Equivariant Geometric Scattering Networks via Vector Diffusion Wavelets

David R. Johnson[1] **Rishabh Anand**[2] **Smita Krishnaswamy**[3,4] **Michael Perlmutter**[5,1]

[1]Program in Computing, Boise State University [2]School of Medicine, Yale University
[3]Department of Computer Science, Yale University [4]Department of Genetics, Yale University
[5]Department of Mathematics, Boise State University

Abstract

We introduce a novel version of the geometric scattering transform for geometric graphs containing scalar and vector node features. This new scattering transform has desirable symmetries with respect to rigid-body roto-translations (i.e., $SE(3)$-equivariance) and may be incorporated into a geometric GNN framework. We empirically show that our equivariant scattering-based GNN achieves comparable performance to other equivariant message-passing-based GNNs at a fraction of the parameter count.

1 Introduction

The field of Geometric Deep Learning (GDL) [Bronstein et al., 2017, 2021], aims to extend the success of deep learning to data sets with geometric structure such as graphs and manifolds. Crucially, most GDL methods aim to represent the data points in a manner that respects the intrinsic symmetries of the data set. That is, two data points will be represented the same way if they differ only by an uninformative deformation such as the relabeling of the vertices of a graph or a global isometry of a manifold. *In this paper, we will focus on geometric graphs where the vertices lie in Euclidean space $\mathbb{R}^d$ and we have both scalar- and vector-valued node features.* Accordingly, following the lead of equivariant GNNs [Satorras et al., 2021, Batzner et al., 2022, Han et al., 2022, Duval et al., 2023], we will aim to produce a method which has desirable symmetries with respect to rigid motions in $\mathbb{R}^d$.

The most prominent success of GDL has been the rise of Graph Neural Networks (GNNs) [Kipf and Welling, 2017, Veličković et al., 2018, Xu et al., 2019, Khemani et al., 2024]. Most common GNNs are based on the *message-passing* paradigm in which each node is represented in a manner informed by its immediate neighbors, often by local averaging. However, despite their success, message-passing networks come with some well-known limitations. Because of the use of local averaging operations, the number of layers must be kept small (typically two or three) in order to prevent the *oversmoothing* problem. However, this introduces a new problem, *underreaching*, in which case the GNN is unable to capture global structure.

One possible solution to the oversmoothing vs. underreaching tradeoff is provided by the *geometric scattering transform* [Zou and Lerman, 2019, Gama et al., 2018, Gao et al., 2019] and associated GNNs [Min et al., 2020, 2021, Tong et al., 2024]). These networks rely on *diffusion wavelets* to capture the multiscale geometry of the graph in a single layer and have been shown not to suffer from oversmoothing [Wenkel et al., 2022]. Additionally, using these wavelets allows for sophisticated processing of the input features with comparatively few learnable parameters.

The purpose of this paper is to introduce a novel version of diffusion wavelets, and corresponding scattering networks, for geometric graphs with both scalar-valued and vector-valued node features. We will prove that our network has desirable symmetries with respect to rotations in

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: New Perspectives in Advancing Graph Machine Learning.

Euclidean space and also conduct numerical experiments showing that our method is able to achieve comparable performance to other equivariant GNNs with a substantially (often at least 90%) lower parameter count.

2 Background

Throughout, we will let $G = (V, E, w)$ be a weighted, undirected, connected graph, with weighted adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. We assume that we are given F features for each node which are stored in an $n \times F$ matrix $\mathbf{X} = \mathbf{X}^{(0)}$, so that the row, $\mathbf{X}[i, :]$, contains the features associated to v_i.

2.1 GNNs and Message Passing Networks

Most common GNNs follow the message passing paradigm in which, in each layer, the representation of each vertex is updated based on its current representation and the representations of its neighbors. As a simple illustration, we consider the Graph Convolutional Network (GCN) introduced in Kipf and Welling [2017]. In this network, the layerwise update rule is given by $\mathbf{X}^{(\ell+1)} = \sigma(\widehat{\mathbf{A}}\mathbf{X}^{(\ell)}\mathbf{\Theta}^{(\ell)})$. Here, $\widehat{\mathbf{A}}$ is an averaging type operator which aggregates information from neighboring vertices, $\mathbf{\Theta}^{(\ell)}$ is a learnable $F_\ell \times F_{\ell+1}$ weight matrix, (where F_ℓ is the number of hidden features in the ℓ-th layer, $F_0 = F$), and σ is an activation function.

It is known that the matrix $\widehat{\mathbf{A}}$ acts as a smoothing operator that averages the node features within local neighborhoods, promoting similar representations for adjacent vertices, which is a useful heuristic for, e.g., node-classification tasks on homophilous citation networks. However, after each layer, the representation of each node becomes progressively smoother. Therefore, the total number of layers must be kept small, typically two or three, to avoid *oversmoothing* [Nt and Maehara, 2019, Qureshi et al., 2023]. However, since each message-passing layer acts locally, this creates a new problem, *underreaching* [Lu et al., 2024], in which the network is unable to utilize long-range interactions, which may limit its effectiveness for tasks requiring the network to capture the global structure of the graph such as predicting binding affinity or total polar surface areas of a molecule [Li et al., 2024].

2.2 Diffusion Wavelets and Geometric Scattering Networks

The geometric scattering transform [Zou and Lerman, 2019, Gama et al., 2018, Gao et al., 2019] (and associated GNNs [Min et al., 2020, 2021, Tong et al., 2024]) provides an alternative to message-passing, which, as shown in Wenkel et al. [2022], allows one to circumvent the oversmoothing vs. underreaching tradeoff via the use of diffusion wavelets to extract multiscale information from the graph and the node features. Adopting the perspective of *graph signal processing* (GSP) [Shuman et al., 2013, Ortega et al., 2018], we will interpret columns $\mathbf{x}$ of the feature matrix $\mathbf{X}$ as *signals*, i.e., functions $\mathbf{x} : V \to \mathbb{R}$, where the $\mathbf{x}[i] = \mathbf{x}(v_i)$ is the value of $\mathbf{x}$ at v_i. We consider the lazy random walk matrix $\mathbf{P} = \frac{1}{2}(\mathbf{I} + \mathbf{D}^{-1}\mathbf{A})$[1], where $\mathbf{D} \in \mathbb{R}^{n \times n}$ is the diagonal degree matrix.

We will use $\mathbf{P}$ to construct *diffusion wavelets*. Letting J be a positive integer, we define diffusion wavelets $\mathcal{W}_J = \{\mathbf{\Psi}_j\}_{j=0}^{J} \cup \{\mathbf{\Phi}_J\}$ by

$$\mathbf{\Psi}_j = \mathbf{P}^{2^{j-1}} - \mathbf{P}^{2^j} = \mathbf{P}^{2^{j-1}}(\mathbf{I} - \mathbf{P}^{2^{j-1}}), \quad \text{for } 1 \leq j \leq J, \tag{1}$$

with $\mathbf{\Psi}_0 = \mathbf{I} - \mathbf{P}$ and $\mathbf{\Phi}_J = \mathbf{P}^{2^J}$. To understand these wavelets, observe that $\mathbf{\Psi}_1\mathbf{x}[i] = (\mathbf{P} - \mathbf{P}^2)\mathbf{x}[i]$, the i-th entry of $\mathbf{\Psi}_1\mathbf{x}$, can be interpreted as describing the differences in the behavior of a signal $\mathbf{x}$ with a one-hop neighborhood of v_i to the behavior of $\mathbf{x}$ in a two-hop neighborhood. Similarly, $\mathbf{\Psi}_0\mathbf{x} = (\mathbf{I} - \mathbf{P})\mathbf{x}$ describes how the behavior of $\mathbf{x}$ at each vertex differs from at its immediate neighbors. Collectively, the filter bank $\mathcal{W}_J = \{\mathbf{\Psi}_j\}_{j=0}^{J} \cup \{\mathbf{\Phi}_J\}$ acts as a multi-scale feature extractor where $\mathbf{\Phi}_J$ extracts global information from $\mathbf{x}$ (at scale 2^J) and each $\mathbf{\Psi}_j$ tracks changes across different scales. From the GSP perspective $\mathbf{\Phi}_J$, is interpreted as a low-pass filter and the $\mathbf{\Psi}_j$ are interpreted as band-pass filters that highlight different frequency bands.

[1] Various versions of the scattering transform [Gama et al., 2018, Gao et al., 2019, Perlmutter et al., 2023] use different normalizations of the diffusion matrix. Here, we focus on the row-normalized diffusion operator. However, all of our theory may be readily adapted to other normalizations following the analysis provided in Perlmutter et al. [2023].

Given the bank of diffusion wavelets $\mathcal{W}_J$, the geometric scattering transform is a non-linear multi-layer feature extractor. It defines *scattering coefficients* via alternating sequences of linear maps (chosen to be wavelets) and entrywise activations, analogous to a neural network. Formally, first- and second-order scattering coefficients are defined by

$$\mathcal{U}[j]\mathbf{x}(v) = \sigma(\mathbf{\Psi}_j\mathbf{x}(v)), \quad \text{and} \quad \mathcal{U}[j,j']\mathbf{x}(v) = \mathcal{U}[j']\mathcal{U}[j]\mathbf{x}(v) = \sigma(\mathbf{\Psi}_j\sigma(\mathbf{\Psi}_j\mathbf{x}(v))), \tag{2}$$

for $0 \leq j \leq j' \leq J$, where σ is an activation function. Further, if desired, m-th order scattering coefficients for $m \geq 3$ can be defined similarly by $\mathcal{U}[j_1, j_2, \ldots, j_m]\mathbf{x}(v) = \mathcal{U}[j_m]\mathcal{U}[j_1, \ldots, j_{m-1}]\mathbf{x}(v)$.

