Recovering Mesh Centroids from LiDAR Point Clouds using SE(3)-Equivariant Graph Networks

Benjamin Banks (s234802)
Jonathan Tybirk (s216136)
Lucas Rieneck Gottfried Pedersen (s234842)

Technical University of Denmark
BSc Artificial Intelligence and Data

June 2025

Abstract

This paper evaluates $SE(3)$-equivariant graph neural networks (EGNNs) for estimating mesh centroids from sparse LiDAR point clouds. Using spherical harmonics and Clebsch-Gordan algebra we construct a strictly rotation- and translation-equivariant message-passing network and compare it with size-matched non-equivariant GNNs, one of which is trained with random rotational augmentation. On 11 590 ShapeNet objects (800 train, 200 validation, 10.590 test) the EGNN attains the best prediction error (0.0703) while preserving 0 equivariance error; the augmented model reaches similar accuracy (0.0706) but retains a small residual equivariance error (0.0027). The EGNN incurs a six-fold inference-time penalty due to sparse Clebsch-Gordan tensor contractions. The results suggest that when ample augmented data are available, architectural equivariance is not strictly required for high accuracy, yet it delivers exact symmetry, greater training stability, and potential data-efficiency benefits that are valuable in low-data or safety-critical scenarios.

Contents

1 Introduction

In recent years, deep learning has revolutionized numerous domains by letting machines learn complex patterns directly from data. When dealing with physical systems that obey fundamental conservation laws or exhibit geometric symmetries, traditional neural network architectures must learn these inherent properties themselves from large amounts of training data. This data-driven approach can be inefficient, as the networks need to independently discover principles that are already well-understood in physics and mathematics.

This report functions as an exploration of the use of symmetry-aware neural networks, namely *rotation and translation equivariant message passing graph neural networks (SE(3) EGNNs)*, for highly stochastic *point cloud tasks*, such as those generated by LiDAR. In these tasks, some information of the true geometry has been lost and has to be inferred by the model, an underdeveloped use of EGNNs in the literature[1]. Specifically, this report focuses on the task of estimating surface mesh centroids when the true mesh is unknown and only a sparse point cloud, generated by a LiDAR-like process, is given. Since centroid calculations inherently respect translational and rotational symmetries, this problem serves as an ideal test case for equivariant architectures in this field.

EGNNs are based on group equivariance, a mathematical property describing mappings where transformations of the input result in corresponding transformations of the output, and offer a principled approach to embedding geometric and physical symmetries directly into neural network architectures. See figure 1 for reference. Recent advances have demonstrated the effectiveness of SE(3)-equivariant models in areas such as point cloud registration and representation learning, notably in tasks such as relative pose estimation from sparse LiDAR scans [1, 2]. However, to our knowledge, regression of global, frame-dependent properties, such as the centroid of an unknown surface mesh, from a single sparse point cloud has not been directly addressed in prior work. This report therefore expands the scope of equivariant GNNs to a new and underutilized setting: high-stochasticity, single-frame geometry inference. We argue that centroid estimation under SE(3) symmetry is a natural and rigorous test case for evaluating the expressiveness and efficiency of equivariant message-passing networks in estimating global geometric parameters from noisy 3D data, and our findings provide an indicator of their suitability for such tasks.

In this report, we:

- Develop the theoretical framework of Brandstetter et al. [3] for understanding group equivariance in the context of Graph Neural Networks,
- Implement novel equivariant GNN architectures specifically designed for mesh centroid estimation from point cloud data,
- Evaluate these architectures against equivalent traditional non-equivariant neural networks, both with and without data augmentation to quantify the benefits of incorporating equivariance,
- Focus specifically on limited-data scenarios where the theoretical advantages of equivariant weight sharing are expected to be most pronounced,
- Analyze the trade-offs between computational resources (inference cost) and physical accuracy.

[1] See next paragraph

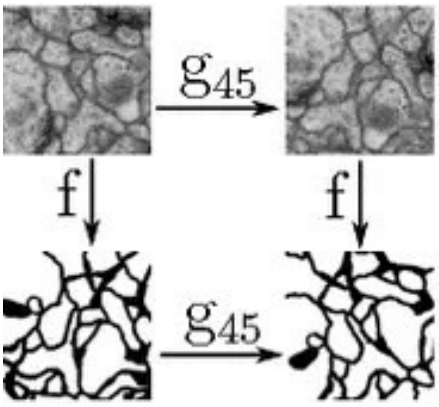

Figure 1: Equivariance shown visually. The input in the top left is transformed by a map f, such as a neural network becoming the image in the bottom left. If the input is rotated before being fed to f however, the output corresponds to the unrotated transformation rotated. If not equivariant, it would have led to differing images in the bottom right. Image adapted from https://blog.csdn.net/qq_25819827/article/details/78515157.

2 Theory

In this section the underlying theoretical framework of SE(3) equivariant neural networks as presented in Brandstetter et al. [3] is introduced. The framework ensures rotation and translation equivariance. First, basic group theory is explained, including the formal definition of equivariance. Then the spherical basis and Clebtsch-Gordan product are introduced, and finally we present message-passing graph neural networks (GNN) and how they become equivariant. The goal of this section is that the reader feels comfortable with the theory and the choices taken in developing this framework, as well as providing references to current developments in the field.

2.1 Basic group theory

Some group theory is needed to understand the math and notation of equivariance. First, groups and representations are introduced.

2.1.1 Groups

A group, $(G, \cdot)$, is defined as a non-empty set, a collection of mathematical objects, equipped with a defined *group product*, $\cdot$, that works between two elements, g_1 and g_2, of the set: $g_1 \cdot g_2$. This product could be any binary operation, but must fulfill 4 group axioms:

1. **Closure:** For any two elements $g_1, g_2 \in G$, the product $g_1 \cdot g_2$ is also in G.
2. **Associativity:** For any three elements $g_1, g_2, g_3 \in G$, the equation $(g_1 \cdot g_2) \cdot g_3 = g_1 \cdot (g_2 \cdot g_3)$ holds.
3. **Identity:** There exists an identity element $e \in G$ such that for all $g \in G$, $e \cdot g = g \cdot e = g$.
4. **Inverse:** For every $g \in G$, there exists an inverse element $g^{-1} \in G$ such that $g \cdot g^{-1} = g^{-1} \cdot g = e$.

Groups SO(3) and SE(3)

To make the abstract group definition more concrete, we consider the groups most relevant for this work: SO(3) and SE(3). These are the groups of 3D rotations and rigid-body transformations, respectively.

The special orthogonal group in 3 dimensions **SO(3)** is a continuous group that describes rotations in 3d space. Group elements can be described by orthogonal 3×3 matrices with determinant 1[2]. This is equivalent to rotation with Euler angles: α, β, and γ, and has 3 degrees of freedom. The group SO(3) satisfies the four group axioms:

1. **Closure:** If one first rotates a point using a rotation matrix R_1, and then applies another rotation R_2, the combined operation R_2R_1 is also a valid rotation, i.e., still in SO(3).
2. **Associativity:** Matrix multiplication is associative, so for any three rotations $R_1, R_2, R_3 \in \mathrm{SO}(3)$, we have $(R_1R_2)R_3 = R_1(R_2R_3)$.
3. **Identity:** The identity rotation is given by the identity matrix I, which leaves any other rotation unchanged: $IR = R$.
4. **Inverse:** Every rotation matrix R has an inverse $R^{-1} = R^\top$, which undoes the rotation.

Note that a SO(3) rotation cannot be represented by a single 3D vector. For example, a rotation around the z-axis leaves a vector pointing along z unchanged, though it will affect other vectors. Thus rotation *around* the axis of a direction cannot be captured by the vector itself. This relates to the concept of a *quotient space*, where elements of a larger group are collected together if they only differ by an element of some subgroup, so that each collection of group elements represent unique transformations of some space. In this case, all rotations around a fixed axis leave that axis unchanged, so the space of directions S^2 consisting of 3d unit vectors can be understood as the quotient SO(3)/SO(2), where SO(2) is the group describing rotations in 2D space. I.e. all SO(3) rotations where one of the axes of rotation are removed. A full treatment of quotient spaces is outside the scope of this report, so we refer the interested reader to standard references for a full formal treatment.

SE(3) is simply interpreted as a rotation followed by a translation, and is denoted

$$\mathrm{SE}(3) = \mathbb{R}^3 \rtimes \mathrm{SO}(3)$$

where $\rtimes$ represents the semi-direct product of the translation group $\mathbb{R}^3$ and the rotation group SO(3). This means the group consists of all combinations of elements in the two groups[3]. The product is called **semi**-direct because rotations affect translations when group elements are multiplied, i.e., there is a one-way interaction.

Concretely, an element of SE(3) can be written as a pair (t, R), with $t \in \mathbb{R}^3$ and $R \in \mathrm{SO}(3)$. When this element acts on a point $x \in \mathbb{R}^3$, the transformation is:

$$x \mapsto Rx + t.$$

It is straightforward to verify that SE(3) also satisfies the four group axioms.

Note that the networks built in this report are equivariant to SE(3), meaning a 3d rotation and translation of the input, leads to an equivalent rotation and translation of the output.

[2]There also exists the orthogonal group, O(3), which differs by also including mirroring, i.e. the matrices can have determinant -1 or 1

[3]It can be useful to define the group of all translations $\mathbb{R}^3$ as a quotient space: SE(3) / SO(3), since all transformations that differ by a rotation are grouped together, leaving one set of transformations per translation

2.1.2 Representations

As groups are abstract concepts, they can both represent physical actions, other mathematical objects such as matrices or be treated as an abstract concept. In our work, our SO(3) group elements will be 3d rotation matrices, R, however, we will also introduce larger vectors consisting of multiple subvectors "blocks", where each vector block must be rotated individually. We will furthermore introduce higher dimensional vectors such as 5d, 7d, etc., which rotate in 3d space in different ways[4].

While all these are different types of objects, they still rotate in 3d based on the same types of rotations, therefore, to ensure mathematical rigor one defines representations $\rho(g)$ which can map group elements to the actual mathematical objects that rotate different-dimensional vectors. These representations are *homomorphisms* of the group meaning they are maps from one group to another that satisfies

$$\rho(g_1) \cdot \rho(g_2) = \rho(g_1 \cdot g_2)$$

Meaning that the representation respects the underlying structure of the group (how group elements combine). This axiom has several consequences, among others that the representation of the identity element must also be the identity element.

Note that not all representations are informative; for example, the trivial representation, which maps every group element to the identity transformation, formally satisfies the homomorphism property but conveys none of the group's structure. In contrast, the representations used here, and commonly employed in the literature on equivariant neural networks, are typically chosen to faithfully represent the group action. That is, they are designed to preserve the distinctness of group elements and capture how different types of mathematical objects (such as scalars, vectors, and higher-order tensors) transform under rotations. These representations encode the geometric content of the group in a way that aligns with the structure of the data.

2.1.3 Irreducible representations

To effectively construct SE(3)-equivariant neural networks, we utilize the concept of irreducible representations (irreps). As we know from diagonalization of matrices, one can often reduce the complexity of transformations by a change of basis, and this is also the case for representations. As it turns out, it's not always possible to diagonalize, but representations can be decomposed into "block-diagonal" form, consisting of smaller matrices on the diagonal. These smaller matrices we call irreducible representations and they cannot themselves be decomposed further into smaller representations. Furthermore, the resulting vector of the change of basis also now consists of *sub-vectors* that are individually transformed by the transformation matrix. These sub-vectors are fundamental to how we will build up our equivariant networks later. For SO(3), these irreps are particularly well-studied and fundamental, directly corresponding to the spherical harmonics of different degrees l, which we will further introduce in section 2.3.

