast and Uncertainty-Aware Directional Message Passing for Non-Equilibrium Molecules". In: *CoRR* abs/2011.14115 (2020). arXiv: 2011.14115. URL: https://arxiv.org/abs/2011.14115.

[8] Y. Liu, L. Wang, M. Liu, X. Zhang, B. Oztekin, and S. Ji. "Spherical Message Passing for 3D Graph Networks". In: *CoRR* abs/2102.05013 (2021). arXiv: 2102.05013. URL: https://arxiv.org/abs/2102.05013.

[9] M. Finzi, S. Stanton, P. Izmailov, and A. G. Wilson. *Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data*. 2020. arXiv: 2002.12880 [stat.ML]. URL: https://arxiv.org/abs/2002.12880.

[10] V. G. Satorras, E. Hoogeboom, and M. Welling. "E(n) Equivariant Graph Neural Networks". In: *CoRR* abs/2102.09844 (2021). arXiv: 2102.09844. URL: https://arxiv.org/abs/2102.09844.

[11] T. Le, F. Noé, and D.-A. Clevert. *Equivariant Graph Attention Networks for Molecular Property Prediction*. 2022. arXiv: 2202.09891 [cs.LG]. URL: https://arxiv.org/abs/2202.09891.

[12] B. Anderson, T.-S. Hy, and R. Kondor. *Cormorant: Covariant Molecular Neural Networks*. 2019. arXiv: 1906.04015 [physics.comp-ph]. URL: https://arxiv.org/abs/1906.04015.

[13] J. Brandstetter, R. Hesselink, E. van der Pol, E. J. Bekkers, and M. Welling. "Geometric and Physical Quantities improve E(3) Equivariant Message Passing". In: *CoRR* abs/2110.02905 (2021). arXiv: 2110.02905. URL: https://arxiv.org/abs/2110.02905.

[14] I. Batatia, D. P. Kovács, G. N. C. Simm, C. Ortner, and G. Csányi. *MACE: Higher Order Equivariant Message Passing Neural Networks for Fast and Accurate Force Fields*. 2023. arXiv: 2206.07697 [stat.ML]. URL: https://arxiv.org/abs/2206.07697.

[15] G. Simeon and G. de Fabritiis. *TensorNet: Cartesian Tensor Representations for Efficient Learning of Molecular Potentials*. 2023. arXiv: 2306.06482 [cs.LG]. URL: https://arxiv.org/abs/2306.06482.

[16] A. Musaelian, S. Batzner, A. Johansson, L. Sun, C. J. Owen, M. Kornbluth, and B. Kozinsky. *Learning Local Equivariant Representations for Large-Scale Atomistic Dynamics*. 2022. arXiv: 2204.05249 [physics.comp-ph]. URL: https://arxiv.org/abs/2204.05249.

[17] P. Thölke and G. D. Fabritiis. "TorchMD-NET: Equivariant Transformers for Neural Network based Molecular Potentials". In: *CoRR* abs/2202.02541 (2022). arXiv: 2202.02541. URL: https://arxiv.org/abs/2202.02541.

[18] Y.-L. Liao and T. Smidt. *Equiformer: Equivariant Graph Attention Transformer for 3D Atomistic Graphs*. 2023. arXiv: 2206.11990 [cs.LG]. URL: https://arxiv.org/abs/2206.11990.

[19] C. Liu, D. Ruhe, and P. Forré. "Multivector Neurons: Better and Faster O(n)-Equivariant Clifford GNNs". In: *ICML 2024 Workshop on Geometry-grounded Representation Learning and Generative Modeling*. 2024. URL: https://openreview.net/forum?id=F03nrnuGuj.

[20] C. Liu, D. Ruhe, F. Eijkelboom, and P. Forré. *Clifford Group Equivariant Simplicial Message Passing Networks*. 2024. arXiv: 2402.10011 [cs.AI]. URL: https://arxiv.org/abs/2402.10011.

[21] M. Zhdanov, D. Ruhe, M. Weiler, A. Lucic, J. Brandstetter, and P. Forré. *Clifford-Steerable Convolutional Neural Networks*. 2024. arXiv: 2402.14730 [cs.LG]. URL: https://arxiv.org/abs/2402.14730.