When used for node-level tasks, one can view the collection of all $\mathcal{U}[j]\mathbf{x}$ and $\mathcal{U}[j,j']\mathbf{x}$ as a new set of node-features which can then be fed into a prediction head, typically an MLP. For graph-level tasks, one typically first performs a global aggregation by computing scattering moments of the form,

$$\mathcal{S}[j,q]\mathbf{x} = \|\mathcal{U}[j]\mathbf{x}\|_q^q = \sum_{i=1}^n |\mathcal{U}[j]\mathbf{x}(v_i)|^q, \quad \text{and} \quad \mathcal{S}[j,j',q]\mathbf{x} = \|\mathcal{U}[j,j']\mathbf{x}\|_q^q = \sum_{i=1}^n |\mathcal{U}[j,j']\mathbf{x}(v_i)|^q,$$

before applying a readout prediction head. Additionally, for both graph-level and node-level tasks, works such as Min et al. [2020, 2021], Tong et al. [2024], Johnson et al. [2025a], Wenkel et al. [2024] have incorporated the scattering transform into high-performing, end-to-end differentiable GNNs.

2.3 Equivariant GNNs

Traditionally, both the message-passing networks and the geometric scattering transforms have mostly focused on abstract graphs, where the v_i are merely thought of as elements of a set. However, for many applications, it is more natural to consider geometric graphs, where the vertices are points in some Euclidean space $\mathbb{R}^d$ and we are given both scalar-valued and vector-valued node features. For instance, with molecular graphs, the vertices may correspond to atoms with known coordinates in $\mathbb{R}^3$. Typically, one has access to both scalar-valued features, such as an encoding of the atom types, as well as vector-valued features such as atomic positions or velocities.

Formally, we define a geometric graph as a weighted, undirected graph $G = (V, E, w)$, where the vertices $v_1, \ldots, v_n$ lie in $\mathbb{R}^d$, and will denote the i-th coordinate of the vertex v_j by $v_j[i]$. We will assume that we are given scalar-valued signals $\mathbf{x}_i : V \to \mathbb{R}$, $1 \leq i \leq F_{\text{scalar}}$, and vector-valued signals $\mathbf{w}_i : V \to \mathbb{R}^d$, $1 \leq i \leq F_{\text{vec}}$. Analogous to Section 2.1, we note that we may organize the scalar-valued signals into an $n \times F_{\text{scalar}}$ matrix $\mathbf{X}$; and similarly, we may organize the vector-valued features into an $n \times F_{\text{vec}} \times d$ tensor $\mathbf{W}$. However, for most of this paper, it will be convenient to work with individual signals $\mathbf{x}_i$ and $\mathbf{w}_i$.

When designing neural networks for geometric graphs, with both scalar-valued and vector-valued signals, it is important to preserve their intrinsic symmetries with respect to rotations. To understand this, consider what should happen if we rotate the entire system by some rotation matrix, $\mathbf{R} \in \mathbb{R}^{d \times d}$. Let $\mathbf{x} = \mathbf{x}^{(0)}$ denote an initial scalar-valued signal, and let $\mathbf{w} = \mathbf{w}^{(0)}$ denote an initial vector-valued singal. Let $\mathbf{x}^{(\ell)}$ and $\mathbf{w}^{(\ell)}$ denote our transformed representation after ℓ layers. Rotations do not affect scalar-valued signals such as atomic types. Thus, we aim to ensure that $\widetilde{\mathbf{x}}^{(\ell)} = \mathbf{x}^{(\ell)}$, where $\widetilde{\mathbf{x}}^{(\ell)}$ is analog of $\mathbf{x}^{(\ell)}$ but for the rotated system. This property is referred to as *rotational invariance*. However, for a vector-valued signals, such as the position or velocity of each vertex, rotating the system will also rotate the value of the feature. Therefore, we aim to ensure that $\widetilde{\mathbf{w}}^{(\ell)} = \mathbf{R}\mathbf{w}^{(\ell)}$. This property is referred to as *rotational equivariance*.

Equivariant GNNs [Satorras et al., 2021, Batzner et al., 2022, Han et al., 2022, Duval et al., 2023] are designed to possess these desirable properties. As an illustrative example, we consider EGNN [Satorras et al., 2021], and let $\mathbf{X}^{(\ell)}$ (and $\mathbf{W}^{(\ell)}$) be the matrix (tensor) of scalar-valued (vector-valued) features after the ℓ-th layer. E-GNN first computes messages $m_{i,j}$ between adjacent nodes v_i and v_j that depend on $\mathbf{X}^{(\ell)}[i,:]$ and $\mathbf{X}^{(\ell)}[j,:]$ as well as the set of distances between the vector-valued features at v_i and v_j, $\{\|\mathbf{W}^{(\ell)}[i,k,:] - \mathbf{W}^{(\ell)}[j,k,:]\|_2^2\}_{k=1}^{F_{\text{vec}}}$, which implies that $m_{i,j}$ is invariant under rotations.[2] It then (i) updates the scalar-valued node features to produce new features so that $\mathbf{X}^{(\ell+1)}[i,:]$ depends on the previous value $\mathbf{X}^{(\ell)}[i,:]$ as well as the messages $m_{i,j}$ and (ii) updates

[2]We present a slightly more general formulation than Satorras et al. [2021], which primarily focuses on the case when there is a single vector-valued signal representing the coordinates of the vertices in 3D space.

the vector-valued features to $\mathbf{W}^{(\ell+1)}[i,:,:]$ that depends on the previous value $\mathbf{W}^{(\ell)}[i,:,:]$ as well as "updates" of the form $\phi(m_{i,j})\left(\mathbf{W}^{(\ell)}[i,:,:]-\mathbf{W}^{(\ell)}[j,:,:]\right)$. Here ϕ is a scalar-valued function, which allows one to verify that these updates will commute with rotations.

Additionally, we note that unlike several previous works that introduce geometric scattering transforms [Chew et al., 2022, 2024, Johnson et al., 2025a] or more general GNNs [Wang et al., 2023, 2024, Chew et al., 2023] for geometric graphs, we do not necessarily assume that our graph is a discretization of an underlying manifold. Furthermore these works did not consider vector-valued node features.

3 Vector Diffusion Wavelets and Equivariant Scattering Transforms

We construct a novel form of diffusion wavelets for vector-valued signals. This will allow us to define equivariant geometric scattering transforms for geometric graphs with both vector-valued and scalar-valued signals. As in Section 2.3, we will assume that we are given F_{scalar} scalar-valued signals $\mathbf{x}_i : V \to \mathbb{R}$, $1 \leq i \leq F_{\text{scalar}}$, and F_{vec} vector-valued signals $\mathbf{w}_i : V \to \mathbb{R}^d$, $1 \leq i \leq F_{\text{vec}}$. Importantly, we note that our framework is not restricted to the case that $d = 3$, although this is the case for many real-world applications. As in Section 2.2, when convenient, we will identify signals with vectors. We will let $\mathbf{w}_i[j] = \mathbf{w}_i(v_j) \in \mathbb{R}^d$ denote the value of the signal $\mathbf{w}_i$ at each point and we will view $\mathbf{w}_i$ as a vector in $\mathbb{R}^{nd}$ given by $\mathbf{w}_i = [\mathbf{w}_i[1]^\top, \ldots, \mathbf{w}_i[n]^T]^\top = [\mathbf{w}_i[1][1], \ldots, \mathbf{w}_i[1][d], \ldots, \mathbf{w}_i[n][1], \ldots, \mathbf{w}_i[n][d]]^\top$.

In order to process the scalar-valued features $\mathbf{x}_i$, we may use the scalar-valued geometric scattering transform introduced in Section 2.2. However, in order to process the vector-valued features, we need to introduce a new form of geometric scattering. Towards this end, we will first define a vector diffusion matrix $\mathbf{Q} \in \mathbb{R}^{nd \times nd}$ using methods inspired by vector diffusion maps [Singer and Wu, 2012]. We will then use $\mathbf{Q}$ to define vector diffusion wavelets and associated scattering networks.

For each node $v_i \in \mathbb{R}^d$, we will construct a local basis $\{\mathbf{u}_{i,1}, \ldots, \mathbf{u}_{i,d}\}$, which will provide a local coordinate system for each node relative to its neighbors. To build this local basis, we let $\mathcal{N}_{v_i} = \{v_j \in V : \{v_i, v_j\} \in E\}$ denote the one-hop neighborhood of v_i and let $n_i = |\mathcal{N}_{v_i}|$ denote the number of neighbors.[3] We then define a relative distance matrix $\mathbf{C}_i \in \mathbb{R}^{d \times n_i}$, whose columns are by $(v_{i_j} - v_i)$ where v_{i_j} is the j-th neighbor of v_i. In order to give more weight to nearer neighbors, we then rescale $\mathbf{C}_i$ by defining $\mathbf{B}_i = \mathbf{C}_i \mathbf{D}_i$, where $\mathbf{D}_i$ is a diagonal matrix defined by $\mathbf{D}_i[j,j] = \sqrt{K_\epsilon(v_i, v_{i_j})}$, with $K_\epsilon(\cdot,\cdot)$ being a Guassian kernel with scale ϵ, $K(v_i, v_{i_j}) = \exp(-\|v_i - v_{i_j}\|_2^2/\epsilon)$.

We next compute the singular value decomposition (SVD), $\mathbf{B}_i = \mathbf{U}_i \mathbf{\Sigma}_i \mathbf{V}_i^\top$. We note that, by the definition of the SVD, the matrix $\mathbf{U}_i$ is unitary, and so its columns $\{\mathbf{u}_{i,1}, \ldots, \mathbf{u}_{i,d}\}$ form an orthonormal basis for $\mathbb{R}^d$, which we interpret as a defining a local coordinate system centered around node v_i. We will assume throughout that each of the singular values, i.e., the diagonal entries of Σ_i, is in decreasing order and that none of the singular values have multiplicity greater than one.