Specifically, each irrep of SO(3) is labeled by a non-negative integer l, and has dimension $2l+1$. For example, the $l = 0$ representation (scalars) is one-dimensional and invariant under rotations, the $l = 1$ representation corresponds to ordinary 3-dimensional vectors transforming under rotations, and higher l irreps describe increasingly complex objects such as tensors. Crucially, any finite-dimensional representation of SO(3) can be expressed uniquely as a direct sum of these irreps.

This decomposition into irreps greatly simplifies theoretical analysis and practical computations in equivariant networks, since each irreducible component transforms independently. This independence forms the basis for efficient equivariant computations, notably employing Wigner D-matrices, as discussed in the next section.

[4]Higher-dimensional tensors are known from physics and can e.g. represent stress in materials, and there is therefore a defined way to rotate them in 3d space

2.1.4 Wigner D-matrices

The irreducible representations of SO(3) are most commonly realized via the *Wigner D-matrices*, denoted $D^{(l)}(R)$, where $l \in \mathbb{N}_0$ is the degree of the representation and $R \in \mathrm{SO}(3)$ is a rotation. These matrices act on vector spaces of dimension $2l+1$, and provide a concrete realization of how type-l tensors transform under rotation.

Each matrix $D^{(l)}(R) \in \mathbb{C}^{(2l+1)\times(2l+1)}$ satisfies the homomorphism property:

$$D^{(l)}(R_1 R_2) = D^{(l)}(R_1) D^{(l)}(R_2)$$

. These matrices generalize the familiar 3D rotation matrices (which correspond to $l = 1$) to higher-order representations, such as those acting on quadrupole tensors ($l = 2$).

Wigner D-matrices are typically parametrized by Euler angles (α, β, γ) which we assume the reader is familiar with.

2.1.5 Equivariance definition

Equivariance is defined for a certain function, $f : X \to Y$, over a certain group, G, and is written mathematically

$$f(\rho_X(g) \cdot x) = \rho_Y(g) \cdot f(x), \quad \forall x, g$$

where f could be any function, e.g. an MLP, and g is a group element, like a rotation matrix. An example of how this could look is seen in figure 1. Note here the representations, ρ_X and ρ_Y, are used to account for the fact that the domain and codomain may be different sets. Commonly one would define the group elements to be mathematical objects that work directly on the domain and equivariance written:

$$f(g \cdot x) = \rho(g) \cdot f(x)$$

2.2 Structure of SE(3) equivariant networks

Firstly, the structure of the SE(3) equivariant neural networks used in this report is introduced. The reader should note that there are generally many possible structures, but the goal of this section is to illuminate the choice as the natural extension of the already established neural networks.

Much like traditional neural networks, we have a number of hidden layers with differing amount of nodes. However, we extend each node to be a type l tensor instead of a scalar. This gives the network architect another choice when building layers, namely that of which types of tensors and how many of each it will be built from. That means effectively that a hidden layer of 3 nodes might consist of 9 numbers, if there is a type 0, 1 and 2 tensor.

In practice, there have been shown to be diminishing returns when using type 2 tensors[4] or higher in your neural networks due to increased time complexity of computations [3]. However, new research has decreased the calculation time of the GC product from $O(L^6)$ to $O(L^3)$, where L is the largest order of the tensors in the product. This could make higher order tensors worth the extra compute [5].

Just like traditional networks, each node in one layer is connected with weights to each node in neighboring layers. This is clearly defined if a neural network consists of only the same type of tensor, but more theory is needed for weights between different type tensors. For now, we note that we have achieved a simple equivariant neural network consisting of nodes of only the same type:

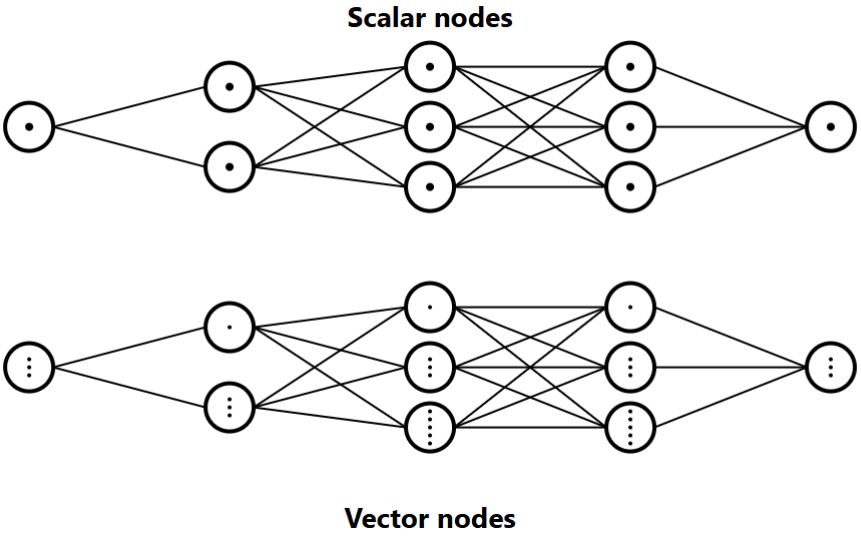

Figure 2: Illustration of traditional neural networks architectures (top) and EGNN architecture (bottom). Note that the EGNN architecture also has scalar values for some nodes, but for others has higher-dimensional vector values. To effectively let a 5d vector influence a 3d vector in the next layer, we use the CG product described later in this report.

2.2.1 Example: Simple Equivariant MLP

The simplest equivariant neural network one could construct is one that only transforms between the same type of nodes, as a linear combination of vectors is equivariant:

$$f(Rx_1, \ldots, Rx_n) = \sum_{i=1}^{n} c_i R x_i = R\left(\sum_{i=1}^{n} c_i x_i\right) = Rf(x_1, \ldots, x_n)$$

To introduce a nonlinearity that remains equivariant, one can base it on the rotation-invariant magnitude $\|x\|$. Defining, for a learnable bias $b \geq 0$,

$$\sigma(x) = \mathrm{ReLU}(\|x\| - b)\,\frac{x}{\|x\|}.$$

Then for any $R \in \mathrm{SO}(d)$,

$$\sigma(Rx) = \mathrm{ReLU}(\|Rx\| - b)\,\frac{Rx}{\|Rx\|} = \mathrm{ReLU}(\|x\| - b)\,\frac{Rx}{\|x\|} = R\,\sigma(x).$$

While this is a completely valid way of creating neural networks, it is somewhat restrictive. One may for example ask, is it not possible to combine the elements of the tensors in between themselves, and also how one would model interaction between tensors of different degrees. One could try to answer these questions with linear algebra (see appendix 7.3), however, this is slow and not easily generalizable. Therefore, we introduce the spherical harmonics and the CG product as theoretical background which ultimately leads to a framework where this is possible.

2.3 Spherical harmonics

Spherical harmonics are functions defined on the shell of a sphere, S^2. Thus they are mappings from the set of unit-vectors, or from two angles, to a real number.

Spherical harmonics work much like the Fourier functions, in the way that they form an infinite orthonormal basis for functions on the sphere, meaning an infinite weighted sum of these harmonics can approach any continuous function on the sphere.

The spherical harmonic functions are given by:

$$Y_l^m(\theta, \phi) = N_l^m \, P_l^{|m|}(\cos\theta) \, e^{im\phi}$$

where:

- $l \in \mathbb{N}_0$ is the *degree*,
- $m \in \{-l, \ldots, l\}$ is the *order*,
- $\theta \in [0, \pi]$ is the *polar angle* (colatitude),
- $\phi \in [0, 2\pi)$ is the *azimuthal angle* (longitude),
- $P_l^{|m|}$ is the associated Legendre polynomial, and
- N_l^m is a normalization constant ensuring orthonormality over the sphere.

For example, the lowest-degree complex spherical harmonics are:

$$\begin{aligned} Y_0^0(\theta, \phi) &= \frac{1}{2}\sqrt{\frac{1}{\pi}} \\ Y_1^{-1}(\theta, \phi) &= \frac{1}{2}\sqrt{\frac{3}{2\pi}} \, \sin\theta \, e^{-i\phi} \\ Y_1^0(\theta, \phi) &= \frac{1}{2}\sqrt{\frac{3}{\pi}} \, \cos\theta \\ Y_1^1(\theta, \phi) &= -\frac{1}{2}\sqrt{\frac{3}{2\pi}} \, \sin\theta \, e^{i\phi} \end{aligned}$$

These functions oscillate on the sphere in both angular directions and are orthonormal, which has the same definition as for orthonormal finite vectors, except the inner product is defined as an integral instead of a dot product:

$$\int_0^{2\pi} \int_0^{\pi} Y_l^m(\theta, \phi) \, Y_{l'}^{m'*}(\theta, \phi) \, \sin\theta \, d\theta \, d\phi = \delta_{ll'} \delta_{mm'}$$

Much like the Fourier basis on the circle is organized by frequency, with each nonzero frequency contributing two functions (sine and cosine), the spherical harmonic basis on the sphere is also organized by frequency bands. These bands are indexed by the non-negative integer l, commonly referred to as the *degree*, same as for irreps. For each l, there are $2l + 1$ linearly independent spherical harmonics, corresponding to the integer orders $m = -l, -l + 1, \ldots, l$. See the image for a visualization of the first four spherical harmonic degrees:

We let the attributes in our hidden layers correspond to functions on the sphere written in the spherical harmonic basis. This gives us the advantage that rotation of the entire function is simply a linear combination of the respective degrees of the spherical harmonics independent of each other. The transformation matrices are precisely the wigner-d matrices of before, and thus, with spherical harmonics, everything is in irreducible form. This is much like how the phase of functions defined in the fourier basis can be adjusted by linear combinations of each frequency.