[22] C. Liu, S. Vadgama, D. Ruhe, E. Bekkers, and P. Forré. *Clifford Group Equivariant Diffusion Models for 3D Molecular Generation*. 2025. arXiv: 2504.15773 [cs.LG]. URL: https://arxiv.org/abs/2504.15773.

[23] A. Pepe, S. Buchholz, and J. Lasenby. "Clifford Group Equivariant Neural Network Layers for Protein Structure Prediction". In: *Northern Lights Deep Learning Conference 2024*. 2024. URL: https://openreview.net/forum?id=JNfpsiGS5E.

[24] J. Behler. "Atom-centered symmetry functions for constructing high-dimensional neural network potentials". In: *The Journal of Chemical Physics* 134.7 (Feb. 2011), p. 074106. ISSN: 0021-9606. DOI: 10.1063/1.3553717. eprint: https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/1.3553717/15435271/074106_1_online.pdf. URL: https://doi.org/10.1063/1.3553717.

[25] L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond. "Enumeration of 166 Billion Organic Small Molecules in the Chemical Universe Database GDB-17". In: *Journal of Chemical Information and Modeling* 52.11 (2012), pp. 2864–2875. DOI: 10.1021/ci300415d.

[26] R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld. "Quantum chemistry structures and properties of 134 kilo molecules". In: *Scientific Data* 1 (2014).

[27] M. Fey and J. E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *Proceedings of the ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019. URL: https://arxiv.org/abs/1903.02428.

[28] PyTorch Geometric Developers. *torch_geometric.datasets.QM9 – PyTorch Geometric documentation*. https://pytorch-geometric.readthedocs.io/en/2.6.1/generated/torch_geometric.datasets.QM9.html. Accessed: 2025-03-25. 2025.

[29] I. Loshchilov and F. Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[30] D. P. Kovács, I. Batatia, E. S. Arany, and G. Csányi. "Evaluation of the MACE force field architecture: From medicinal chemistry to materials science". In: *The Journal of Chemical Physics* 159.4 (July 2023). ISSN: 1089-7690. DOI: 10.1063/5.0155322. URL: http://dx.doi.org/10.1063/5.0155322.

[31] S. Aykent and T. Xia. "GotenNet: Rethinking Efficient 3D Equivariant Graph Neural Networks". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=5wxCQDtbMo.

[32] J. Godwin, M. Schaarschmidt, A. Gaunt, A. Sanchez-Gonzalez, Y. Rubanova, P. Veličković, J. Kirkpatrick, and P. Battaglia. *Simple GNN Regularisation for 3D Molecular Property Prediction & Beyond*. 2022. arXiv: 2106.07971 [cs.LG]. URL: https://arxiv.org/abs/2106.07971.

[33] S. Zaidi, M. Schaarschmidt, J. Martens, H. Kim, Y. W. Teh, A. Sanchez-Gonzalez, P. Battaglia, R. Pascanu, and J. Godwin. *Pre-training via Denoising for Molecular Property Prediction*. 2022. arXiv: 2206.00133 [cs.LG]. URL: https://arxiv.org/abs/2206.00133.

[34] Y.-L. Liao, B. Wood, A. Das, and T. Smidt. "EquiformerV2: Improved Equivariant Transformer for Scaling to Higher-Degree Representations". In: *The Thirteenth International Conference on Learning Representations*. 2024. DOI: 10.48550/arXiv.2306.12059.

[35] J. Vince. *Geometric Algebra for Computer Graphics*. Springer-Verlag London Limited 2008, 2009.

Table A.1. Geometric product table for the basis blades in $\mathbb{G}_{3,0,0}$.