For all i and j, we define $\mathcal{O}_{i,j} = \mathbf{U}_i \mathbf{U}_j^\top \in \mathbb{R}^{d \times d}$ which *shifts between the local coordinate systems* centered at v_i and v_j. We note that the SVD is only unique up to sign-flips. That is, one may obtain a different SVD, $\mathbf{B}_i' = \mathbf{U}_i' \mathbf{\Sigma}_i (\mathbf{V}')_i^\top$ by replacing both $\mathbf{u}_{i,k}$ and $\mathbf{v}_{i,k}$ with $\mathbf{u}_{i,k}' = -\mathbf{u}_{i,k}$ and $\mathbf{v}_{i,k}' = -\mathbf{v}_{i,k}$ for any fixed k. However, in order to ensure that the matrices $\mathcal{O}_{i,j}$ are well-defined (i.e., independent of these sign choices), we will employ a *sign-flipping* technique detailed in Appendix C.

Using these $\mathcal{O}_{i,j}$, we define a $nd \times nd$ vector-valued diffusion matrix in block form by

$$\mathbf{Q}[i,j] = \mathbf{P}[i,j]\mathcal{O}_{i,j} \in \mathbb{R}^{d \times d}$$

and observe that by construction, we have $\mathbf{Q}[i,j] = \mathbf{0}$ unless $\{v_i, v_j\} \in E$. This implies we will only need to compute $\mathcal{O}_{i,j}$ if $\{v_i, v_j\} \in E$ or $i = j$, which significantly reduces our computational cost.

Given $\mathbf{Q}$, we may then define vector diffusion wavelets and scattering coefficients analogous to (1) and (2). Specifically, we define $\widetilde{\mathcal{W}}_J = \{\widetilde{\mathbf{\Psi}}_j\}_{j=0}^J \cup \{\widetilde{\mathbf{\Phi}}_J\}$ by $\widetilde{\mathbf{\Psi}}_0 = \mathbf{I} - \mathbf{Q}$, $\widetilde{\mathbf{\Phi}}_J = \mathbf{Q}^{2^J}$, and

$$\widetilde{\mathbf{\Psi}}_j = \mathbf{Q}^{2^{j-1}} - \mathbf{Q}^{2^j} = \mathbf{Q}^{2^{j-1}}(\mathbf{I} - \mathbf{Q}^{2^{j-1}}), \quad 1 \leq j \leq J. \tag{3}$$

[3] We will assume that we have $n_i \geq d$ for all i. Otherwise, we will modify the edge set by adding edges between each v_i and its nearest neighbors until $\deg(v_i) \geq d$.

We then define the vector-valued scattering coefficients of a vector-valued signal $\mathbf{w} : V \to \mathbb{R}^d$ by

$$\widetilde{\mathcal{U}}[j]\mathbf{w}(v) = \sigma(\widetilde{\mathbf{\Psi}}_j \mathbf{w}(v)), \quad \text{and} \quad \widetilde{\mathcal{U}}[j,j']\mathbf{w}(v) = \widetilde{\mathcal{U}}[j']\widetilde{\mathcal{U}}[j]\mathbf{w}(v) = \sigma(\widetilde{\mathbf{\Psi}}_{j'}\sigma(\widetilde{\mathbf{\Psi}}_j \mathbf{w}(v))). \tag{4}$$

As in the scalar-feature case, if one wishes, m-th order scattering coefficients for $m \geq 3$ can be defined similarly the rule $\widetilde{\mathcal{U}}[j_1, j_2, \ldots, j_m]\mathbf{w}(v) = \widetilde{\mathcal{U}}[j_m]\widetilde{\mathcal{U}}[j_1, \ldots, j_{m-1}]\mathbf{w}(v)$. Additionally, when it is convenient to think of the signal $\mathbf{w}$ as a vector in $\mathbb{R}^{nd}$, we will write the first-order coefficients as $\widetilde{\mathcal{U}}[j]\mathbf{w} = \sigma(\widetilde{\mathbf{\Psi}}_j \mathbf{w})$ instead of $\widetilde{\mathcal{U}}[j]\mathbf{w}(v) = \sigma(\widetilde{\mathbf{\Psi}}_j \mathbf{w}(v))$ (and similarly with the higher-order coefficients).

3.1 Theoretical Results

The following result establishes frame bounds for the vector diffusion wavelets, similar to analogous results for the scalar-valued diffusion wavelets [Gama et al., 2018, Perlmutter et al., 2023]. It implies that vector diffusion wavelet transform may be stably inverted and is robust to additive noise.[4]

Theorem 3.1. *There exists a universal constant $c > 0$ such that for all $\mathbf{w} \in \mathbb{R}^{nd}$ we have*

$$c\,\frac{d_{\min}}{d_{\max}}\|\mathbf{w}\|_2^2 \leq \left\|\widetilde{\mathcal{W}}_J \mathbf{w}\right\|_2^2 := \sum_{j=0}^{J} \left\|\widetilde{\mathbf{\Psi}}_j \mathbf{w}\right\|_2^2 + \left\|\widetilde{\mathbf{\Phi}}_J \mathbf{w}\right\|_2^2 \leq \frac{d_{\max}}{d_{\min}}\|\mathbf{w}\|_2^2.$$

We next show that our vector diffusion wavelets and scattering coefficients are equivariant to the actions of the Special Orthogonal group, $SO(d)$, i.e., the set of all $d \times d$ rotation matrices. Will assume that our entire system has been subjected to the same global rotation $\mathbf{R} \in SO(d)$ and use bars to denote objects in the rotated coordinate system so that e.g., $\overline{v}_i = (\overline{v}_i[1], \overline{v}_i[2], \overline{v}_i[3])^\top$ denotes the position of the i-th vertex in the rotated system and $\overline{\mathbf{Q}}$ denotes the vector diffusion matrix constructed from the $\overline{v}_i$. We observe that we have $\overline{v}_i = \mathbf{R}v_i$, and, motivated by examples such as when our vector-valued-node features are either the coordinates or the velocities of each vertex, we will assume that rotating the entire system rotates the values of vector-valued features so that we have $\overline{\mathbf{w}}_f = \mathbf{R} \cdot \mathbf{w}$, $\mathbf{R} \cdot \mathbf{w}_f = [(\mathbf{R}\mathbf{w}_f(v_1)^\top, \ldots, (\mathbf{R}\mathbf{w}_f(v_n))^\top]^\top$. Furthermore, we will assume that rotations do not change the connectivity structure of the graph so that we have $\overline{\mathbf{A}} = \mathbf{A}$. The following theorems establish the equivariance of vector diffusion wavelets and associated scattering coefficients. We also note Figure 2 in the appendix, which provides a visual illustration of Theorem 3.2.

Theorem 3.2 (Wavelet Equivariance)**.** *For any vector-valued node feature $\mathbf{w}$, we have, a*

$$\overline{\widetilde{\mathbf{\Psi}}}_j \overline{\mathbf{w}} = \overline{\widetilde{\mathbf{\Psi}}}_j(\mathbf{R} \cdot \mathbf{w}) = \mathbf{R} \cdot \widetilde{\mathbf{\Psi}}_j \mathbf{w}, \quad \textit{for all} \quad 0 \leq j \leq J, \quad \textit{and} \quad \overline{\widetilde{\mathbf{\Phi}}}_J \overline{\mathbf{w}} = \mathbf{R} \cdot \widetilde{\mathbf{\Phi}}_j \mathbf{w}.$$

Theorem 3.3 (Scattering Equivariance)**.** *Assume that σ commutes with rotations, i.e., that $\sigma(\mathbf{R} \cdot \mathbf{w}) = \mathbf{R} \cdot \sigma(\mathbf{w})$ for all rotation matrices $\mathbf{R}$. Then, for all $m \geq 1$, we have*

$$\overline{\widetilde{\mathcal{U}}}[j_1, \ldots, j_m]\overline{\mathbf{w}} = \overline{\widetilde{\mathcal{U}}}[j_1, j_2, \ldots, j_m](\mathbf{R} \cdot \mathbf{w}) = \mathbf{R} \cdot \widetilde{\mathcal{U}}[j_1, \ldots, j_m]\mathbf{w}.$$

We note that Theorem 3.3 introduces the assumption that σ commutes with rotations. To understand why this condition is necessary, consider the case where $m = 1$ and recall the definition, $\widetilde{\mathcal{U}}[j_1]\mathbf{w}(v) = \sigma(\widetilde{\mathbf{\Psi}}_{j_1}\mathbf{w}(v))$. Applying Theorem 3.2 allows us to see that

$$\overline{\widetilde{\mathcal{U}}}[j_1]\mathbf{w}(v) = \sigma(\overline{\widetilde{\mathbf{\Psi}}}_{j_1}\mathbf{w}(v)) = \sigma(\mathbf{R} \cdot \widetilde{\mathbf{\Psi}}_{j_1}\mathbf{w}(v)).$$

Therefore, the equivariance result will not hold without this assumption. We also note that we may readily construct activations σ which satisfy this assumption by defining $\sigma(\mathbf{w})(v) = \sigma_{\text{radial}}(\|\mathbf{w}(v)\|_2)\frac{\mathbf{w}(v)}{\|\mathbf{w}(v)\|_2}$ when $\mathbf{w}(v) \neq 0$, and $\sigma(\mathbf{w})(v) = 0$ when $\mathbf{w}(v) = 0$, where σ_{radial} is any real-valued function defined on $[0, \infty)$.

4 Experimental Results

We conduct experiments with an Equivariant Scattering-based GNN (ESc-GNN), illustrated in Figure 3, based on the vector-valued diffusion wavelets and scattering transform introduced in Section 3.