Then we can combine two such functions by multiplying them together, which is an equivariant operation on the function space. But when multiplying sine and cosines functions of certain frequencies together, one can rewrite them to sine and cosine functions of higher and lower frequencies. The derivation for actual spherical harmonics is long, and thus we have here exemplified the calculations with simple fourier functions:

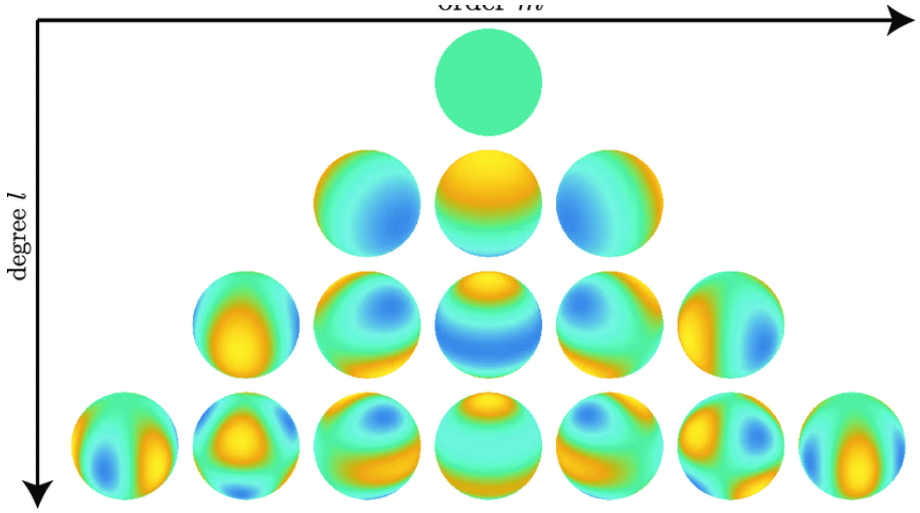

Figure 3: Visualization of spherical harmonics for the first four degrees. Colors represent function values, with blue indicating negative values and yellow indicating positive values. Image adapted from https://www.researchgate.net/publication/321331471_Filtering_on_the_Unit_Sphere_Using_Spherical_Harmonics.

$$
\begin{aligned}
&(0.7\sin(\theta)+\cos(\theta))\cdot(\sin(\theta)-0.3\cos(\theta))\\
&=0.7\sin^2(\theta)+0.79\cos(\theta)\sin(\theta)-0.3\cos^2(\theta)\\
&=\sin^2(\theta)+0.79\cos(\theta)\sin(\theta)-0.3\sin^2(\theta)-0.3\cos^2(\theta)\\
&=\sin^2(\theta)+\frac{0.79}{2}\cdot\sin(2\theta)-0.3\\
&=\frac{1-\cos(2\theta)}{2}+\frac{0.79}{2}\cdot\sin(2\theta)-0.3\\
&=\frac{0.79}{2}\cdot\sin(2\theta)-0.5\cos(2\theta)+0.2
\end{aligned}
\tag{1}
$$

as one can see this results in constant terms as well as ones with 2x frequency. The same applies to spherical harmonics. The general rule is that when you multiply a degree-l_1 and a degree-l_2 spherical harmonic, the result can be rewritten as a sum of spherical harmonics with degrees ranging from $|l_1 - l_2|$ to $l_1 + l_2$. We will not prove this, but the reader can learn more in standard references on harmonic analysis on the sphere, such as in texts on representation theory of SO(3) or in mathematical physics treatments of angular momentum addition.

As one can see this is a way of producing nodes of different dimensionality. However, it requires two inputs for this product. In practice we keep one input fixed, a, which contains geometric information about our data. We call it the steering attribute, and it is written in spherical harmonic form, approaching a dirac delta function on the sphere in the direction of the vector a as the amount of spherical harmonics increases. Thus it has degrees spanning as high as wanted, just like how adding more Fourier coefficients increases exactness of functions on the circle. The hidden node values, h, from the current layer will be the other input besides a. Here practical considerations come in, so while it wouldn't normally make sense to have multiple coefficients for the same fourier functions, we do that with spherical harmonics, as they can carry information for the neural network (think having multiple functions in one node). We also often restrict the amount degree of the spherical harmonics being outputted for efficiency's sake, as higher degree spherical harmonics are more expensive to multiply.

However, the output is not clearly connected to our representation of the spherical harmonics in vector-form. So far, we would have to calculate the weighted sum of the spherical harmonics

with the coefficients of both vectors, multiply the subsequent functions together, do trigonometry to write everything in terms of spherical harmonics and then go back to the vector representation by noting coefficients. To achieve this multiplication directly in the vector space, we can use the Clebtsch-Gordan (CG) product instead.

2.4 Clebtsch-Gordan product

The Clebtsch-Gordan product simply performs the above product of functions directly in the vector space representing the spherical harmonics. One can compute the Clebtsch-Gordan product using the Clebtsch-Gordan coefficients, which could technically be calculated with trigonometric rules like we did in 1. In practice these are found in pre-computed tables. The formula is to calculate the Clebtsch-Gordan product is:

$$\left(\tilde{\mathbf{h}}^{(l_1)} \otimes_{\text{cg}} \tilde{\mathbf{h}}^{(l_2)}\right)^{(l)}_{m} = \sum_{m_1=-l_1}^{l_1} \sum_{m_2=-l_2}^{l_2} C^{(l,m)}_{(l_1,m_1)(l_2,m_2)}\, h^{(l_1)}_{m_1}\, h^{(l_2)}_{m_2}$$

Here, l and m index the degree and component of the output tensor, respectively. The input tensors are indexed by their respective degrees l_1 and l_2, with components m_1 and m_2. The coefficients $C^{(l,m)}_{(l_1,m_1)(l_2,m_2)}$ are the Clebsch–Gordan coefficients.

2.4.1 Example: Equivariant Network Layer

To illustrate the concepts introduced above, consider a concrete example of an equivariant network layer. Suppose we have geometric information $\mathbf{a}$ (the steering attribute) and a hidden layer with 20 nodes distributed as 5 type-0 (scalars), 10 type-1 (vectors), and 5 type-2 (tensors). The steering attribute $\mathbf{a}$ is represented as a 9-dimensional vector containing spherical harmonics of degrees 0, 1, and 2.

To compute one node in the next layer, say, a type-1 node, we would evaluate the Clebsch-Gordan product between each of the 20 input nodes and the 3 types in $\mathbf{a}$, yielding $20 \times 3 = 60$ potential couplings. Each coupling is weighted by a learnable parameter w_k. However, due to the triangle inequality constraint in the CG coefficients, many of these products vanish: for instance, coupling two type-0 tensors or a type-0 with a type-2 cannot produce a type-1 output, so 10 of the 60 terms contribute zero.

The non-zero CG products are aggregated and passed through an equivariant nonlinearity as described in Section 2.2.1. This process is repeated for each output node, with the specific learnable weights w_k determining how strongly different geometric relationships influence the final representation.

2.5 Invariant feature extraction from equivariant representations

A key property in SE(3) equivariant networks is the ability to extract rotation-invariant features from equivariant representations. This is crucial for creating features that can be processed by standard scalar neural network components without breaking equivariance of the full network.

For any irreducible tensor representation of SO(3), the L2 norm (magnitude) is invariant under rotation. Specifically, a type-l feature vector $\mathbf{h}^{(l)} \in \mathbb{R}^{2l+1}$ transforms under a rotation $R \in \text{SO}(3)$ as

$$\mathbf{h}^{(l)} \mapsto D^{(l)}(R)\, \mathbf{h}^{(l)},$$

where $D^{(l)}(R)$ is the Wigner D-matrix associated with the l-th irreducible representation. These matrices are orthogonal, satisfying

$$D^{(l)}(R)^\top D^{(l)}(R) = I, \quad \text{and} \quad \det D^{(l)}(R) = 1.$$

As a result, the norm of $\mathbf{h}^{(l)}$ is preserved under the action of SO(3):

$$\|D^{(l)}(R)\,\mathbf{h}^{(l)}\|_2^2 = \left(D^{(l)}(R)\,\mathbf{h}^{(l)}\right)^\top \left(D^{(l)}(R)\,\mathbf{h}^{(l)}\right) = \mathbf{h}^{(l)\top} D^{(l)}(R)^\top D^{(l)}(R)\mathbf{h}^{(l)} = \|\mathbf{h}^{(l)}\|_2^2.$$

Thus, the quantity $\|\mathbf{h}^{(l)}\|_2$ is invariant under any SO(3) rotation.

2.6 Graph neural networks (GNNs)

Graph Neural Networks (GNNs) are a type of feedforward[5] neural network designed specifically to process graph-structured data. In such graphs, each node and each directed edge can carry its own feature vector.
GNNs come in many forms but typically maintain their initial connectivity structure throughout all hidden layers, unlike MLPs or CNNs. Thus, only the node and edge feature vectors are updated from one layer to the next, while the connectivity between nodes remains fixed.
At each layer, every node updates its feature vector by aggregating information received from incoming edges and their source nodes. This aggregation is performed by a learned *message function*, which computes individual messages for each incoming edge based on the features of the source node, the target node itself, and the connecting edge. These messages are then combined using a fixed aggregation operator (such as sum, mean, or max). The aggregated messages are subsequently combined with the node's own current features using another learned transformation called the *node update function*. Together, these transformations yield updated node representations that capture both local and global structural information, though the global information is limited based on node connectivity and amount of message passing layers.
Formally, the update rule for node i at layer l is expressed as:

$$f_i^{(l+1)} = \psi_f^{(l)}\left(f_i^{(l)}, \bigoplus_{j\in\mathcal{N}(i)} \psi_m^{(l)}\left(f_i^{(l)}, f_j^{(l)}, a_{ji}^{(l)}\right)\right)$$

where:

$f_i^{(l)}$: Feature vector of node i at layer l

$a_{ji}^{(l)}$: Feature vector of directed edge from node j to node i at layer l

$\mathcal{N}(i)$: Set of source nodes j with edges directed into node i

$\psi_m^{(l)}$: Learned message function computing information passed from node j to node i

$\bigoplus$: Aggregation operator combining messages (e.g., sum, mean, max)

$\psi_f^{(l)}$: Learned non-linear node update function combining aggregated messages with the node's current featu

Interestingly, a convolutional layer in a CNN can be viewed as a special case of a graph neural network layer.

2.7 Making GNNs equivariant

Making a graph neural network SE(3)-equivariant requires maintaining equivariance with respect to translation and rotation of the feature vectors throughout all layers. Consequently, the node update function and message function must each be designed to explicitly preserve this equivariance property, which is done using the EGNNs described in the previous theory.

[5]Specialized recurrent variants of GNNs exist, but these are beyond the current scope.

3 Methods

Equivariant and traditional graph neural networks (GNNs) are benchmarked on a controlled geometric-regression task against a baseline. Three-dimensional object meshes are converted to point clouds by two simulated LiDAR sensors; only the point cloud is provided to the models, which must predict the original mesh centroid. The following section first presents the dataset and pre-processing pipeline, and then details the models together with the training and evaluation procedures.

3.1 Dataset

We use 11 590 Wavefront `.obj` meshes from the ShapeNet Parts subset.[6] Only vertex positions and face indices are retained; textures and part labels are ignored.