GP	λ	e_1	e_2	e_3	e_{12}	e_{23}	e_{31}	e_{123}
λ	λ^2	λe_1	λe_2	λe_3	λe_{12}	λe_{23}	λe_{31}	λe_{123}
e_1	λe_1	1	e_{12}	$-e_{31}$	e_2	e_{123}	$-e_3$	e_{23}
e_2	λe_2	$-e_{12}$	1	e_{23}	$-e_1$	e_3	e_{123}	e_{31}
e_3	λe_3	e_{31}	$-e_{23}$	1	e_{123}	$-e_2$	e_1	e_{12}
e_{12}	λe_{12}	$-e_2$	e_1	e_{123}	-1	$-e_{31}$	e_{23}	$-e_3$
e_{23}	λe_{23}	e_{123}	$-e_3$	e_2	e_{31}	-1	$-e_{12}$	$-e_1$
e_{31}	λe_{31}	e_3	e_{123}	$-e_1$	$-e_{23}$	e_{12}	-1	$-e_2$
e_{123}	λe_{123}	e_{23}	e_{31}	e_{12}	$-e_3$	$-e_1$	$-e_2$	-1

A Geometric product in $\mathbb{G}_{(3,0,0)}$

The geometric product is defined between all basis blades. Table A.1 from [35] shows the product between all pairs of basis blades in $\mathbb{G}_{3,0,0}$. Given two multivectors A and B in $\mathbb{G}_{(3,0,0)}$:

$$A = \lambda_0 + \lambda_1 e_1 + \lambda_2 e_2 + \lambda_3 e_3 + \lambda_4 e_{12} + \lambda_5 e_{23} + \lambda_6 e_{31} + \lambda_7 e_{123} \tag{33}$$

$$B = \beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123} \tag{34}$$

We can write the product AB as:

$$\begin{aligned} AB = {} & \lambda_0(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \\ & + \lambda_1 e_1(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \\ & + \lambda_2 e_2(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \\ & + \lambda_3 e_3(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \\ & + \lambda_4 e_{12}(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \\ & + \lambda_5 e_{23}(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \\ & + \lambda_6 e_{31}(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \\ & + \lambda_7 e_{123}(\beta_0 + \beta_1 e_1 + \beta_2 e_2 + \beta_3 e_3 + \beta_4 e_{12} + \beta_5 e_{23} + \beta_6 e_{31} + \beta_7 e_{123}) \end{aligned} \tag{35}$$

Expanding all products explicitly, using table A.1 gives:

$$\begin{aligned} AB = {} & \lambda_0\beta_0 + \lambda_0\beta_1 e_1 + \lambda_0\beta_2 e_2 + \lambda_0\beta_3 e_3 + \lambda_0\beta_4 e_{12} + \lambda_0\beta_5 e_{23} + \lambda_0\beta_6 e_{31} + \lambda_0\beta_7 e_{123} \\ & + \lambda_1\beta_0 e_1 + \lambda_1\beta_1 + \lambda_1\beta_2 e_{12} - \lambda_1\beta_3 e_{31} + \lambda_1\beta_4 e_2 + \lambda_1\beta_5 e_{123} - \lambda_1\beta_6 e_3 + \lambda_1\beta_7 e_{23} \\ & + \lambda_2\beta_0 e_2 - \lambda_2\beta_1 e_{12} + \lambda_2\beta_2 + \lambda_2\beta_3 e_{23} - \lambda_2\beta_4 e_1 + \lambda_2\beta_5 e_3 + \lambda_2\beta_6 e_{123} + \lambda_2\beta_7 e_{31} \\ & + \lambda_3\beta_0 e_3 + \lambda_3\beta_1 e_{31} - \lambda_3\beta_2 e_{23} + \lambda_3\beta_3 + \lambda_3\beta_4 e_{123} - \lambda_3\beta_5 e_2 + \lambda_3\beta_6 e_1 + \lambda_3\beta_7 e_{12} \\ & + \lambda_4\beta_0 e_{12} - \lambda_4\beta_1 e_2 + \lambda_4\beta_2 e_1 + \lambda_4\beta_3 e_{123} - \lambda_4\beta_4 - \lambda_4\beta_5 e_{31} + \lambda_4\beta_6 e_{23} - \lambda_4\beta_7 e_3 \\ & + \lambda_5\beta_0 e_{23} + \lambda_5\beta_1 e_{123} - \lambda_5\beta_2 e_3 + \lambda_5\beta_3 e_2 + \lambda_5\beta_4 e_{31} - \lambda_5\beta_5 - \lambda_5\beta_6 e_{12} - \lambda_5\beta_7 e_1 \\ & + \lambda_6\beta_0 e_{31} + \lambda_6\beta_1 e_3 + \lambda_6\beta_2 e_{123} - \lambda_6\beta_3 e_1 - \lambda_6\beta_4 e_{23} + \lambda_6\beta_5 e_{12} - \lambda_6\beta_6 - \lambda_6\beta_7 e_2 \\ & + \lambda_7\beta_0 e_{123} + \lambda_7\beta_1 e_{23} + \lambda_7\beta_2 e_{31} + \lambda_7\beta_3 e_{12} - \lambda_7\beta_4 e_3 - \lambda_7\beta_5 e_1 - \lambda_7\beta_6 e_2 - \lambda_7\beta_7 \end{aligned} \tag{36}$$