[4]Proofs of all theorems are provided in Appendix A.

We consider node-level and graph-level tasks on synthetic $3D$ point clouds randomly sampled from the surface of randomly generated ellipsoids. As baselines, we used ESc-GNN (non-equivariant), an ablated version of our model which treats each vector-valued signal $\mathbf{w} : V \to \mathbb{R}^3$ as three separate scalar-valued signals; a non-equivariant scattering based GNN, LEGS [Tong et al., 2020]; two equivariant GNNs, EGNN [Satorras et al., 2021] and TFN [Thomas et al., 2018]; and several standard message passing networks, GCN [Kipf and Welling, 2017], GAT [Veličković et al., 2018], and GIN [Xu et al., 2019]. (For a detailed description of our architecture and experimental setup, as well as a discussion of computational complexity, please see Appendices D and E.) For node features, we use the $3D$ coordinates of the points. Collectively, our experiments demonstrate the importance of rotational equivariance, with our ESc-GNN achieving comparable performance to other equivariant GNNs with significantly fewer trainable parameters.[5]

We first consider a graph-level task, on a data set of $N_G = 512$ graphs, where the goal is to predict the Euclidean diameter of each graph, defined by $\mathrm{diam}(G_j) = \max_{v,v' \in V(G_j)} \|v - v'\|_2$. To construct these graphs, we define random ellipsoids $\mathcal{E}_j = \{(x, y, z) \in \mathbb{R}^3 : x^2/a_j^2 + y^2/b_j^2 + z^2/c_j^2 = 1\}$, $1 \leq j \leq N_G$, where a_j, b_j, and c_j are i.i.d. randomly generated coefficients with each $a_j \sim \mathcal{N}(3, 0.5)$ and $b_j, c_j \sim \mathcal{N}(1, 0.2)$, and, for convenience, we use x, y and z to denote the coordinates of a point in $3D$ space, i.e., $v = (x, y, z)$. We note that by construction, we will usually have $a_j^2 > b_j^2, c_j^2$ so these ellipsoids will typically be more "stretched out" along the x-axis than along the y and z axes.

From each ellipsoid $\mathcal{E}_j$, we then sample $n = 128$ points, $\{v_i^{(j)}\}_{i=1}^n$, and construct a k-NN graph with $k = 5$. To demonstrate the utility of equivariant models, we rotate each ellipsoid in the test set (and only in the test set) by 90 degrees so that they are most stretched out along the y axis. Our results are shown in Table 1. We observe that while the non-equivariant models are able to perform well on the (non-rotated) validation set, they fail spectacularly on the test set. For example, the non-equivariant version of ESc-GNN has an average validation mean square error (MSE) of 0.3126, but average test MSE of 2.9181. By contrast, the equivariant version of ESc-GNN, as well as the other equivariant networks, do not suffer from this limitation and achieve similar performance on validation and test sets. Additionally, we note that ESc-GNN outperforms the equivariant baselines with only $59,779$ parameters compared to the $284,036$ parameters in EGNN and the $29,672,961$ parameters in TFN.

We next consider a node-level task where the goal is to learn the value of a function $\mathbf{h} : \mathcal{E} \to \mathbb{R}^3$ defined on the underlying ellipsoid. At each vertex v_i, the direction of $\mathbf{h}(v_i)$ is chosen to be the outward normal vector of $\mathcal{E}$. The magnitude is constructed via a randomly generated bandlimited function, generated using the first $K = 16$ eigenfunctions of the Laplace-Beltrami operator. (See Appendix E.3 for details.) Overall, our results, shown in Table 2, tell a similar story to the graph-level results in Table 1, with the non-equivariant methods perform significantly worse on the test set than on the (non-rotated) validation set (although the drop in performance is not quite as extreme as before). The performance of ESc-GNNis comparable to the other equivariant GNNs, slightly better than EGNN and slightly worse than TFN. However, ESc-GNN uses only $57,250$ parameters in comparison to $834,824$ for EGNN and $29,664,384$ for TFN. Therefore, these results indicate that ESc-GNN is a fast, lightweight alternative to standard equivariant GNNs.

Table 1: Results on the diameter prediction task (mean $\pm$ std. across 5-fold CV). Validation MSE is the (average) of the best score achieved during training for each fold. Best is **bolded**; second-best is underlined. ESc-GNN outperforms other equivariant methods with a fraction of the parameter count.

Model	**Validation MSE** $\downarrow$	**(Rotated) test MSE** $\downarrow$	**Parameter count**
ESc-GNN (Ours)	0.0035 ± 0.0011	$\mathbf{0.0037 \pm 0.0022}$	$59,779$
ESc-GNN (non-equivariant) (Ours)	0.3126 ± 0.5386	2.9181 ± 5.6549	$14,481$
LEGS	1.1812 ± 2.5469	2.2759 ± 3.0514	$19,525$
GCN	0.1684 ± 0.3247	1.4566 ± 2.8307	$60,801$
GAT	0.3964 ± 0.6675	1.4443 ± 2.3166	$61,313$
GIN	0.4778 ± 0.6908	1.6425 ± 2.5529	$93,825$
EGNN (2-layer)	0.0326 ± 0.0310	$\underline{0.0284 \pm 0.0249}$	$284,036$
TFN (4-layer)	0.0787 ± 0.0070	0.0828 ± 0.0122	$29,672,961$

[5]Our code is available at `https://github.com/dj408/esc-gnn`

Table 2: Node-level results (mean $\pm$ std. across five-fold CV). Validation MSE is the (average) of the best score achieved during training for each fold. Best test MSE score is **bolded**; second-best is underlined. ESc-GNN is the second best method to TFN, but with less than 0.2% of the parameters.

Model	Validation MSE $\downarrow$	(Rotated) test MSE $\downarrow$	Parameter count
ESc-GNN (Ours)	0.1525 ± 0.0069	0.1513 ± 0.0080	$57,250$
ESc-GNN (non-equivariant) (Ours)	0.2650 ± 0.2654	0.3552 ± 0.2867	$14,611$
LEGS	0.2224 ± 0.2167	0.3575 ± 0.3239	$15,335$
GCN	0.3528 ± 0.3075	0.4820 ± 0.3669	$44,451$
GAT	0.3594 ± 0.3257	0.4792 ± 0.3639	$44,963$
GIN	0.3175 ± 0.2839	0.4536 ± 0.3512	$77,475$
EGNN (7-layer)	0.1949 ± 0.0297	0.1960 ± 0.0311	$834,824$
TFN (4-layer)	0.1137 ± 0.0032	$\mathbf{0.1145 \pm 0.0039}$	$29,664,384$

5 Conclusion

We have introduced a novel version of the geometric scattering transform for vector-valued signals. We have proved theoretical results demonstrating the rotational equivariance of our method and conducted experiments showing that it may be effectively incorporated into a GNN, which achieves comparable performance to baselines with a fraction of the parameter count.

References

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 929–947, 2024.

Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):2453, 2022.

Dhananjay Bhaskar, Yanlei Zhang, Charles Xu, Xingzhi Sun, Oluwadamilola Fasina, Guy Wolf, Maximilian Nickel, Michael Perlmutter, and Smita Krishnaswamy. Learning graph geometry and topology using dynamical systems based message-passing. *arXiv preprint arXiv:2309.09924*, 2023.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Joyce Chew, Holly Steach, Siddharth Viswanath, Hau-Tieng Wu, Matthew Hirn, Deanna Needell, Matthew D Vesely, Smita Krishnaswamy, and Michael Perlmutter. The manifold scattering transform for high-dimensional point cloud data. In *Topological, Algebraic and Geometric Learning Workshops 2022*, pages 67–78. PMLR, 2022.

Joyce Chew, Matthew Hirn, Smita Krishnaswamy, Deanna Needell, Michael Perlmutter, Holly Steach, Siddharth Viswanath, and Hau-Tieng Wu. Geometric scattering on measure spaces. *Applied and Computational Harmonic Analysis*, 70:101635, 2024. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2024.101635. URL https://www.sciencedirect.com/science/article/pii/S1063520324000125.

Joyce A Chew, Deanna Needell, and Michael Perlmutter. A convergence rate for manifold neural networks. In *2023 International Conference on Sampling Theory and Applications (SampTA)*, pages 1–5. IEEE, 2023.

Alexandre Duval, Simon V Mathis, Chaitanya K Joshi, Victor Schmidt, Santiago Miret, Fragkiskos D Malliaros, Taco Cohen, Pietro Lio, Yoshua Bengio, and Michael Bronstein. A hitchhiker's guide to geometric gnns for 3d atomic systems. *arXiv preprint arXiv:2312.07511*, 2023.

Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural networks*, 107:3–11, 2018.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

Fernando Gama, Alejandro Ribeiro, and Joan Bruna. Diffusion scattering transforms on graphs. In *International Conference on Learning Representations*, 2018.

Feng Gao, Guy Wolf, and Matthew Hirn. Geometric scattering for graph data analysis. In *Proceedings of the 36th International Conference on Machine Learning, PMLR*, volume 97, pages 2122–2131, 2019.

Jiaqi Han, Yu Rong, Tingyang Xu, and Wenbing Huang. Geometrically equivariant graph neural networks: A survey. *arXiv preprint arXiv:2202.07230*, 2022.

David R Johnson, Joyce A Chew, Siddharth Viswanath, Edward De Brouwer, Deanna Needell, Smita Krishnaswamy, and Michael Perlmutter. Manifold filter-combine networks. *Sampling Theory, Signal Processing, and Data Analysis*, 23(2):17, 2025a.

David R Johnson, Smita Krishnaswamy, and Michael Perlmutter. Infogain wavelets: Furthering the design of diffusion wavelets for graph-structured data. *arXiv preprint arXiv:2504.08802*, 2025b.