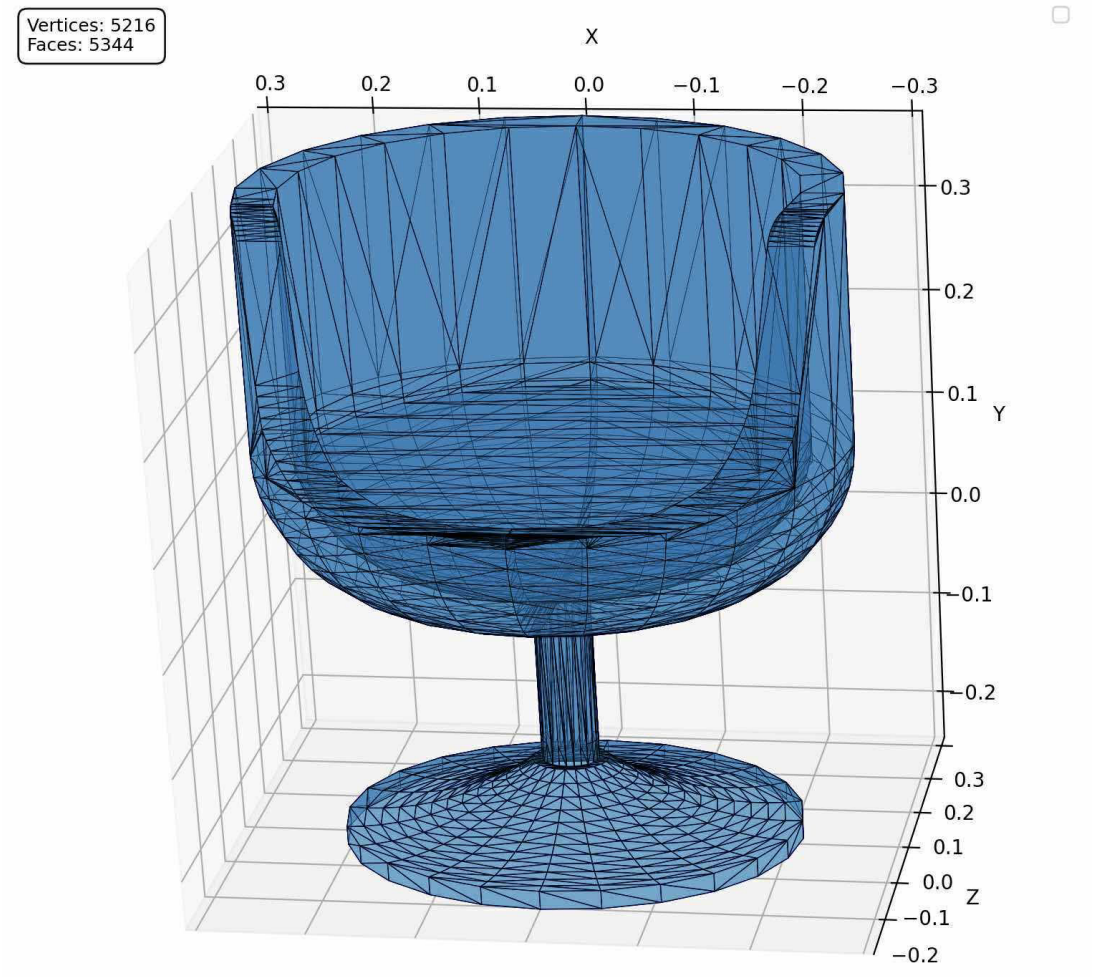

Figure 4: Example ShapeNet object (*chair* class) used as raw input.

3.2 Pre-processing Pipeline

Figure 4 shows a representative mesh, and Figure 5 shows the corresponding point cloud produced by the two–sensor LiDAR simulation. The following points concisely detail the pre-processing pipeline.

[6] https://paperswithcode.com/dataset/shapenet

1. Mesh normalisation

1. *Random uniform rotation* Removes pose bias so models cannot exploit a preferred orientation.

2. *Isotropic scaling.* Computes the mesh centroid with formula:

$$\mathbf{c}_{\text{mesh}} = \frac{\sum_{f=1}^{F}(\mathbf{v}_{f0} + \mathbf{v}_{f1} + \mathbf{v}_{f2})A_f}{3\sum_{f=1}^{F} A_f}, \quad A_f = \|(\mathbf{v}_{f1} - \mathbf{v}_{f0}) \times (\mathbf{v}_{f2} - \mathbf{v}_{f0})\|$$

where F is the amount of faces in the mesh, f a face index, v_{f0}, v_{f1} and v_{f2} are the coordinates of each face vertex respectively for face f, and A_f is the area of the face. Let $r_{\max} = \max_v \|\mathbf{v} - \mathbf{c}_{\text{mesh}}\|$. Divide all vertex coordinates by $r_{\max}$ so the farthest vertex lies one unit from the centroid. Reduces object size variance mitigating size bias and increasing training effectiveness.

2. LiDAR simulation Choose $\mathbf{d} \sim \text{Uniform}(\{\mathbf{x} \in \mathbb{R}^3 \mid \|\mathbf{x}\| \le 0.5\})$ and set $\mathbf{o} = \mathbf{c}_{\text{mesh}} + \mathbf{d}$. Place two sensors at

$$\mathbf{c}_1 = \mathbf{o} + (0, 0, 2), \qquad \mathbf{c}_2 = \mathbf{o} - (0, 0, 2),$$

each 2 units from $\mathbf{o}$ and therefore 4 units apart. Each camera emits a 30×30 ray grid within a 45° cone to a maximum range of 4 units, producing 1 800 rays in total. Placing the cameras around the randomly offset o instead of c_{mesh} prevents data leakage through camera positions.[7]

3. Point-cloud normalisation Let $\{\mathbf{p}_i\}_{i=1}^{M}$ be the ray-mesh intersection points. Translate them so their mean is in the origin and scale so the largest distance from the origin is 1:

$$\mathbf{p}'_i = \frac{\mathbf{p}_i - \frac{1}{M}\sum_j \mathbf{p}_j}{\max_k \|\mathbf{p}_k - \frac{1}{M}\sum_j \mathbf{p}_j\|}.$$

Apply the same transform to $\mathbf{c}_{\text{mesh}}$. The regression target is then

$$\mathbf{c}'_{\text{mesh}} = \frac{\mathbf{c}_{\text{mesh}} - \frac{1}{M}\sum_j \mathbf{p}_j}{\max_k \left\|\mathbf{p}_k - \frac{1}{M}\sum_j \mathbf{p}_j\right\|}.$$

4. Graph construction Each point becomes a node with position feature $\mathbf{x}_i \in \mathbb{R}^3$. Directed edges connect every node to its ten nearest neighbours; edge features are the displacement vectors $\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$. Each node also has the attribute mean edge length, which is defined as:

$$\bar{r}_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \|\mathbf{r}_{ij}\|.$$

The final dataset contains 11 590 graphs with 22–1 601 nodes each (mean 604). The mean offset between the point-cloud centroid and the mesh centroid (what the model needs to predict) is 0.1066 units (standard deviation 0.0651).

[7] Meshes closer to a camera yield locally denser point clouds. If the camera baseline were fixed at the origin, a model could infer camera locations from density and then predict the target centroid as the midpoint of the cameras. Randomly translating the camera pair around $\mathbf{c}_{\text{mesh}}$ breaks this shortcut and forces the network to learn genuine 3-D geometry.

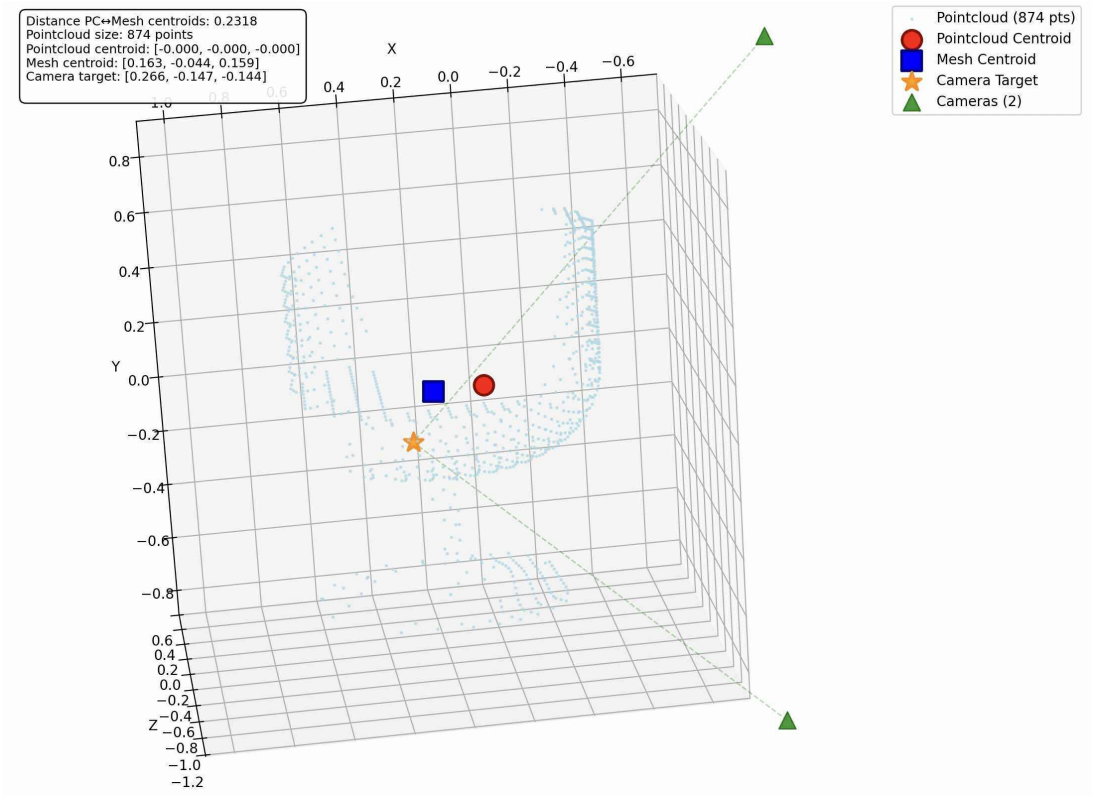

Figure 5: Point cloud produced by the two–sensor LiDAR simulation after normalisation.

3.3 Models

We keep number of learnable parameters approximately constant across models. Both learnable models use four message-passing layers with residual connections, mean pooling, and a three-layer MLP. Inputs are node positions $\mathbf{x}_i$, edge displacements $\mathbf{r}_{ij}$, edge lengths $d_{ij} = \|\mathbf{r}_{ij}\|$, and node-level $\bar{r}_i$.

Three predictors are compared:

- **Baseline**: returns the point-cloud centroid, i.e. **0** after normalisation
- **Basic GNN**: non-equivariant message-passing network
- **Basic GNN Augmented**: non-equivariant message-passing network with data augmentation in training
- **Equivariant GNN**: rotation-equivariant network with steerable kernels

All implementations use PyTorch; code and configuration files are available on our GitHub repository https://github.com/jonathantybirk/equivariant-mesh-centroid-estimation.

3.3.1 Non-Equivariant Graph Neural Network

Our first learned model implements a graph neural network with enhanced geometric feature extraction, designed to provide access to the same geometric information available to the equivariant model but processed without equivariance guarantees.

To enable fair comparison with the equivariant architecture, we expand both node and edge features through polynomial transformations that approximate the geometric expressiveness of spherical harmonics. Node positions $[x, y, z]$ are transformed into 11-dimensional features

$[x, y, z, x^2, y^2, z^2, xy, xz, yz, \|x\|, \|x\|^2]$, while edge displacement vectors $r_{ij} = x_j - x_i$ undergo the same polynomial expansion. This feature engineering provides access to quadratic terms and cross-products that encode geometric relationships similar to spherical harmonic representations, but without mathematical equivariance guarantees.

The architecture employs 2 message passing layers with 16-dimensional hidden representations. Message computation incorporates source nodes, target nodes, edge features, and explicit distance information: $m_{ij} = \text{MLP}([h_i, h_j, e_{ij}, \|r_{ij}\|])$. Node updates use a gating mechanism that mimics the equivariant model's design, where gates are computed from current node features, aggregated messages, and average neighbor distances. Final centroid prediction uses softmax-weighted aggregation: $\hat{c} = \sum_{i=1}^{N} \text{softmax}(\text{MLP}(h_i)) \cdot x_i$, matching the equivariant model's output strategy.

This architecture serves as a "fair comparison" baseline by providing equivalent geometric information access through polynomial feature expansion while using similar gating mechanisms and prediction strategies. The key difference is that geometric relationships are learned rather than enforced through mathematical construction. The model contains 7,041 parameters.