Finally, we collect like terms by grade:

$$\begin{aligned} AB = {} & \lambda_0\beta_0 + \lambda_1\beta_1 + \lambda_2\beta_2 + \lambda_3\beta_3 - \lambda_4\beta_4 - \lambda_5\beta_5 - \lambda_6\beta_6 - \lambda_7\beta_7 \\ & + (\lambda_0\beta_1 + \lambda_1\beta_0 - \lambda_2\beta_4 + \lambda_3\beta_6 + \lambda_4\beta_2 - \lambda_5\beta_7 - \lambda_6\beta_3 - \lambda_7\beta_5)e_1 \\ & + (\lambda_0\beta_2 + \lambda_1\beta_4 + \lambda_2\beta_0 - \lambda_3\beta_5 - \lambda_4\beta_1 + \lambda_5\beta_3 - \lambda_6\beta_7 - \lambda_7\beta_6)e_2 \\ & + (\lambda_0\beta_3 - \lambda_1\beta_6 + \lambda_2\beta_5 + \lambda_3\beta_0 - \lambda_4\beta_7 - \lambda_5\beta_2 + \lambda_6\beta_1 - \lambda_7\beta_4)e_3 \\ & + (\lambda_0\beta_4 + \lambda_1\beta_2 - \lambda_2\beta_1 + \lambda_3\beta_7 + \lambda_4\beta_0 - \lambda_5\beta_6 + \lambda_6\beta_5 + \lambda_7\beta_3)e_{12} \\ & + (\lambda_0\beta_5 + \lambda_1\beta_7 + \lambda_2\beta_3 - \lambda_3\beta_2 + \lambda_4\beta_6 + \lambda_5\beta_0 - \lambda_6\beta_4 + \lambda_7\beta_1)e_{23} \\ & + (\lambda_0\beta_6 - \lambda_1\beta_3 + \lambda_2\beta_7 + \lambda_3\beta_1 - \lambda_4\beta_5 + \lambda_5\beta_4 + \lambda_6\beta_0 + \lambda_7\beta_2)e_{31} \\ & + (\lambda_0\beta_7 + \lambda_1\beta_5 + \lambda_2\beta_6 + \lambda_3\beta_4 + \lambda_4\beta_3 + \lambda_5\beta_1 + \lambda_6\beta_2 + \lambda_7\beta_0)e_{123} \end{aligned} \tag{37}$$

Which produces a new multivector with coefficients for each blade defined as above.

B Weighted geometric product in $\mathbb{G}_{(3,0,0)}$

The geometric product between multivectors in $\mathbb{G}_{(3,0,0)}$ consists of 64 interaction pairs that can be grouped into 20 different interaction types, i.e. (scalar/scalar), (scalar/vector), (vector/bivector), etc. For example for the grade-0 (scalar) component of the geometric product between multivectors A and B, we can write:

$$\begin{aligned}\langle AB\rangle_0 = {} & \lambda_0\beta_0 && \text{(scalar} \cdot \text{scalar)}\\ & + \lambda_1\beta_1 + \lambda_2\beta_2 + \lambda_3\beta_3 && \text{(vector} \cdot \text{vector)}\\ & - \lambda_4\beta_4 - \lambda_5\beta_5 - \lambda_6\beta_6 && \text{(bivector} \cdot \text{bivector)}\\ & - \lambda_7\beta_7 && \text{(trivector} \cdot \text{trivector)}\end{aligned} \tag{38}$$