Chaitanya K. Joshi, Cristian Bodnar, Simon V. Mathis, Taco Cohen, and Pietro Liò. On the expressive power of geometric graph neural networks. In *International Conference on Machine Learning*, 2023.

Bharti Khemani, Shruti Patil, Ketan Kotecha, and Sudeep Tanwar. A review of graph neural networks: concepts, architectures, techniques, challenges, datasets, applications, and future directions. *Journal of Big Data*, 11(1):18, 2024.

T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *Proc. of ICLR*, 2017.

Xuan Li, Zhanke Zhou, Jiangchao Yao, Yu Rong, Lu Zhang, and Bo Han. Neural atoms: Propagating long-range interaction in molecular graphs through efficient communication channel. In *ICLR*, 2024.

Weigang Lu, Ziyu Guan, Wei Zhao, Yaming Yang, and Long Jin. Nodemixup: Tackling under-reaching for graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38 No.13, pages 14175–14183, 2024.

Yimeng Min, Frederik Wenkel, and Guy Wolf. Scattering GCN: Overcoming oversmoothness in graph convolutional networks. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Yimeng Min, Frederik Wenkel, and Guy Wolf. Geometric scattering attention networks. In *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.

Hoang Nt and Takanori Maehara. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

A. Ortega, P. Frossard, J. Kovačević, J. M. F. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

Michael Perlmutter, Alexander Tong, Feng Gao, Guy Wolf, and Matthew Hirn. Understanding graph neural networks with generalized geometric scattering transforms. *SIAM Journal on Mathematics of Data Science*, 5(4):873–898, 2023. doi: 10.1137/21M1465056. URL https://doi.org/10.1137/21M1465056.

Shaima Qureshi et al. Limits of depth: Over-smoothing and over-squashing in gnns. *Big Data Mining and Analytics*, 7(1):205–216, 2023.

Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013.

Amit Singer and Hau-Tieng Wu. Vector diffusion maps and the connection laplacian. *Communications on pure and applied mathematics*, 65(8):1067–1144, 2012.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. arXiv:1802.08219, 2018.

Alexander Tong, Frederik Wenkel, Dhananjay Bhaskar, Kincaid Macdonald, Jackson Grady, Michael Perlmutter, Smita Krishnaswamy, and Guy Wolf. Learnable filters for geometric scattering modules. *IEEE Transactions on Signal Processing*, 72:2939–2952, 2024.

Zekun Tong, Yuxuan Liang, Changsheng Sun, Xinke Li, David Rosenblum, and Andrew Lim. Digraph inception convolutional networks. *Advances in neural information processing systems*, 33:17907–17918, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Zhiyang Wang, Luana Ruiz, and Alejandro Ribeiro. Convolutional filtering on sampled manifolds. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

Zhiyang Wang, Luana Ruiz, and Alejandro Ribeiro. Geometric graph filters and neural networks: Limit properties and discriminability trade-offs. *IEEE Transactions on Signal Processing*, 2024.

Frederik Wenkel, Yimeng Min, Matthew Hirn, Michael Perlmutter, and Guy Wolf. Overcoming oversmoothness in graph convolutional networks via hybrid scattering networks. *arXiv preprint arXiv:2201.08932*, 2022.

Frederik Wenkel, Semih Cantürk, Stefan Horoi, Michael Perlmutter, and Guy Wolf. Towards a general recipe for combinatorial optimization with multi-filter gnns. In *The Third Learning on Graphs Conference*, 2024.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Dongmian Zou and Gilad Lerman. Graph convolutional neural networks via scattering. *Applied and Computational Harmonic Analysis*, 49(3)(3):1046–1074, 2019.

A Proofs of Main Theorems

A.1 The Proof of Theorem 3.1

Proof. Let $\mathbf{Q}'$ be the $d \times d$ matrix defined in the same manner as $\mathbf{Q}$, but with the $d \times d$ identity matrix in place of $\mathcal{O}_{i,j}$, i.e.,

$$\mathbf{Q}'[i,j] = \mathbf{P}[i,j]\mathbf{I}.$$

Let $\widetilde{\mathcal{W}}'_J = \{\widetilde{\mathbf{\Psi}}'_j\}_{j=0}^J \cup \{\widetilde{\mathbf{\Phi}}'_J\}$ be analogous to $\widetilde{\mathcal{W}}_J$, but with $\mathbf{Q}'$ in place of $\mathbf{Q}$, i.e., $\widetilde{\mathbf{\Psi}}'_0 = \mathbf{I} - \mathbf{Q}'$,

$$\widetilde{\mathbf{\Psi}}'_j = (\mathbf{Q}')^{2^{j-1}} - (\mathbf{Q}')^{2^j}, \quad 1 \leq j \leq J,$$

and $\widetilde{\mathbf{\Phi}}'_J = (\mathbf{Q}')^{2^J}$.

The following lemma shows that a result similar to Theorem 3.1 holds for $\widetilde{\mathcal{W}}'_J$.

Lemma A.1. *For all* $\mathbf{w} \in \mathbb{R}^{nd}$*, we have*

$$c\,\frac{d_{\min}}{d_{\max}}\|\mathbf{w}\|_2^2 \leq \left\|\widetilde{\mathcal{W}}'_J\mathbf{w}\right\|_2^2 := \sum_{j=0}^{J}\left\|\widetilde{\mathbf{\Psi}}'_j\mathbf{w}\right\|_2^2 + \left\|\widetilde{\mathbf{\Phi}}'_J\mathbf{w}\right\|_2^2 \leq \frac{d_{\max}}{d_{\min}}\|\mathbf{w}\|_2^2,$$

where c *a universal constant.*

We will also need the following lemma which provides a simplified expression for $\mathbf{Q}^m$

Lemma A.2. *For all* $m \geq 0$*, we may write* $\mathbf{Q}^m$ *in block form as*

$$\mathbf{Q}^m[i,j] = \mathbf{P}^m[i,j]\mathcal{O}_{i,j},$$

where $\mathbf{P}^m[i,j] \in \mathbb{R}$ *is the* i,j*-th entry of* $\mathbf{P}^m$*.*

For proofs of Lemma A.1 and A.2, please see Appendices B.1 and B.2.

Now, let $\mathbf{w} = [\mathbf{w}[1]^\top, \ldots, \mathbf{w}[n]^\top]^\top \in \mathbb{R}^{nd}$ be a vector-valued signal (so that $\mathbf{w}[k] = \mathbf{w}(v_k) \in \mathbb{R}^d$). Define $\mathbf{y} \in \mathbb{R}^{nd}$ by $\mathbf{y} = [\mathbf{y}[1]^\top, \ldots, \mathbf{y}[n]^\top]$, where

$$\mathbf{y}[k] = \mathbf{U}_k\mathbf{w}[k].$$

Note that since $\mathbf{U}_k$ is unitary, we have $\|\mathbf{y}[k]\|_2 = \|\mathbf{w}[k]\|_2$ for all k, which further implies that $\|\mathbf{y}\|_2 = \|\mathbf{w}\|_2$. Next observe that by Lemma A.2 we have, for $1 \leq j \leq J$,

$$\widetilde{\mathbf{\Psi}}_j[i,k] = \mathbf{Q}^{2^{j-1}}[i,k] - \mathbf{Q}^{2^j}[i,k] = \mathbf{P}^{2^{j-1}}[i,k]\mathcal{O}_{i,k} - \mathbf{P}^{2^j}[i,k]\mathcal{O}_{i,k} = \widetilde{\mathbf{\Psi}}'_j[i,k]\mathcal{O}_{i,k},$$

where in the final equality we use the fact that $\widetilde{\mathbf{\Psi}}'_j[i,k] = \left(\mathbf{P}^{2^{j-1}}[i,k] - \mathbf{P}^{2^j}[i,k]\right)\mathbf{I}$. Similarly, we have $\widetilde{\mathbf{\Psi}}_0[i,k] = \widetilde{\mathbf{\Psi}}'_0[i,k]\mathcal{O}_{i,k}$ and $\widetilde{\mathbf{\Phi}}_J[i,k] = \widetilde{\mathbf{\Phi}}'_J[i,k]\mathcal{O}_{i,k}$. Therefore, for all $0 \leq j \leq J$, we have

$$\begin{aligned}
(\widetilde{\mathbf{\Psi}}_j\mathbf{y})[i] &= \sum_{k=1}^{n}\widetilde{\mathbf{\Psi}}_j[i,k]\mathbf{y}[k] \\
&= \sum_{k=1}^{n}\widetilde{\mathbf{\Psi}}'_j[i,k]\mathcal{O}_{i,k}\mathbf{U}_k\mathbf{w}[k] \\
&= \sum_{k=1}^{n}\widetilde{\mathbf{\Psi}}'_j[i,k]\mathbf{U}_i\mathbf{U}_k^\top\mathbf{U}_k\mathbf{w}[k] \\
&= \mathbf{U}_i\sum_{k=1}^{n}\widetilde{\mathbf{\Psi}}'_j[i,k]\mathbf{w}[k] \\
&= \mathbf{U}_i((\mathbf{\Psi}'_j\mathbf{w})[i]),
\end{aligned}$$

and likewise $(\widetilde{\mathbf{\Phi}}_J\mathbf{y})[i] = \mathbf{U}_i((\widetilde{\mathbf{\Phi}}'_J\mathbf{w})[i])$. Since $\mathbf{U}_i$ is unitary, this implies that

$$\sum_{j=0}^{J}\left\|\widetilde{\mathbf{\Psi}}'_j\mathbf{y}\right\|_2^2 + \left\|\widetilde{\mathbf{\Phi}}'_J\mathbf{y}\right\|_2^2 = \sum_{j=0}^{J}\left\|\widetilde{\mathbf{\Psi}}_j\mathbf{w}\right\|_2^2 + \left\|\widetilde{\mathbf{\Phi}}_J\mathbf{w}\right\|_2^2. \tag{5}$$

However, by Lemma A.1, we have

$$\begin{aligned}
c\,\frac{d_{\min}}{d_{\max}}\|\mathbf{w}\|_2^2 &= c\,\frac{d_{\min}}{d_{\max}}\|\mathbf{y}\|_2^2 \\
&\leq \sum_{j=0}^{J}\left\|\widetilde{\mathbf{\Psi}}'_j\mathbf{y}\right\|_2^2 + \left\|\widetilde{\mathbf{\Phi}}'_J\mathbf{y}\right\|_2^2 \\
&\leq \frac{d_{\max}}{d_{\min}}\|\mathbf{y}\|_2^2 \\
&= \frac{d_{\max}}{d_{\min}}\|\mathbf{w}\|_2^2.
\end{aligned}$$

Combining this with (5) completes the proof.