Data Augmentation The non-equivariant model is enhanced with a data augmentation. The data augmentation is a random rotation of the point cloud. This is done by generating a random rotation matrix $\mathbf{R} \in SO(3)$ using Rodrigues' rotation formula, and applying it to the input data: $\mathbf{x}' = \mathbf{R}\mathbf{x}$, $\mathbf{r}'_{ij} = \mathbf{R}\mathbf{r}_{ij}$.

3.3.2 Equivariant GNN

Before delving into symbols we summarise the forward pass in four stages:

1. **Initialisation**: node coordinates $\mathbf{x}_i$ are converted to spherical–harmonic features; edges receive spherical–harmonic attributes *(Section 2.3)*.
2. **Message construction**: for every edge $j \to i$ we form $\mathbf{h}_{ij}$ (concatenated source, target, and distance) and couple it with $\mathbf{a}_{ij}$ through Clebsch–Gordan products *(Section 2.4)* to obtain edge messages $\mathbf{m}_{ij}$.
3. **Gated node update**: aggregated messages are modulated by scalar gates derived from rotation–invariant features *(Section 2.5)*.
4. **Invariant read-out**: the same invariants feed a small MLP whose softmax weights the node coordinates to regress the centroid.

For every degree $0 \le l \le L$ let $V_l = \mathbb{R}^{2l+1}$ carry the Wigner $D^{(l)}$ irriducable representation(Section 2.1.3). With multiplicity M the *equivariant feature space* at a node is the direct sum

$$\mathbf{f}_i = \bigoplus_{l=0}^{L} \bigoplus_{m=1}^{M} \mathbf{f}_{i,l}^{(m)} \in \bigoplus_{l=0}^{L} V_l^{\oplus M}$$

Each block $\mathbf{f}_{i,l}^{(m)} \in V_l$ transforms independently under rotations. The concatenation order is $[l = 0^{(1)}, \ldots, l = 0^{(M)}, l = 1^{(1)}, \ldots]$.

For an edge $j \to i$ define $\mathbf{h}_{ij} = \mathbf{f}_i \oplus \mathbf{f}_j \oplus d_{ij}$ where $d_{ij} = \|\mathbf{r}_{ij}\|$. Messages are produced in one line:

$$\boxed{\mathbf{m}_{ij} = \sum_{k \in \mathcal{K}} w_k \left(\mathbf{h}_{ij}^{(k)} \otimes_{\text{cg}} \mathbf{a}_{ij}\right)} \tag{2}$$

Here $\mathcal{K}$ enumerates all Clebsch–Gordan couplings that satisfy the triangle inequality $|l_h - l_a| \le l_{\text{out}} \le l_h + l_a$ (the admissible set discussed in Section 2.4); w_k are learnable scalars different across the M channels;

Rotationally invariant scalars are extracted as $\mathbf{s}(\cdot) = [\langle\cdot\rangle_{l=0}, \|\cdot\|_{l=1}, \dots]$ (see Section 2.5). The gate for irrep l is

$$\mathbf{G}_i^{(l)} = \psi^{(l)}(\mathbf{s}(\mathbf{f}_i),\ \mathbf{s}(\textstyle\sum_j \mathbf{m}_{ij}),\ \bar{d}_i) \in \mathbb{R},$$

and the node update is the block-wise residual

$$\mathbf{f}_i \leftarrow \mathbf{f}_i + \mathbf{G}_i \odot \sum_j \mathbf{m}_{ij}.$$

Practical details. Clebsch-Gordan tensors are sparse; we pre-compute and cache only the non-zero couplings in a list of $|\mathcal{K}|$ tensors, each carrying a learnable weight w_k for every multiplicity channel. Index bookkeeping in the PyTorch code slices the concatenated feature tensor exactly as dictated by the irrep dimensions but does not affect the abstract equations above. CG coefficents and spherical harmonics are computed using the e3nn library [6].

Hyperparameters. The PyTorch implementation uses the settings summarised in Table 1. All runs employ the Adam optimiser with the listed learning rate and weight decay.

Symbol	Value	Description
T	2	message–passing steps
L_{edge}	2	max SH degree for edge attributes
M	3	node multiplicity (channels per irrep)
MLP_{msg}	[64, 32]	hidden dims in σ_g
$\text{MLP}_{\text{final}}$	[64, 32]	hidden dims in readout g_θ
p_{drop}	0.02	dropout rate
η	10^{-3}	learning rate
λ	10^{-5}	weight decay

Table 1: Key hyperparameters for the Equivariant GNN.

3.4 Training and Evaluation

The four models are trained for 100 epochs and evaluated on the same dataset, as described in Section 3.1. One thousand graphs form the development set (800 train, 200 validation); the remaining 10 590 graphs are held out for testing. This small training set aligns with our focus on limited-data scenarios, where equivariant architectures are theoretically expected to demonstrate their greatest advantages through efficient parameter sharing. The large test set of over 10,000 samples ensures robust statistical analysis of the final model comparisons. Two key metrics are used to assess model performance:

Prediction Accuracy: Measured as the L2 error between predicted and true mesh centroids:

$$\text{L}^2 \text{ distance} = \|\hat{\mathbf{c}} - \mathbf{c}_{\text{true}}\|_2 = \sqrt{\sum_{i=1}^{n} (\hat{c}_i - c_{\text{true},i})^2}$$

where $\hat{\mathbf{c}}$ is the predicted centroid and $\mathbf{c}_{\text{true}}$ is the ground truth mesh centroid.

Equivariance Error: Measures how well models satisfy rotational equivariance. For each test sample, we compute the predictions on both the original and rotated inputs, and then calculate the equivariance error as the L2 distance between the two predictions. The rotation is done in the same random way as the data augmentation for the non-equivariant model.

1. Compute predictions on both original ($\hat{\mathbf{c}}$) and rotated ($\hat{\mathbf{c}}'$) inputs

2. Calculate equivariance error: $\|\hat{\mathbf{c}}' - \mathbf{R}\hat{\mathbf{c}}\|_2$

Perfect equivariance yields zero error, while violations produce positive values proportional to the deviation from expected behavior.

3.4.1 Statistical Analysis

Comprehensive statistical analysis is used to assess the significance of of the differences between the models. This is performed using:

Confidence Intervals: 95% confidence intervals are calculated for all metrics using Student's t-distribution to account for sample size and variance. Note that the assumption of normality is held due to the large sample size of appx. 10 000, and that our data points are independent and identically distributed.

Paired t-tests: Since all models are evaluated on identical test samples, paired t-tests assess the statistical significance of performance differences between model pairs. This accounts for sample-specific difficulty and provides greater statistical power than an independent t-test.

Multiple Comparisons: All pairwise model comparisons are performed for both performance and equivariance metrics, with a Bonferroni-corrected significance threshold at $p < \frac{0.05}{16} = 0.003125$.

Models are ranked by three criteria: prediction accuracy (lower L2 distance is better), equivariance performance (lower error is better), and parameter efficiency (fewer parameters for similar performance is better).

Having established our experimental framework, dataset, and evaluation methodology, we now turn to the empirical results obtained from our comprehensive comparison of equivariant and non-equivariant approaches to centroid prediction.

4 Results

4.1 Model Performance Comparison

Following the methodology outlined in Section 3.4, we present the experimental results from our comprehensive evaluation of four different approaches: the simple Baseline model, Basic GNN, Basic GNN + Augmentation, and Equivariant GNN. All models were evaluated on the held-out test dataset consisting of 10,585 LiDAR point cloud samples, with assessment focused on the two key metrics established in our evaluation framework: prediction accuracy (measured as L^2 distance between predicted and true centroids) and equivariance error (measuring deviation from perfect rotational equivariance).

Figure 6 presents a comprehensive visual comparison of these results, revealing clear performance distinctions between equivariant and non-equivariant approaches.

4.2 Statistical Significance Analysis

Paired t-tests on the same test samples L2 prediction error revealed that the Equivariant GNN and Basic GNN + Augmentation comparison yielded a p-value of 0.218, indicating no statistically significant difference at $\alpha = 0.05$.

All other pairwise comparisons of L2 prediction error demonstrated statistically significant differences. The Equivariant GNN significantly outperformed the Basic GNN ($p = 6.11 \times 10^{-56}$), and the Basic GNN + Augmentation similarly outperformed the Basic GNN ($p = 1.32 \times 10^{-98}$). All learned models significantly surpassed the Baseline performance with p-values approaching zero.

The Equivariant GNN's t-test could not reject zero equivariance error, and had a mean of 0, strongly indicating that the model was implemented correctly with full equivariance. The EGNN

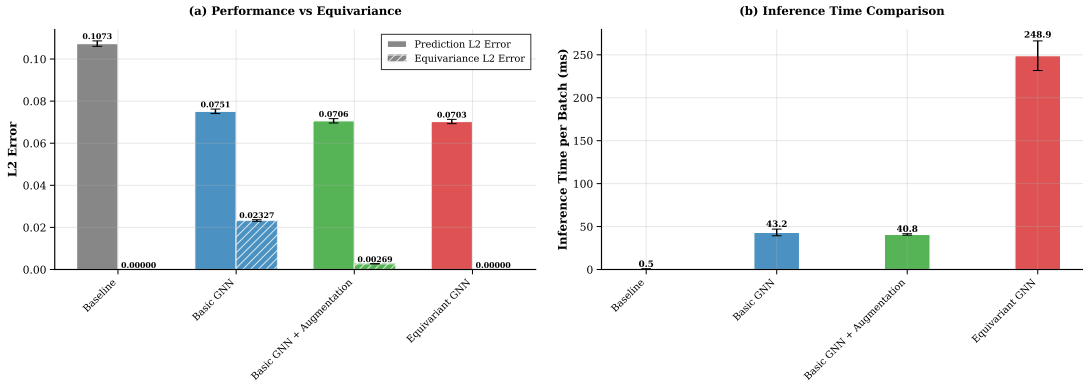

Figure 6: Comparative analysis of model performance showing (a) mean prediction L^2 error (solid bars) and (b) equivariance L^2 error (striped bars) both with 95% confidence intervals.

also significantly outperformed both the Basic GNN + Augmentation and the Basic GNN on this metric, confirmed by a paired t-test. The Basic GNN exhibited a non-zero equivariacne error, with a p-value below 0.001. The Basic GNN + Augmentation also showed significantly better equivariance than the Basic GNN, with a mean difference of 0.020572 and a p-value approaching 0.

4.3 Training Dynamics Analysis

Figure 7 presents a comprehensive analysis of the training dynamics across all models, illustrating training loss, validation loss, and generalization gap evolution over 100 epochs.