Similarly, for the grade-1 (vector), grade-2 (bivector), and grade-3 (trivector) components:

$$\begin{aligned}\langle AB\rangle_1 = {} & \lambda_0\beta_1 e_1 + \lambda_0\beta_2 e_2 + \lambda_0\beta_3 e_3 && \text{(scalar} \cdot \text{vector)}\\ & + \lambda_1\beta_0 e_1 + \lambda_2\beta_0 e_2 + \lambda_3\beta_0 e_3 && \text{(vector} \cdot \text{scalar)}\\ & + \lambda_1\beta_4 e_2 - \lambda_1\beta_6 e_3 - \lambda_2\beta_4 e_1 + \lambda_2\beta_5 e_3 + \lambda_3\beta_6 e_1 - \lambda_3\beta_5 e_2 && \text{(vector} \cdot \text{bivector)}\\ & + \lambda_4\beta_2 e_1 - \lambda_4\beta_1 e_2 + \lambda_5\beta_3 e_2 - \lambda_5\beta_2 e_3 - \lambda_6\beta_3 e_1 + \lambda_6\beta_1 e_3 && \text{(bivector} \cdot \text{vector)}\\ & - \lambda_4\beta_7 e_3 - \lambda_5\beta_7 e_1 - \lambda_6\beta_7 e_2 && \text{(bivector} \cdot \text{trivector)}\\ & - \lambda_7\beta_5 e_1 - \lambda_7\beta_6 e_2 - \lambda_7\beta_4 e_3 && \text{(trivector} \cdot \text{bivector)}\end{aligned} \tag{39}$$

$$\begin{aligned}\langle AB\rangle_2 = {} & \lambda_0\beta_4 e_{12} + \lambda_0\beta_5 e_{23} + \lambda_0\beta_6 e_{31} && \text{(scalar} \cdot \text{bivector)}\\ & + \lambda_4\beta_0 e_{12} + \lambda_5\beta_0 e_{23} + \lambda_6\beta_0 e_{31} && \text{(bivector} \cdot \text{scalar)}\\ & + \lambda_1\beta_2 e_{12} - \lambda_1\beta_3 e_{31} - \lambda_2\beta_1 e_{12} + \lambda_2\beta_3 e_{23} - \lambda_3\beta_2 e_{23} + \lambda_3\beta_1 e_{31} && \text{(vector} \cdot \text{vector)}\\ & + \lambda_4\beta_6 e_{23} - \lambda_4\beta_5 e_{31} - \lambda_5\beta_6 e_{12} + \lambda_5\beta_4 e_{31} + \lambda_6\beta_5 e_{12} - \lambda_6\beta_4 e_{23} && \text{(bivector} \cdot \text{bivector)}\\ & + \lambda_1\beta_7 e_{23} + \lambda_2\beta_7 e_{31} + \lambda_3\beta_7 e_{12} && \text{(vector} \cdot \text{trivector)}\\ & + \lambda_7\beta_1 e_{23} + \lambda_7\beta_2 e_{31} + \lambda_7\beta_3 e_{12} && \text{(trivector} \cdot \text{vector)}\end{aligned} \tag{40}$$

$$\begin{aligned}\langle AB\rangle_3 = {} & \lambda_0\beta_7 e_{123} && \text{(scalar} \cdot \text{trivector)}\\ & + \lambda_7\beta_0 e_{123} && \text{(trivector} \cdot \text{scalar)}\\ & + (\lambda_1\beta_5 + \lambda_2\beta_6 + \lambda_3\beta_4) e_{123} && \text{(vector} \cdot \text{bivector)}\\ & + (\lambda_4\beta_3 + \lambda_5\beta_1 + \lambda_6\beta_2) e_{123} && \text{(bivector} \cdot \text{vector)}\end{aligned} \tag{41}$$