□

A.2 The Proof of Theorem 3.2

To prove Theorem 3.2, we need the following lemma which establishes the equivariance of the powered vector diffusion matrix $\mathbf{Q}^m$, which is also illustrated by Figure 1. For a proof, please see Appendix B.3.

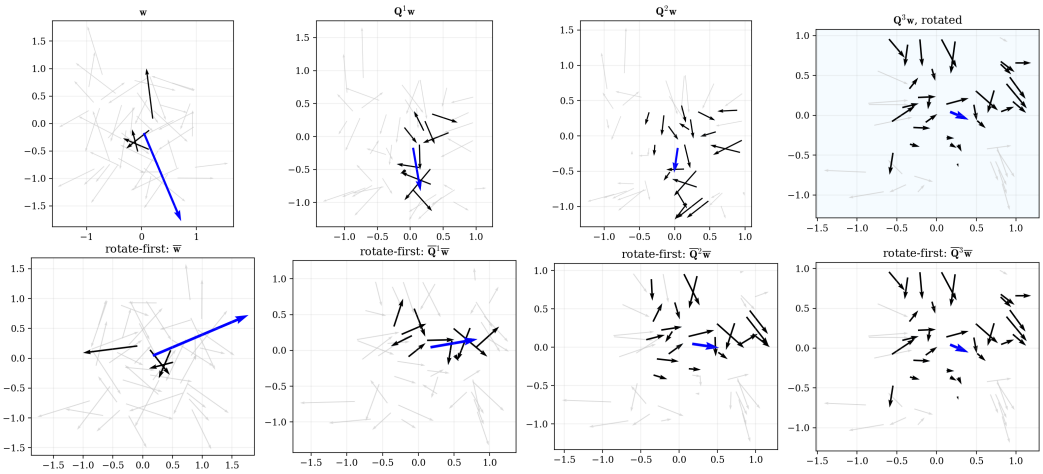

Figure 1: Illustration of rotational equivariance of $\mathbf{Q}^m$, applied to a $2D$ vector field. (The field was generated with random uniform sampling of points $x_i \sim \mathcal{U}([-1,1]^2)$, angles $\theta_i \sim \mathcal{U}[0, 2\pi)$, and magnitudes $m_i \sim \mathcal{U}[0.4, 1.0]$ such that $w_i = m_i[\cos\theta_i,\ \sin\theta_i]$). The central vector (the magnitude of which was made to be the largest, for illustrative purposes) is colored blue. The top row shows $\mathbf{Q}^m$ applied to vectors in an unrotated vector field (where we first used the sample points x_i to construct a k-NN graph, $k = 3$). The bottom row shows the same system, rotated 90 degrees counter-clockwise, then diffused by $\overline{\mathbf{Q}}$. The black vectors highlight those involved in the next diffusion step. That is, the vectors from which the central vector will receive (indirect) diffusion messages: first, its (symmetric) k-nearest neighbor vectors, then neighbors of neighbors, and so on. After three diffusion steps, the top (unrotated) system is rotated 90 degrees counter-clockwise, like the bottom system was initially. (This panel has a light blue background). Notably, this panel is identical, (other than the background) to the panel immediately below it thereby demonstrating the equivariance of $\mathbf{Q}^m$, since diffusing and then rotating yields the same result as rotating and then diffusing.

Lemma A.3. *For any $m \geq 0$, and any vector-valued node feature $\mathbf{w}$ we have*

$$\overline{\mathbf{Q}}^m \overline{\mathbf{w}} = \overline{\mathbf{Q}}^m (\mathbf{R} \cdot \mathbf{w}) = \mathbf{R} \cdot (\mathbf{Q}^m \mathbf{w}).$$

The proof of Theorem 3.2. We first observe that the claim $\overline{\widetilde{\mathbf{\Phi}}}_J \overline{\mathbf{w}} = \mathbf{R} \cdot \widetilde{\mathbf{\Phi}}_j \mathbf{w}$ follows immediately from Lemma A.3, setting $m = 2^J$.

Now, fix $0 \leq j \leq J$, and observe that we may write $\widetilde{\mathbf{\Psi}}_j = \mathbf{Q}^{t_1} - \mathbf{Q}^{t_2}$ where $t_1 = 0, t_2 = 1$ if $j = 0$ and otherwiswe we have $t_1 = 2^{j-1}, t_2 = 2^j$.

Thus, using Lemma A.3, we observe

$$\begin{aligned}\overline{\widetilde{\mathbf{\Psi}}}_j\overline{\mathbf{w}} &= (\overline{\mathbf{Q}}^{t_1} - \overline{\mathbf{Q}^{t_2}})\overline{\mathbf{w}} \\ &= \overline{\mathbf{Q}}^{t_1}\overline{\mathbf{w}} - \overline{\mathbf{Q}^{t_2}}\overline{\mathbf{w}} \\ &= \mathbf{R}\cdot(\mathbf{Q}^{t_1}\mathbf{w}) - \mathbf{R}\cdot(\mathbf{Q}^{t_2}\mathbf{w}) \\ &= \mathbf{R}\cdot((\mathbf{Q}^{t_1} - \mathbf{Q}^{t_2})\mathbf{w}) \\ &= \mathbf{R}\cdot(\widetilde{\mathbf{\Psi}}_j\mathbf{w}),\end{aligned}$$

where in the final equality we use the fact that $\mathbf{R}\cdot(\mathbf{w}_1 - \mathbf{w}_2) = \mathbf{R}\cdot\mathbf{w}_1 - \mathbf{R}\cdot\mathbf{w}_2$ for any vector-valued node signals $\mathbf{w}_1$ and $\mathbf{w}_2$. □

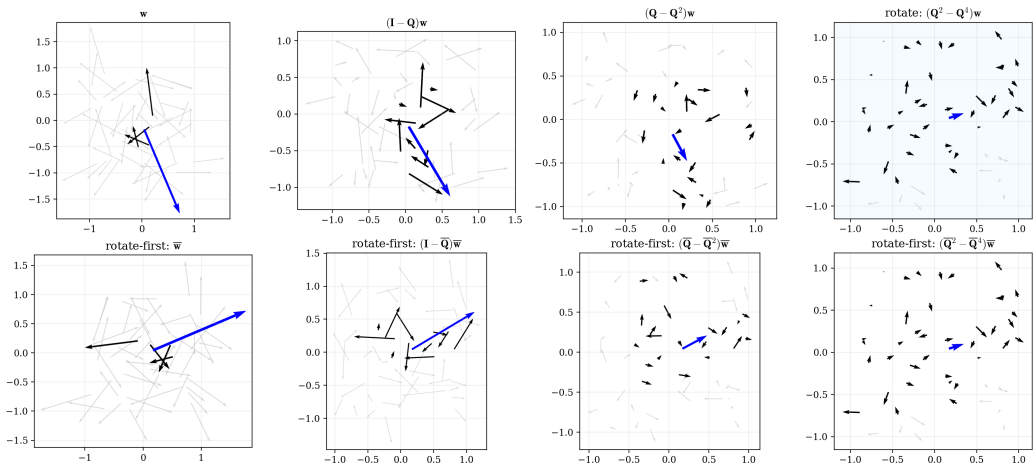

Figure 2: Illustration of rotational equivariance of vector diffusion wavelets applied to a vector field (which is defined the same way as in Figure 1). Here, the top row shows vector diffusion wavelets constructed from $\mathbf{Q}$ applied to vectors in the unrotated vector field; the bottom row shows the same system, rotated 90 degrees counter-clockwise, then diffused using wavelets constructed from $\overline{\mathbf{Q}}$. The black vectors again highlight those from which the central vector will receive (indirect) messages in the next diffusion step. After three wavelet diffusion steps, the top (unrotated) system is rotated 90 degrees counter-clockwise, like the bottom system was initially (this panel has a light blue background). The fact that the diffused vector fields in the rightmost column are equivalent shows the rotational equivariance of the wavelets.