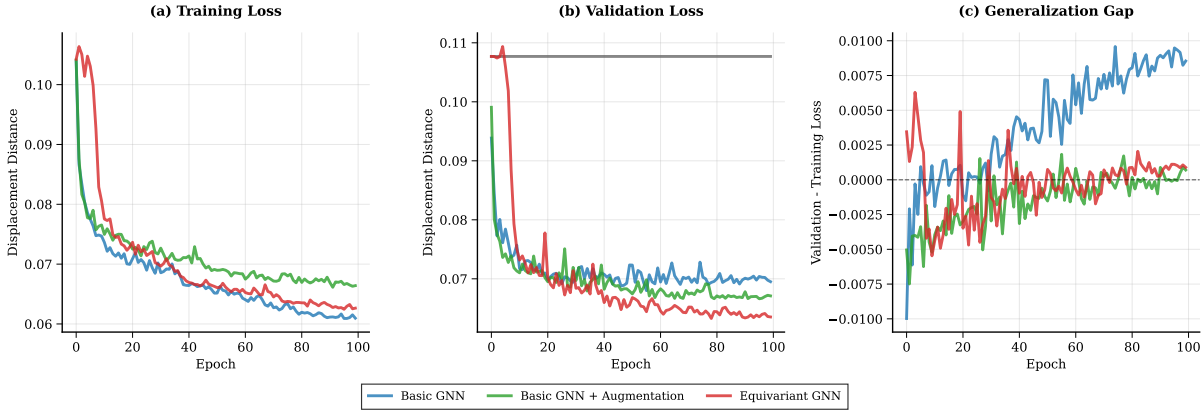

Figure 7: Training dynamics analysis showing (a) training L^2 error curves, (b) validation L^2 error curves, and (c) generalization gap (validation - training L^2 error) for all models over 100 epochs. The Equivariant GNN demonstrates stable learning with minimal overfitting, while the BasicGNN with augmentation shows effective regularization through data augmentation.

All models demonstrated smooth convergence throughout the 100-epoch training period. The Equivariant GNN exhibited initially slower convergence before the rapid descent phase, but subsequently maintained consistently the lowest validation L^2 error. The Basic GNN with augmentation exhibited more stable training behavior compared to the standard Basic GNN.

Analysis of the generalization gap reveals that the Equivariant GNN maintained the most stable generalization gap throughout training. The Basic GNN + Augmentation demonstrated improved generalization compared to the standard Basic GNN, albeit with the most variable generalization gap, as seen as large fluctuations in the generalization gap around epoch 2, 20, 30

and 40. The Basic GNN without augmentation exhibited the largest generalization gap across the training period.

4.4 Model Complexity

The Equivariant GNN uses 8,531 parameters, the Basic GNN variants use 7,041 parameters each, and the Baseline has trivially 0 parameters. The Equivariant GNN represents a 21% increase in parameter count compared to the Basic GNN variants.

5 Discussion

5.1 Key Findings and Implications

The experimental results reveal several important insights regarding the effectiveness of SE(3)-equivariant graph neural networks for centroid prediction in LiDAR point clouds. Most notably, the Equivariant GNN achieved prediction accuracy comparable to the best-performing non-equivariant approach (Basic GNN + Augmentation) while simultaneously maintaining perfect rotational equivariance. This finding challenges the assumption that incorporating geometric constraints necessarily compromises prediction performance, demonstrating instead that equivariance can be achieved without sacrificing accuracy.

The statistical analysis confirms that both the Equivariant GNN and Basic GNN + Augmentation significantly outperformed the standard Basic GNN, with the key distinction being the equivariance properties. While data augmentation improved the Basic GNN's robustness to rotations, it still exhibited measurable equivariance error (0.002694 ± 0.002176), whereas the Equivariant GNN achieved perfect equivariance (0.000000 error). This demonstrates the advantage to robustness of building geometric constraints directly into the network architecture rather than relying solely on data augmentation.

5.2 Expected vs. Observed Performance Benefits

The comparable performance between the Equivariant GNN and Basic GNN + Augmentation was somewhat unexpected, as other studies and theoretical considerations suggested the equivariant model should achieve superior validation accuracy. The weight-sharing inherent in SE(3)-equivariant architectures was anticipated to provide a significant advantage by increasing the training data efficiency through implicit parameter sharing across all possible rotations. Additionally, it was expected that non-equivariant models would suffer from reduced parameter efficiency, as substantial portions of their learned weights would necessarily be devoted to capturing rotated versions of the same underlying geometric relationships, effectively "wasting" representational capacity on redundant transformations that equivariant architectures handle automatically through their mathematical structure. These combined theoretical benefits were thought to translate to improved generalization and reduced overfitting compared to approaches relying solely on explicit data augmentation. This expectation was further supported by prior work demonstrating the effectiveness of geometric equivariance in related domains, where Weiler et al. [7] showed that 3D Steerable CNNs achieved "excellent accuracy and data efficiency" in protein structure tasks, establishing that equivariant architectures can provide both theoretical guarantees and empirical performance improvements.

However, the empirical results show that with the particular model configurations and hyperparameters employed in this study, data augmentation achieved comparable prediction performance to architectural equivariance. It is important to note that these results reflect the specific experimental setup rather than the optimal performance potential of each approach.

Notably, both approaches - architectural equivariance and data augmentation - demonstrated significant improvements over the basic non-equivariant GNN, confirming that addressing rotational symmetry is beneficial for this task. The observed performance parity between these two approaches does not necessarily indicate fundamental equivalence between data augmentation and architectural equivariance. Rather, it suggests that within the scope of this experimental setup, the theoretical advantages of equivariant weight sharing did not translate to measurable performance improvements over well-implemented data augmentation. This could be attributed to suboptimal hyperparameter choices, architectural differences, dataset characteristics, or task complexity[8]. The primary demonstrated advantage of the equivariant approach lies in the theoretical guarantees and perfect equivariance it provides, independent of hyperparameter tuning.

An additional consideration is that translation equivariance was incorporated into all network architectures through the point cloud centering preprocessing step. This normalization effectively provided all models with translation invariance, meaning the Equivariant GNN's primary theoretical advantage was limited to rotational equivariance rather than full SE(3) equivariance.

5.3 Theoretical vs. Statistical Equivariance

The perfect equivariance achieved by the Equivariant GNN validates the theoretical framework established in Section 2. The SE(3)-steerable kernels successfully preserved the geometric structure of the input transformations, translating rotations of the input point cloud into corresponding rotations of the predicted centroid. This theoretical guarantee manifested empirically as zero equivariance error across all test samples.

In contrast, the data augmentation approach, while improving equivariance compared to the unaugmented Basic GNN, represents an approximation rather than a guarantee. However, the remarkably low equivariance error achieved by the augmented model (0.002694 ± 0.002176) suggests an important insight: when the underlying task is inherently rotationally symmetric, as centroid prediction fundamentally is, models seem to naturally learn to respect that symmetry during training because violating equivariance would compromise performance. The data augmentation effectively teaches the network that rotational consistency is essential for optimal prediction accuracy. The residual equivariance error observed in the augmented model reflects the inherent limitations of learning equivariance through data exposure rather than architectural constraints, but the magnitude of this error demonstrates how effectively non-equivariant architectures can approximate equivariant behavior when the task structure demands it, but we cannot extrapolate beyond this specific task with certainty.

The theoretical guarantee of perfect equivariance provides some benefits, even though prediction accuracy is not one of them. The Equivariant GNN's zero equivariance error eliminates rotational bias entirely, ensuring that model predictions remain perfectly consistent regardless of the input's spatial orientation. This reliability can be crucial in applications where reproducible behavior is essential, such as robotic manipulation, medical imaging, or safety-critical systems, where even small inconsistencies due to input orientation could have significant consequences. Furthermore, the mathematical guarantee provides interpretability and trust in model behavior: practitioners can be certain that the model's geometric reasoning is fundamentally sound and will not exhibit unexpected rotational dependencies during deployment.

5.4 Computational Trade-offs

The results highlight important computational considerations in the equivariance-performance trade-off. The Equivariant GNN required 21% more parameters than the Basic GNN variants (8,531 vs. 7,041 parameters) and exhibited significantly longer inference times (248.9 ± 112.6 ms

[8]Unfortunately, it was not within the scope of the resources of this project to do a full hyperparameter search

vs. 40.8 ± 5.4 ms for Basic GNN + Augmentation). This computational overhead could be due to our implementation of the equivariant model, and especially handling the CG product.

The training dynamics analysis reveals a clear hierarchy in learning behavior that strongly favors the equivariant approach. As shown in Figure 7, the Equivariant GNN achieved the lowest validation loss and maintained the most stable generalization gap throughout training, followed by the Basic GNN + Augmentation in second place, with the standard Basic GNN exhibiting the worst performance across all training metrics. This consistent ranking, Equivariant GNN best, augmented GNN second, basic GNN worst, suggests that our equivariant model is fundamentally more robust to overfitting and demonstrates superior learning stability.

Even though there was no performance difference between the Equivariant GNN and Basic GNN + Augmentation on the test set, the Equivariant GNN's superior training dynamics suggest that architectural equivariance provides benefits beyond final prediction accuracy, offering more reliable convergence and better generalization behavior. However, it is important to note that these training observations are based on a relatively small development set (800 training samples and 200 validation samples), and the generally high variance in error across models means these training dynamics could be influenced by random effects rather than representing robust statistical evidence. This potential stability could be beneficial for model training, as it may reduce the need for extensive hyperparameter tuning and could lead to better performance on out-of-distribution data, but this finding would require validation on larger datasets to establish statistical significance.

5.5 Limitations and Scope

Several important limitations in the experimental design may have prevented the equivariant architecture from reaching its full potential. The sparse nature of the Clebsch-Gordan product introduced significant training challenges, as many tensor product combinations yield zero outputs due to symmetry constraints. This sparsity complicates gradient flow and requires careful initialization and learning rate scheduling to achieve stable convergence.

The current implementation employs only a single Clebsch-Gordan product per node before the aggregation function, which limits the model's expressiveness. Multi-layer Clebsch-Gordan products within each message passing step could potentially capture more complex geometric relationships and improve performance, but this architectural exploration was not pursued due to computational constraints and training complexity.

Furthermore, the hyperparameter optimization for the equivariant model was limited compared to what might be achievable with extensive computational resources. More sophisticated architectural choices, such as varying the number of message passing layers, exploring different multiplicities for each irreducible representation type, or implementing more complex gating mechanisms, could potentially unlock additional performance gains. The comparable performance observed between equivariant and augmentation approaches may therefore reflect suboptimal hyperparameter choices rather than fundamental limitations of the equivariant approach.

Additionally, the study was restricted to spherical harmonics representations up to degree 2, which constrains the complexity of geometric features that can be captured. Higher-order representations could improve expressiveness but would require addressing the associated computational overhead and training stability challenges.

5.6 Implications for Real-world Applications and Sustainability

The theoretical guarantee of perfect equivariance extends beyond prediction accuracy to broader computational sustainability considerations. Equivariant architectures reduce the need for extensive data augmentation during training, which can significantly lower computational costs and energy consumption. While individual augmented samples require minimal additional

computation, the cumulative effect across large-scale training can be substantial, particularly when considering the multiple rotation variants typically applied per training example. We note that increased inference time in our study leads in the opposite direction, towards increased compute and power usage, however, acknowledge that further optimization can be made.

Computational efficiency gains align with the United Nations Sustainable Development Goals (SDGs), particularly SDG 9 (Industry, Innovation and Infrastructure) through more resource-efficient machine learning systems. In autonomous vehicles and robotic systems, where LiDAR-based perception must operate reliably across diverse orientations, equivariant models could reduce both training costs and the need for orientation-specific recalibration, supporting more sustainable deployment of autonomous technologies.

Similarly, in automated sorting and recycling systems (SDG 12: Responsible Consumption and Production), orientation-invariant centroid prediction could improve processing efficiency while reducing computational overhead from data augmentation. Environmental monitoring applications (SDG 13: Climate Action) could benefit from more reliable geometric inference with lower computational footprints, enabling broader deployment of monitoring systems with limited energy budgets.