If we apply separate weights to each of these 20 interaction terms and collect terms with the same weights, we can derive a weighted geometric product that is equivariant to $O(3)$:

$$\begin{aligned}&\mathrm{GP}(A,B)_w = \\ &w_s\lambda_0\beta_0 + w_v(\lambda_1\beta_1 + \lambda_2\beta_2 + \lambda_3\beta_3) - w_b(\lambda_4\beta_4 + \lambda_5\beta_5 + \lambda_6\beta_6) - w_t\lambda_7\beta_7 \\ &+ \big(w_{sv}\lambda_0\beta_1 + w_{vs}\lambda_1\beta_0 + w_{vb}(-\lambda_2\beta_4 + \lambda_3\beta_6) + w_{bv}(\lambda_4\beta_2 - \lambda_6\beta_3) - w_{bt}\lambda_5\beta_7 - w_{tb}\lambda_7\beta_5\big)e_1 \\ &+ \big(w_{sv}\lambda_0\beta_2 + w_{vs}\lambda_2\beta_0 + w_{vb}(\lambda_1\beta_4 - \lambda_3\beta_5) + w_{bv}(-\lambda_4\beta_1 + \lambda_5\beta_3) - w_{bt}\lambda_6\beta_7 - w_{tb}\lambda_7\beta_6\big)e_2 \\ &+ \big(w_{sv}\lambda_0\beta_3 + w_{vs}\lambda_3\beta_0 + w_{vb}(-\lambda_1\beta_6 + \lambda_2\beta_5) + w_{bv}(-\lambda_5\beta_2 - \lambda_6\beta_1) - w_{bt}\lambda_4\beta_7 - w_{tb}\lambda_7\beta_4\big)e_3 \\ &+ \big(w_{sb}\lambda_0\beta_4 + w_{bs}\lambda_4\beta_0 + w_{vv}(\lambda_1\beta_2 - \lambda_2\beta_1) + w_{bb}(-\lambda_5\beta_6 + \lambda_6\beta_5) + w_{vt}\lambda_3\beta_7 + w_{tv}\lambda_7\beta_3\big)e_{12} \\ &+ \big(w_{sb}\lambda_0\beta_5 + w_{bs}\lambda_5\beta_0 + w_{vv}(\lambda_2\beta_3 - \lambda_3\beta_2) + w_{bb}(\lambda_4\beta_6 - \lambda_6\beta_4) + w_{vt}\lambda_1\beta_7 + w_{tv}\lambda_7\beta_1\big)e_{23} \\ &+ \big(w_{sb}\lambda_0\beta_6 + w_{bs}\lambda_6\beta_0 + w_{vv}(-\lambda_1\beta_3 + \lambda_3\beta_1) + w_{bb}(-\lambda_4\beta_5 + \lambda_5\beta_4) + w_{vt}\lambda_2\beta_7 + w_{tv}\lambda_7\beta_2\big)e_{31} \\ &+ \big(w_{st}\lambda_0\beta_7 + w_{tvb}(\lambda_1\beta_5 + \lambda_2\beta_6 + \lambda_3\beta_4) + w_{tbv}(\lambda_4\beta_3 + \lambda_5\beta_1 + \lambda_6\beta_2) + w_{ts}\lambda_7\beta_0\big)e_{123}\end{aligned} \tag{42}$$

The weights can be initialized in many different ways. In our experiments, we chose to initialize them using a normal distribution with zero mean and standard deviation $1/\sqrt{8}$. Additionally, one could add a bias term to the scalar component. Adding bias to the trivector component maintains rotation equivariance, but breaks equivariance with respect to reflection.

Table D.1. Hyperparameters used for training.

Hyperparameter	Value
`batch_size`	100
`learning rate` (initial)	5e-4
`minimum learning rate`	1e-6
`weight decay`	0.01
`patience` (lr decay)	5
`patience` (early stopping)	30
`alpha` (EMA smoothing)	0.9
`train/val/test split sizes`	[110000, 10000, 10831]
`Global seed`	0
`Data split seed (ablation/variation)`	0
`Data split seeds (main evaluation)`	[1, 2, 3]