A.3 The Proof of Theorem 3.3

Proof. We argue by induction. For the case case, $m = 1$, we use the assumption that $\mathbf{R}$ commutes with rotations to see,

$$\overline{\widetilde{\mathcal{U}}}[j_1]\overline{\mathbf{w}} = \sigma\left(\overline{\widetilde{\mathbf{\Psi}}}_{j_1}\overline{\mathbf{w}}\right) = \sigma\left(R\cdot\widetilde{\mathbf{\Psi}}_{j_1}\mathbf{w}\right) = R\cdot\sigma\left(\widetilde{\mathbf{\Psi}}_{j_1}\mathbf{w}\right) = R\cdot(\widetilde{\mathcal{U}}[j_1]\mathbf{w}).$$

Now, assume the result holds for some $m \geq 1$. Then by the inductive hypothesis, we have

$$\begin{aligned}\overline{\widetilde{\mathcal{U}}}[j_1,\ldots,j_m,j_{m+1}]\overline{\mathbf{w}} &= \overline{\widetilde{\mathcal{U}}}[j_{m+1}]\overline{\widetilde{\mathcal{U}}}[j_1,\ldots,j_m]\overline{\mathbf{w}} \\ &= \overline{\widetilde{\mathcal{U}}}[j_{m+1}](\mathbf{R}\cdot\widetilde{\mathcal{U}}[j_1,j_2,\ldots,j_m]\mathbf{w}) \\ &= \mathbf{R}\cdot(\overline{\widetilde{\mathcal{U}}}[j_{m+1}]\widetilde{\mathcal{U}}[j_1,j_2,\ldots,j_m]\mathbf{w}) \\ &= \mathbf{R}(\widetilde{\mathcal{U}}[j_1,j_2,\ldots,j_m,j_{m+1}]\mathbf{w}).\end{aligned}$$

□

B Proofs of auxiliary lemmas

B.1 The Proof of Lemma A.1

We first recall the following result from Perlmutter et al. [2023] which shows that the diffusion wavelets $\mathcal{W}_J$ are a non-expansive frame on the weighted inner product space defined by $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle_{\mathbf{D}^{1/2}} = \langle \mathbf{D}^{1/2}\mathbf{x}_1, \mathbf{D}^{1/2}\mathbf{x}_2 \rangle_2$, with corresponding norm $\|\mathbf{x}\|_{\mathbf{D}^{1/2}} = \|\mathbf{D}^{1/2}\mathbf{x}\|_2$. (See also Proposition 4.1 of Gama et al. [2018] and Theorem 1 of Tong et al. [2024].) The key to the proof of Proposition B.2 will be to combine this result with the inequality (6), stated below, that relates this weighted norm to the unweighted ℓ^2 norm.

Proposition B.1 (Proposition 2.2 of Perlmutter et al. [2023])**.** *$\mathcal{W}_J$ is a nonexpansive frame, i.e., there exists a universal constant $c > 0$, which in particular is independent of J, and the geometry of G such that*

$$c\|\mathbf{x}\|^2_{\mathbf{D}^{1/2}} \leq \left\|\mathcal{W}_J\mathbf{x}\right\|^2_{\mathbf{D}^{1/2}} \coloneqq \sum_{j=0}^{J} \left\|\mathbf{\Psi}_j\mathbf{x}\right\|^2_{\mathbf{D}^{1/2}} + \left\|\mathbf{\Phi}_J\mathbf{x}\right\|^2_{\mathbf{D}^{1/2}} \leq \|\mathbf{x}\|^2_{\mathbf{D}^{1/2}} \quad \textit{for all } \mathbf{x} \in \mathbb{R}^n.$$

We may use Proposition B.1 to prove the following corollary which analyzes the frame bounds of the diffusion wavelets with respect to the unweighted ℓ^2 norm.

Corollary B.2. *For all $\mathbf{x} \in \mathbb{R}^n$, we have*

$$c\,\frac{d_{\min}}{d_{\max}}\|\mathbf{x}\|_2^2 \leq \left\|\mathcal{W}_J\mathbf{x}\right\|_2^2 \coloneqq \sum_{j=0}^{J} \left\|\mathbf{\Psi}_j\mathbf{x}\right\|_2^2 + \left\|\mathbf{\Phi}_J\mathbf{x}\right\|_2^2 \leq \frac{d_{\max}}{d_{\min}}\|\mathbf{x}\|_2^2,$$

where $d_{\min}$ and $d_{\max}$ denote the minimal and maximal vertex degrees and c is a universal constant.

Proof. Let $\mathbf{x} \in \mathbb{R}^n$. It is straightforward to see that

$$d_{\min}\|\mathbf{x}\|_2^2 \leq \|\mathbf{x}\|^2_{\mathbf{D}^{1/2}} \leq d_{\max}\|\mathbf{x}\|_2^2. \tag{6}$$

Therefore, by Proposition B.1, we have

$$\begin{aligned}
\sum_{j=0}^{J} \left\|\mathbf{\Psi}_j\mathbf{x}\right\|_2^2 + \left\|\mathbf{\Phi}_J\mathbf{x}\right\|_2^2 &\leq d_{\max}\left(\sum_{j=0}^{J} \left\|\mathbf{\Psi}_j\mathbf{x}\right\|^2_{\mathbf{D}^{1/2}} + \left\|\mathbf{\Phi}_J\mathbf{x}\right\|^2_{\mathbf{D}^{1/2}}\right) \\
&\leq d_{\max}\|\mathbf{x}\|^2_{\mathbf{D}^{1/2}} \\
&\leq \frac{d_{\max}}{d_{\min}}\|\mathbf{x}\|_2^2,
\end{aligned}$$

which establishes the upper frame bound (i.e., the rightmost inequality). Similarly, to establish the lower frame bound, we have

$$\begin{aligned}
\sum_{j=0}^{J} \left\|\mathbf{\Psi}_j\mathbf{x}\right\|_2^2 + \left\|\mathbf{\Phi}_J\mathbf{x}\right\|_2^2 &\geq d_{\min}\left(\sum_{j=0}^{J} \left\|\mathbf{\Psi}_j\mathbf{x}\right\|^2_{\mathbf{D}^{1/2}} + \left\|\mathbf{\Phi}_J\mathbf{x}\right\|^2_{\mathbf{D}^{1/2}}\right) \\
&\geq c d_{\min}\|\mathbf{x}\|^2_{\mathbf{D}^{1/2}} \\
&\geq c\,\frac{d_{\min}}{d_{\max}}\|\mathbf{x}\|_2^2.
\end{aligned}$$

□

The proof of Lemma A.1. Let $\mathbf{w} : V \to \mathbb{R}^d$, written in vector form as

$$\begin{aligned}
\mathbf{w} &= [\mathbf{w}[1]^\top, \ldots, \mathbf{w}[n]^T]^\top \\
&= [\mathbf{w}[1][1], \ldots, \mathbf{w}[1][d], \mathbf{w}[2][1], \ldots, \mathbf{w}[2][d], \ldots, \mathbf{w}[n][1], \ldots, \mathbf{w}[n][d]]^\top.
\end{aligned}$$

We first observe that we may write $\mathbf{Q}' = \mathbf{P} \otimes \mathbf{I}$, where $\mathbf{I}$ is the $d \times d$ identity matrix and $\otimes$ denotes the Kronecker product. By the mixed-product property of the Kronecker product, this implies that for all $m \geq 0$, we have

$$(\mathbf{Q}')^m = \mathbf{P}^m \otimes \mathbf{I}.$$

Therefore, by linearity, we have

$$\widetilde{\mathbf{\Psi}}_j' = \mathbf{\Psi}_j \otimes \mathbf{I},$$

and similarly $\widetilde{\mathbf{\Phi}}_J' = \mathbf{\Phi}_J \otimes \mathbf{I}$.

Now, let $\mathbf{\Pi} \in \mathbb{R}^{nd \times nd}$ be the permutation matrix such that applying $\mathbf{\Pi}$ to $\mathbf{w}$ yields

$$\begin{aligned}\mathbf{\Pi}\mathbf{w} &= [\mathbf{w}[1][1], \ldots, \mathbf{w}[n][1], \mathbf{w}[1][2], \ldots, \mathbf{w}[n][2], \ldots, \mathbf{w}[1][d], \ldots, \mathbf{w}[n][d]]^\top \\ &= [\mathbf{w}[:][1]^\top, \mathbf{w}[:][1]^\top, \ldots, \mathbf{w}[:][d]^\top]^\top,\end{aligned}$$

where $\mathbf{w}[:][k]^\top = [\mathbf{w}[1][k], \ldots, x[n][k]]^\top$ for all $1 \leq k \leq d$. (Formally, let $\mathbf{\Pi}$ be the matrix corresponding to the permutation $\widetilde{\sigma} : \{1, 2, 3, \ldots, nd\} \to \{1, 2, 3, \ldots, nd\}$, $\widetilde{\sigma}(i) = n \cdot ((i-1) \bmod d) + \lceil \frac{i}{d} \rceil$.) To understand this permutation, note that we have reorder the entries of the vector $\mathbf{w}$ so that all entries corresponding to the first output dimension of the function $\mathbf{w} : V \to \mathbb{R}^d$ come first, then all of the entries corresponding to the second output dimension come next, etc.