5.7 Future Research Directions

Several promising avenues emerge from this work that could provide deeper insights into the practical benefits of equivariant architectures. A critical evaluation would involve training each model on progressively smaller data subsets to assess whether the Equivariant GNN maintains accuracy longer as data become scarce. This data-efficiency analysis could reveal whether the theoretical advantages of weight sharing manifest more clearly in data-limited scenarios, potentially demonstrating that equivariant models provide greater value when training data is expensive or difficult to obtain.

Additionally, the observed superior training dynamics of the Equivariant GNN, including lower validation loss and more stable generalization gaps, require validation on larger datasets to establish statistical significance. The current observations are based on relatively small training (800 samples) and validation (200 samples) sets with high error variance, making it unclear whether these training advantages represent robust patterns or random effects. Experiments with larger datasets and multiple random seeds would be essential to confirm whether equivariant architectures truly provide more stable learning behavior.

Additionally, future work should explore architectural improvements that could unlock the full potential of equivariant approaches, including multi-layer Clebsch-Gordan products within message passing steps and higher-order spherical harmonic representations. Comprehensive hyperparameter optimization using more extensive computational resources could also determine whether the observed performance parity reflects fundamental limitations or simply suboptimal model configurations.

From an implementation perspective, the observed 6x inference-time penalty likely stems from sparse Clebsch-Gordan tensor contractions and Python overheads rather than fundamental algorithmic limitations. While this project focused on understanding the mathematical model construction, future research would benefit from leveraging highly optimized pre-implemented libraries such as e3nn or specialized CUDA kernels for spherical harmonic operations. Such optimizations could dramatically reduce the computational overhead and make equivariant architectures more practical for real-time applications, potentially changing the cost-benefit analysis in favor of equivariant approaches.

Another avenue for improvement lies in addressing the expressiveness gap between architectures. The Basic GNN employs polynomial edge features up to degree 2, while the Equivariant GNN restricts itself to degree-2 spherical harmonics, potentially limiting its representational capacity. Future work could explore higher-order spherical harmonic representations or richer

learnable steering attributes to close this expressiveness gap without sacrificing equivariance guarantees. Additionally, implementing more sophisticated equivariant nonlinearities beyond the simple norm-based activations used in this study could enhance the model's ability to capture complex geometric relationships while maintaining theoretical constraints.

6 Conclusion

This work systematically investigated the value of embedding $SE(3)$ symmetry directly into graph neural network (GNN) architectures for the task of estimating mesh centroids from single noisy LiDAR point clouds. Starting from a rigorous theoretical treatment of group representations and their realisation through spherical harmonics and Clebsch-Gordan products, we derived a strictly rotation-and translation–equivariant message–passing network. A non-equivariant counterpart with identical geometric feature access and a data-augmented variant served as fair baselines. All implementations were evaluated on an extensive ShapeNet–based benchmark comprising more than 10 500 test samples.

Key findings. The **Equivariant GNN** achieved an L^2 centroid error that is statistically indistinguishable from the best non-equivariant model (**Basic GNN + Augmentation**) while guaranteeing zero equivariance error by construction. Compared with the plain **Basic GNN**, both equivariant weight sharing and rotational data augmentation yielded sizeable accuracy improvements, confirming that respecting symmetry, either through theoretical guarantees or implicit data augmentation, is indeed beneficial for this regression task.

Trade–offs. Perfect equivariance was attained at a moderate parameter increase (21 %) and a six-fold inference-time penalty. The latter is primarily attributable to the sparse and Python-level implementation of Clebsch-Gordan tensor contractions and therefore likely to diminish with dedicated GPU kernels or highly optimised libraries (e.g. `e3nn`). Training dynamics further suggest that architectural equivariance stabilises learning and mitigates overfitting, an advantage that may outweigh the runtime overhead in data-scarce or safety-critical scenarios.

Limitations. The present study intentionally constrained the scope to relatively small networks (~8500 parameters) and second-order spherical harmonics. More expressive equivariant layers, deeper multiplicities, or higher-order harmonic expansions were not explored and could reveal larger accuracy gaps. Moreover, hyper-parameter tuning for the equivariant model was limited due to computational cost, and the Clebsch-Gordan product was applied only once per message-passing step, potentially restricting representational capacity.

Implications. Our empirical evidence challenges the notion that equivariance necessarily improves task performance in our area: when ample data and extensive augmentation are available, a carefully engineered non-equivariant network can match equivariant accuracy. Nevertheless, the mathematical guarantee of symmetry preservation offers reproducibility, interpretability, and potential energy savings by eliminating the need for augmentation-all highly desirable properties for robotics, medical imaging, and sustainability-oriented applications.

Future work. Promising directions include (i) exploiting recently proposed $\mathcal{O}(L^3)$ Gaunt-based accelerations to enable higher-order tensor interactions, (ii) performing a controlled data-efficiency study to quantify how quickly equivariant and non-equivariant models degrade under reduced training data, (iii) integrating more powerful equivariant nonlinearities and multi-layer Clebsch–Gordan blocks, and (iv) porting the implementation to optimised libraries to

reassess the computational cost-benefit balance. Such investigations will help clarify in which regimes architectural equivariance delivers the greatest practical return.

Though we note that performance in the use-case of geometric regression of point clouds was not provably better with EGGNs, we note that further research, extended resources and architectural exploration is needed for a definitive conclusion with statistical significance. In summary, we demonstrated that enforcing symmetry at the architectural level achieves competitive accuracy with perfect geometric consistency and offers some advantages in training stability and reliability. Equivariant GNNs constitute a principled and increasingly practical tool for 3-D perception tasks where respecting physical symmetries is paramount.

7 Appendix

7.1 GitHub Repository

Code and configuration files are available on our GitHub repository https://github.com/jonathantybirk/equivarian mesh-centroid-estimation.

7.2 Additional Results

Table 2: Performance, Equivariance, and Inference Time Rankings (Lower is Better)

Model	Performance Score	Equivariance Error	Inference Time (ms)
Equivariant GNN	0.070306 ± 0.053856	0.000000 ± 0.000000	248.9 ± 112.6
Basic GNN + Augmentation	0.070640 ± 0.054294	0.002694 ± 0.002176	40.8 ± 5.4
Basic GNN	0.075149 ± 0.055868	0.023267 ± 0.019843	43.2 ± 25.4
Baseline	0.107317 ± 0.068083	N/A	0.5 ± 2.5

Table 3: Performance Statistics (95% Confidence Intervals)

Model	Mean L2 Dist	95% CI	p-value	n
Baseline	0.107317	[0.106020, 0.108614]	< 0.001	10585
Basic GNN	0.075149	[0.074085, 0.076214]	< 0.001	10585
Basic GNN + Augmentation	0.070640	[0.069605, 0.071674]	< 0.001	10585
Equivariant GNN	0.070306	[0.069280, 0.071332]	< 0.001	10585

Table 4: Equivariance Statistics (95% Confidence Intervals)

Model	Mean Eq Error	95% CI	p-value	n
Baseline	N/A	N/A	N/A	N/A
Basic GNN	0.023267	[0.022889, 0.023645]	< 0.001	10585
Basic GNN + Augmentation	0.002694	[0.002653, 0.002736]	< 0.001	10585
Equivariant GNN	≈ 0 (perfect)	$[\approx 0, \approx 0]$	Perfect Eq	10585

7.3 Motivating Example: Equivariant MLP Level 1 (3D tensor) to Level 1 (3D tensor)

We first present a motivating example with a simplified problem to enhance understanding. We construct a simple MLP layer for message passing, mapping level 1 vectors to level 1 vectors. As this is a linear mapping from $\mathbb{R}^3 \to \mathbb{R}^3$, without loss of generality, we describe the function as:

$$f(x) = K(x) \cdot x, \quad K \in \mathbb{R}^{3\times 3}.$$

The equivariance of the function f is defined by:

$$f(Rx) = Rf(x), \quad \forall R \in SO(3), \forall x \in \mathbb{R}^3.$$

$$f(Rx) = K(Rx) \cdot Rx.$$

Table 5: Inference Time Statistics (95% Confidence Intervals)

Model	Mean Batch Time (ms)	95% CI (ms)	Std (ms)	n
Baseline	0.5	[0.105, 0.876]	2.5	166
Basic GNN	43.2	[39.270, 47.045]	25.4	166
Basic GNN + Augmentation	40.8	[39.984, 41.642]	5.4	166
Equivariant GNN	248.9	[231.699, 266.200]	112.6	166

Table 6: Pairwise Paired T-Tests (Performance)

Comparing models pairwise (paired t-test on same test samples)				
Model 1	Model 2	Mean Diff	p-value	Significant
Baseline	Basic GNN	0.032168	< 0.001	Yes
Baseline	Basic GNN + Augmentation	0.036677	< 0.001	Yes
Baseline	Equivariant GNN	0.037011	< 0.001	Yes
Basic GNN	Basic GNN + Augmentation	0.004510	1.32×10^{-98}	Yes
Basic GNN	Equivariant GNN	0.004843	6.11×10^{-56}	Yes
Basic GNN + Augmentation	Equivariant GNN	0.000334	0.218	No

Table 7: Pairwise Paired T-Tests (Equivariance)

Comparing models pairwise (paired t-test on same test samples)				
Model 1	Model 2	Mean Diff	p-value	Significant
Baseline	Basic GNN	N/A	N/A	N/A
Baseline	Basic GNN + Augmentation	N/A	N/A	N/A
Baseline	Equivariant GNN	N/A	N/A	N/A
Basic GNN	Basic GNN + Augmentation	0.020572	< 0.001	Yes
Basic GNN	Equivariant GNN	0.023267	< 0.001	Yes
Basic GNN + Augmentation	Equivariant GNN	0.002694	< 0.001	Yes

Table 8: Model Ranking Summary

Ranking Category	Model (Score)
Performance Ranking (L2 Distance)	
1st (Best)	Equivariant GNN (0.070306)
2nd	Basic GNN + Augmentation (0.070640)
3rd	Basic GNN (0.075149)
4th (Worst)	Baseline (0.107317)
Equivariance Ranking (Error)	
1st (Perfect)	Equivariant GNN (0.000000)
2nd	Basic GNN + Augmentation (0.002694)
3rd	Basic GNN (0.023267)
N/A	Baseline (not equivariant)
Parameter Count Ranking	
1st (Fewest)	Baseline (0 parameters)
2nd	Basic GNN (7,041 parameters)
3rd	Basic GNN + Augmentation (7,041 parameters)
4th (Most)	Equivariant GNN (8,531 parameters)

Thus:

$$K(Rx) \cdot Rx = RK(x)x.$$

Since this must hold for all x, we conclude:

$$K(Rx)R = RK(x) \quad \Leftrightarrow \quad K(Rx) = RK(x)R^{-1}.$$

7.3.1 Kernel Form via Polar Decomposition

Express x in polar form:

$$x = R_x \, r \, e_z, \quad e_z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

This leads to the identity:

$$K(x) = K(R_x r e_z) = R_x K(r e_z) R_x^{-1} = R_x K_z(r) R_x^{-1}.$$

However, $R_x \in SO(3)$ is not uniquely defined for a given x, since a vector in $\mathbb{R}^3$ has only a direction (S^2) and no rotation around its own axis. This makes R_x ill-defined. To see the issue concretely, consider the following:

Let $x = \frac{\sqrt{2}}{2} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$, constructed by a 45° rotation around the x-axis followed by a 90° rotation around the y-axis from e_z. Let R_z be a 30° rotation about the z-axis. The rotated vector Rx can also be expressed via a different sequence, resulting in a different R_{Rx}. Then,

$$K(x) = R_x K_z R_x^T,$$

$$K(Rx) = RK(x)R^T = RR_x K_z R_x^T R^T,$$

$$K(Rx) = R_{Rx} K_z R_{Rx}^T$$

may not match due to differing z-rotations in R_x and R_{Rx}.