C Including the geometric product in the message block

We test two variations of an alternative message block that utilizes the weighted geometric product. Here, we describe the variation between sender and receiver pairs. For each edge (i, j) we compute the weighted geometric product:

$$A_{ij} = \mathrm{GP}(A_i, A_j)_w \tag{43}$$

where A_i and A_j are the sender and receiver multivector states. The resulting multivector A_{ij} is then passed through a linear layer:

$$A'_{ij} = \mathbf{W} \cdot A_{ij} \tag{44}$$

The grades of this transformed multivector are then gated and aggregated analogously to the base architecture formulation instead of using the grades of the sender multivector. One exception is that the scalar message also uses the multivector grade:

$$\mathbf{m}_i^s = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(s)} \circ \langle A'_{ij} \rangle_0 \tag{45}$$

$$\mathbf{m}_i^v = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(v)} \circ \langle A'_{ij} \rangle_1 + \mathbf{g}_{ij}^{(d)} \circ \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|} \tag{46}$$

$$\mathbf{m}_i^b = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(b)} \circ \langle A'_{ij} \rangle_2 \tag{47}$$

$$\mathbf{m}_i^t = \sum_{j \in \mathcal{N}(i)} \mathbf{g}_{ij}^{(t)} \circ \langle A'_{ij} \rangle_3 \tag{48}$$

The variation that takes the weighted geometric product between the sender state and a linear projection of itself uses the same formulation as above, but with the weighted geometric product input changed accordingly.

D Hyperparameters for training

The same hyperparameters are used across all experiments, with the exception that ablation and architecture variation studies use a channel dimension of $F = 64$, whereas the main evaluation uses $F = 128$ for all targets except for $\langle R^2 \rangle$, which still uses $F = 64$. All experiments are trained with $T = 4$ message passing rounds.

Two types of random seeds are used in the experiments. A global random seed is fixed to zero in all runs to ensure reproducibility of stochastic elements such as model weight initialization and any other random operations during training. In addition, a data split seed controls the shuffling used to generate the train/validation/test splits (with fixed split sizes but different assignments). For ablation and variation experiments, only split seed 0 is used. For the main evaluation, results are averaged over split seeds 1, 2, and 3, corresponding to three distinct data splits. The set of hyperparameters is listed in Table D.1.

Table E.1. Overview and description of each architectural addition and ablation.

Additions	**Description**
Sender/receiver GP	Using the geometric product between sender and receiver nodes in the message block as described in Appendix C.
Sender/copy GP	Using the geometric product between sender nodes and a linear projection of themselves in the message block as described in Appendix C.
3 GP in update block	Setting $N = 3$ in Eq. 20 and Eq. 22 to extend the update block with three successive geometric product layers instead of two. And we set $X_{2,i} = Y_{1,i}$ for the third geometric product.
Grade-wise linear layers	Replace all shared linear layers with grade-wise linear layers, as described in Eq. 10.
Ablations	
Removal of second GP	Remove the second geometric product layer from the update block, i.e. set $N = 1$ in Eq. 20 and Eq. 22.
Non-weighted GPs	Remove learnable scalar weights from the geometric product operation, making the product unweighted and implemented as in Appendix A.
No output networks	Removal of the gated equivariant blocks for dipole moment (μ) prediction, and two-layer MLP's for scalar and ($\langle R^2 \rangle$) prediction. As described in section 3.1.
Trivectors initialized as 0	Initialize the trivector component of the node state to zero instead of using learned embeddings of the atom type.
Shared update MLP	Use a single MLP shared across all atom types, rather than atom-type specific MLP's in the update block to compute residual update gates (i.e. removing z_i index in Eq. 21.
Base architecture	The default architecture as described in Section 3.1.

E Descriptions of each architecture addition/ablation.

Table E.1 describes each addition/ablation for the architecture variation and ablation study in detail.

F Results per individual seed/split

Table F.1 shows the MAE of GA-GNN on each random data split across the 4 QM9 targets as well as the mean and standard deviation.

Table F.1. MAE of GA-GNN on each random split with mean and standard deviation.

Target	**Split 1**	**Split 2**	**Split 3**	**Mean ± Std**
ϵ_{HOMO}	20.3650	21.3509	20.4838	20.7332 ± 0.5382
μ	0.0109	0.0108	0.0109	0.0109 ± 0.0001
$\langle R^2 \rangle$ (grade-wise linear layers)	0.0585	0.0676	0.0633	0.0631 ± 0.0046
$\langle R^2 \rangle$ (shared linear layers)	0.0633	0.0622	0.0677	0.0644 ± 0.0029
α	0.0461	0.0462	0.0439	0.0454 ± 0.0013