To resolve this, we must discover that $K_z(r)$ be invariant under z-axis rotations:

$$K_z(r) = K(re_z) = K(R_z re_z) = R_z K(re_z) R_z^{-1}.$$

This implies:

$$K_z(r) R_z = R_z K_z(r).$$

Let:

$$R_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad K_z(r) = \begin{pmatrix} A(r) & b(r) \\ c(r)^T & d(r) \end{pmatrix}.$$

Then:

$$K_z(r)R_z = \begin{pmatrix} A(r)R_2(\theta) & b(r) \\ c(r)^T R_2(\theta) & d(r) \end{pmatrix}, \quad R_z K_z(r) = \begin{pmatrix} R_2(\theta)A(r) & R_2(\theta)b(r) \\ c(r)^T & d(r) \end{pmatrix}.$$

Equating terms yields:

$$\begin{aligned} R_2(\theta)A(r) &= A(r)R_2(\theta), \\ R_2(\theta)b(r) &= b(r), \\ c(r)^T &= c(r)^T R_2(\theta), \\ d(r) &= d(r). \end{aligned}$$

The second and third imply $b(r) = 0$ and $c(r) = 0$. The first implies:

$$A(r) = \begin{pmatrix} a & b \\ -b & a \end{pmatrix}.$$

Thus,

$$K_z(r) = \begin{pmatrix} a & b & 0 \\ -b & a & 0 \\ 0 & 0 & d \end{pmatrix}.$$

So

$$K(x) = R_x \begin{pmatrix} a & b & 0 \\ -b & a & 0 \\ 0 & 0 & d \end{pmatrix} R_x^T.$$

This can be rewritten as:

$$K(x) = R_x \left(a(I - e_z e_z^T) + d e_z e_z^T + b \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \right) R_x^T.$$

Now use:

$$K(x) = aR_x(I - e_z e_z^T)R_x^T + dR_x e_z e_z^T R_x^T + bR_x \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} R_x^T,$$

$$= aR_x I R_x^T - aR_x e_z e_z^T R_x^T + dR_x e_z e_z^T R_x^T + bR_x \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} R_x^T.$$

Given $x = R_x r e_z$, then $x/r = R_x e_z$, so:

$$K(x) = aI - a\frac{x}{r}\left(\frac{x}{r}\right)^T + d\frac{x}{r}\left(\frac{x}{r}\right)^T + bR_x \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} R_x^T,$$

$$= aI + (d-a)\frac{x}{r}\left(\frac{x}{r}\right)^T + bR_x \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} R_x^T.$$

Define $c = d - a$,

$$K(x) = aI + c\frac{x}{r}\left(\frac{x}{r}\right)^T + bR_x[-e_z]_\times R_x^T.$$

Note the definition of the cross-product matrix:

$$[-e_z]_\times = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad [z]_\times x = z \times x.$$

Since $[R_x e_z]_\times = R_x[-e_z]_\times R_x^T$, we get:

$$K(x) = aI + c\frac{x}{r}\left(\frac{x}{r}\right)^T + b_2[x/r]_\times.$$

Note that K(x)*x reduces to $k \cdot x$ for some constant k, reducing the amount of weights, which is why we'd usually use a different parameter, called a steering attribute, for k. I.e. K(a)*x. We will later define what we use for a

Equivariance Check

We still need to show it is indeed equivariant:

$$\begin{aligned}
K(Rx) &= aI + c\frac{Rx}{r}\left(\frac{Rx}{r}\right)^T + b_2\left[\frac{Rx}{r}\right]_\times \\
&= aRR^{-1} + c\frac{Rx}{r}\left(\frac{x}{r}\right)^T R^T + b_2 R\left[\frac{x}{r}\right]_\times R^T \\
&= aRIR^{-1} + c\frac{Rx}{r}\left(\frac{x}{r}\right)^T R^T + b_2 R\left[\frac{x}{r}\right]_\times R^T \\
&= R\left(aI + c\frac{x}{r}\left(\frac{x}{r}\right)^T + b_2\left[\frac{x}{r}\right]_\times\right)R^T \\
&= RK(x)R^T
\end{aligned}$$

7.3.2 Equivariant Nonlinearity

Let $\sigma(Rx) = R\sigma(x)$. Define:

$$\sigma(x) = \mathrm{ReLU}(\|x\| - b)\cdot\frac{x}{\|x\|},$$

Then:

$$\begin{aligned}
\sigma(Rx) &= \mathrm{ReLU}(\|Rx\| - b)\cdot\frac{Rx}{\|Rx\|} \\
&= \mathrm{ReLU}(\|x\| - b)\cdot\frac{Rx}{\|x\|} = R\sigma(x).
\end{aligned}$$

7.3.3 Full Layer

A full (simple) equivariant layer can be written:

$$h_i' = \sum_j \sigma(K_j(x)h_j).$$

We want to extend this theory to all combinations to and from all types of tensors. To achieve this, we will define our attributes in the *spherical harmonic basis*. While the hidden layer vectors can simply be seen as numbers which rotate in a defined way, spherical harmonics provide a way to interpret these vectors and provides a way to multiply them meaningfully and equivariantly, which also provides a way to for different dimensional vectors to interact. Meanwhile, spherical harmonics also provide a way to represent information from physical vectors in a meaningful way in higher dimensions. After defining the spherical harmonics, we can show their usefullness in an example neural network.

7.4 EGNN Theory's Connection to Motivating Example

We now explicitly connect the previously introduced Clebsch-Gordan (CG) product to the motivating example that used linear algebra. We begin by considering a general CG expansion mapping type-1 tensors to type-1 tensors:

$$\tilde{\mathbf{h}}^{(1)} = w_{01}(a^{(0)} \otimes_{\mathrm{cg}} \mathbf{h}^{(1)})^{(1)} + w_{11}(a^{(1)} \otimes_{\mathrm{cg}} \mathbf{h}^{(1)})^{(1)} + w_{21}(a^{(2)} \otimes_{\mathrm{cg}} \mathbf{h}^{(1)})^{(1)}.$$

Scalar–Vector Coupling ($\mathbf{0} \otimes \mathbf{1} \to \mathbf{1}$) Since a scalar (type-0) has a single component, the Clebsch-Gordan coefficient is straightforward:

$$(a^{(0)} \otimes_{\mathrm{cg}} \mathbf{h}^{(1)})^{(1)} = a^{(0)}\mathbf{h}^{(1)}.$$

Vector–Vector Coupling ($\mathbf{1} \otimes \mathbf{1} \to \mathbf{1}$) For the vector-vector coupling, standard CG coefficients yield a cross-product relationship in Cartesian coordinates:

$$(a^{(1)} \otimes_{\text{cg}} \mathbf{h}^{(1)})^{(1)} = [a^{(1)}]_\times \mathbf{h}^{(1)},$$

where the cross-product matrix is defined as:

$$[a^{(1)}]_\times = \begin{pmatrix} 0 & -a_3^{(1)} & a_2^{(1)} \\ a_3^{(1)} & 0 & -a_1^{(1)} \\ -a_2^{(1)} & a_1^{(1)} & 0 \end{pmatrix}.$$

Quadrupole-Vector Coupling ($\mathbf{2} \otimes \mathbf{1} \to \mathbf{1}$) For the coupling between type-2 and type-1 tensors, the nonzero CG coefficients result in a symmetric, traceless tensor given by:

$$S(a^{(2)}) = \begin{pmatrix} s_{xx} & s_{xy} & s_{xz} \\ s_{xy} & s_{yy} & s_{yz} \\ s_{xz} & s_{yz} & -(s_{xx} + s_{yy}) \end{pmatrix}, \quad \text{tr}(S(a^{(2)})) = 0,$$

where components relate to spherical harmonics through:

$$\begin{aligned} s_{xx} &= \tfrac{1}{2}(a_2^{(2)} + a_{-2}^{(2)}), & s_{yy} &= -\tfrac{1}{2}(a_2^{(2)} + a_{-2}^{(2)}), \\ s_{xy} &= \tfrac{1}{2}(a_{-2}^{(2)} - a_2^{(2)}), & s_{xz} &= \tfrac{1}{\sqrt{2}}(a_1^{(2)} - a_{-1}^{(2)}), \\ s_{yz} &= -\tfrac{1}{\sqrt{2}}(a_1^{(2)} + a_{-1}^{(2)}). \end{aligned}$$

Thus,

$$(a^{(2)} \otimes_{\text{cg}} \mathbf{h}^{(1)})^{(1)} = S(a^{(2)})\mathbf{h}^{(1)}.$$

Explicit Cartesian Form Combining the above expressions results in an explicit, purely Cartesian representation:

$$\tilde{\mathbf{h}} = w_0 a^{(0)}\mathbf{h} + w_1 [a^{(1)}]_\times \mathbf{h} + w_2 S(a^{(2)})\mathbf{h}.$$

Recalling the motivating example, the kernel acting on $\mathbf{h}$ is:

$$K(x)\mathbf{h} = (aI + b[\hat{x}]_\times + c\hat{x}\hat{x}^\top)\mathbf{h}, \quad \hat{x} = \frac{x}{\|x\|}.$$

Identifying the geometric steering channels as:

$$\begin{aligned} a^{(0)} &= a, \\ a^{(1)} &= \hat{x}, \\ S(a^{(2)}) &= \hat{x}\hat{x}^\top - \tfrac{1}{3}I, \end{aligned}$$

and setting weights $w_0 = a, w_1 = b, w_2 = c$, reproduces exactly the kernel structure from the motivating example.

References

[1] Hanjun Kang et al. “Equi-GSPR: Equivariant Graph Matching for Robust Point Cloud Registration”. In: *European Conference on Computer Vision (ECCV)*. 2024.

[2] Yijia Lin et al. “SE3ET: SE(3)-Equivariant Transformers for Low-Overlap Point Cloud Registration”. In: *arXiv preprint arXiv:2407.01234* (2024).

[3] Johannes Brandstetter et al. “Geometric and Physical Quantities Improve E(3) Equivariant Message Passing”. In: *International Conference on Learning Representations (ICLR)*. arXiv:2110.02905. 2022. URL: https://arxiv.org/abs/2110.02905.

[4] Nathaniel Thomas et al. *Tensor field networks: Rotation- and translation-equivariant neural networks for 3D point clouds*. 2018. arXiv: 1802.08219 [cs.LG]. URL: https://arxiv.org/abs/1802.08219.

[5] Shengjie Luo, Tianlang Chen, and Aditi S. Krishnapriyan. “Enabling Efficient Equivariant Operations in the Fourier Basis via Gaunt Tensor Products”. In: *International Conference on Learning Representations (ICLR)*. Project lead: Shengjie Luo. 2024. URL: https://arxiv.org/abs/2401.10216.

[6] Mario Geiger et al. *Euclidean neural networks: e3nn*. Version 0.5.0. Apr. 2022. DOI: 10.5281/zenodo.6459381. URL: https://doi.org/10.5281/zenodo.6459381.

[7] Maurice Weiler et al. *3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data*. 2018. arXiv: 1807.02547 [cs.LG]. URL: https://arxiv.org/abs/1807.02547.