Since $\mathbf{Q}' = \mathbf{P} \otimes \mathbf{I}$, applying $\mathbf{\Pi}$ to both the columns and the rows of $\mathbf{Q}'$ yields

$$\mathbf{\Pi}\mathbf{Q}'\mathbf{\Pi}^\top = \mathbf{I} \otimes \mathbf{P} = \begin{pmatrix} \mathbf{P} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P} \end{pmatrix}.$$

Moreover, since $\mathbf{\Pi}^\top\mathbf{\Pi} = \mathbf{I}$, we see that

$$(\mathbf{\Pi}\mathbf{Q}'\mathbf{\Pi}^\top)^m = \mathbf{\Pi}(\mathbf{Q}')^m\mathbf{\Pi}^\top = \begin{pmatrix} \mathbf{P}^m & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{P}^m & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{P}^m \end{pmatrix} = \mathbf{I} \otimes \mathbf{P}^m.$$

for all $m \geq 0$, and so

$$\mathbf{\Pi}\widetilde{\mathbf{\Psi}}_j'\mathbf{\Pi} = \mathbf{I} \otimes \mathbf{\Psi}_j = \begin{pmatrix} \mathbf{\Psi}_j & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{\Psi}_j & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{\Psi}_j \end{pmatrix},$$

and similarly, $\mathbf{\Pi}\widetilde{\mathbf{\Phi}}_J'\mathbf{\Pi} = \mathbf{I} \otimes \mathbf{\Phi}_J$. Therefore, we have

$$\mathbf{\Pi}\widetilde{\mathbf{\Psi}}_j'\mathbf{w} = (\mathbf{\Pi}\widetilde{\mathbf{\Psi}}_j'\mathbf{\Pi}^\top)(\mathbf{\Pi}\mathbf{w}) = [\mathbf{\Psi}_j\mathbf{w}[:][1], \mathbf{\Psi}_j\mathbf{w}[:][2], \ldots, \mathbf{\Psi}_j\mathbf{w}][:\,[d]]^\top,$$

with an analogous equation for $\widetilde{\mathbf{\Phi}}_J'$. Thus,

$$\begin{aligned}\sum_{j=0}^{J} \|\widetilde{\mathbf{\Psi}}_j'\mathbf{w}\|^2 + \|\widetilde{\mathbf{\Phi}}_J'\mathbf{w}\|^2 &= \sum_{j=0}^{J} \|\mathbf{\Pi}\widetilde{\mathbf{\Psi}}_j'\mathbf{w}\|_2^2 + \|\widetilde{\mathbf{\Phi}}_J'\mathbf{w}\|_2^2 \\ &= \sum_{j=0}^{J}\sum_{k=1}^{d} \|\mathbf{\Psi}_j\mathbf{w}[:][k]\|_2^2 + \sum_{k=1}^{d} \|\mathbf{\Phi}_J\mathbf{w}[:][k]\|_2^2 \\ &= \sum_{k=1}^{d} \left(\sum_{j=0}^{J} \|\mathbf{\Psi}_j\mathbf{w}[:][k]\|_2^2 + \|\mathbf{\Phi}_J\mathbf{w}[:][k]\|_2^2 \right). \qquad (7)\end{aligned}$$

By Corollary B.2, we have

$$c\frac{d_{\min}}{d_{\max}}\|\mathbf{w}[:][k]\|_2^2 \leq \sum_{j=0}^{J}\|\mathbf{\Psi}_j\mathbf{w}[:][k]\|_2^2 + \|\mathbf{\Phi}_J\mathbf{w}[:][k]\|_2^2 \leq \frac{d_{\max}}{d_{\min}}\|\mathbf{w}[:][k]\|_2^2,$$

and so, using (7) together with the fact that $\|\mathbf{w}\|_2^2 = \sum_{k=1}^{d}\|\mathbf{w}[:][k]\|_2^2$, we have

$$\begin{aligned}
c\frac{d_{\min}}{d_{\max}}\|\mathbf{w}\|_2^2 &= c\frac{d_{\min}}{d_{\max}}\sum_{k=1}^{d}\|\mathbf{w}[:][k]\|_2^2 \\
&\leq \sum_{k=1}^{d}\left(\sum_{j=0}^{J}\|\mathbf{\Psi}_j\mathbf{w}[:][k]\|_2^2 + \|\mathbf{\Phi}_J\mathbf{w}[:][k]\|_2^2\right) \\
&= \sum_{j=0}^{J}\|\widetilde{\mathbf{\Psi}}_j'\mathbf{w}\|^2 + \|\widetilde{\mathbf{\Phi}}_J'\mathbf{w}\|^2 \\
&= \sum_{k=1}^{d}\left(\sum_{j=0}^{J}\|\mathbf{\Psi}_j\mathbf{w}[:][k]\|_2^2 + \|\mathbf{\Phi}_J\mathbf{w}[:][k]\|_2^2\right) \\
&\leq \sum_{k=1}^{d}\frac{d_{\max}}{d_{\min}}\|\mathbf{w}[:][k]\|_2^2 \\
&= \frac{d_{\max}}{d_{\min}}\|\mathbf{w}\|_2^2.
\end{aligned}$$

□

B.2 The Proof of Lemma A.2

Proof. When $m = 0$, this follows from the fact that for all i, $\mathcal{O}_{i,i} = \mathbf{U}_i\mathbf{U}_i^\top = \mathbf{I}$ since $\mathbf{U}_i$ is unitary, and so $\mathbf{Q}^0 = \mathbf{I} = \mathbf{I}\,\mathbf{I} = \mathbf{P}^0\mathcal{O}_{i,i}$. The case $m = 1$ follows directly from the definition of $\mathbf{Q}$.

Now, reasoning by induction, assume the result holds for m. Then, using block matrix multiplication, we have

$$\begin{aligned}
\mathbf{Q}^{m+1}[i,j] &= \sum_{k=1}^{n}\mathbf{Q}^m[i,k]\mathbf{Q}[k,j] \\
&= \sum_{k=1}^{n}\mathbf{P}^m[i,k]\mathbf{P}[k,j]\mathcal{O}_{i,k}\mathcal{O}_{k,j} \\
&= \sum_{k=1}^{n}\mathbf{P}^m[i,k]\mathbf{P}[k,j]\mathbf{U}_i\mathbf{U}_k^\top\mathbf{U}_k\mathbf{U}_j^\top \\
&= \sum_{k=1}^{n}\mathbf{P}^m[i,k]\mathbf{P}[k,j]\mathbf{U}_i\mathbf{U}_j^\top \\
&= \sum_{k=1}^{n}\mathbf{P}^m[i,k]\mathbf{P}[k,j]\mathcal{O}_{i,j} \\
&= \mathbf{P}^{m+1}[i,j]\mathcal{O}_{i,j}.
\end{aligned}$$

□

B.3 The proof of Lemma A.3

We first prove the following auxiliary lemma

Proposition B.3. *$\overline{\mathcal{O}}_{i,j}$, the change of coordinate matrix in the rotated coordinate system, satisfies*

$$\overline{\mathcal{O}}_{i,j} = \mathbf{R}\mathcal{O}_{i,j}\mathbf{R}^\top.$$

Therefore, the corresponding vector-valued diffusion matrix $\overline{\mathbf{Q}}$ can be written in block form by

$$\overline{\mathbf{Q}}[i,j] = \mathbf{P}[i,j]\mathbf{R}\mathcal{O}_{i,j}\mathbf{R}^\top.$$

Proof. Since $\overline{v}_i = \mathbf{R}v_i$, we have that $\overline{v}_i - \overline{v}_j = \mathbf{R}(v_i - v_j)$. This implies $\overline{\mathbf{C}}_i = \mathbf{R}\mathbf{C}_i$ and $\overline{\mathbf{D}_i} = \mathbf{D}_i$ (since rotation preserves distances), which implies

$$\overline{\mathbf{B}}_i = \mathbf{R}\mathbf{B}_i.$$

Therefore, $\overline{\mathbf{B}}_i = \mathbf{R}\mathbf{U}_i\mathbf{\Sigma}_i\mathbf{V}_i^\top$ is a singular value decomposition of $\overline{\mathbf{B}}_i$ and so we have $\overline{\mathbf{U}}_i = \mathbf{R}\mathbf{U}_i$. This implies that

$$\overline{\mathcal{O}}_{i,j} = \overline{\mathbf{U}}_i\overline{\mathbf{U}}_j^\top = (\mathbf{R}\mathbf{U}_i)(\mathbf{R}\mathbf{U}_j)^\top = \mathbf{R}\mathbf{U}_i\mathbf{U}_j^\top\mathbf{R}^\top = \mathbf{R}\mathcal{O}_{i,j}\mathbf{R}^\top,$$

which proves the first claim. The second claim follows immediately recalling that we assume that the edge-connectivity is unchanged by the rotation and so $\overline{\mathbf{P}} = \mathbf{P}$. □

We will now prove Lemma A.3.

Proof. In the case where $m = 0$, the result simply states that $\overline{\mathbf{w}} = \mathbf{R} \cdot \mathbf{w}$, which is true by the assumption that rotating the system rotates the vector-valued node features (vertex by vertex).

For $m \geq 1$, we will use induction. In the base case, $m = 1$, we will use the definition of block-matrix multiplication, as well as Lemma B.3 to write

$$\begin{aligned}
(\overline{\mathbf{Q}}\overline{\mathbf{w}})[i] &= (\overline{\mathbf{Q}}(\mathbf{R} \cdot \mathbf{w}))[i] && (8)\\
&= \sum_{j=1}^{n} \overline{\mathbf{Q}}[i,j](\mathbf{R} \cdot \mathbf{w})[j] && (9)\\
&= \sum_{j=1}^{n} \mathbf{P}[i,j]\mathbf{R}\mathcal{O}_{i,j}\mathbf{R}^\top\mathbf{R}\mathbf{w}[j] && (10)\\
&= \mathbf{R}\sum_{j=1}^{n} \mathbf{P}[i,j]\mathcal{O}_{i,j}\mathbf{w}[j] && (11)\\
&= \mathbf{R}(\mathbf{Q}\mathbf{w}[i]). && (12)
\end{aligned}$$

Since i was arbitrary, this implies $\overline{\mathbf{Q}}\overline{\mathbf{w}} = \mathbf{R} \cdot (\mathbf{Q}\mathbf{w})$ and establishes the base case.

We now assume the result holds for some $m \geq 1$. We let $\mathbf{y} = \mathbf{Q}^m\mathbf{w}$ and let $\overline{\mathbf{y}} = \overline{\mathbf{Q}}^m\overline{\mathbf{w}}$. Observe that $\overline{\mathbf{y}} = \mathbf{R} \cdot (\mathbf{Q}^m\mathbf{w}) = \mathbf{R} \cdot \mathbf{y}$ by the inductive hypothesis. Thus,

$$\begin{aligned}
\overline{\mathbf{Q}}^{m+1}\overline{\mathbf{w}} &= \overline{\mathbf{Q}}\overline{\mathbf{y}}\\
&= \overline{\mathbf{Q}}(\mathbf{R} \cdot \mathbf{y})\\
&= \mathbf{R} \cdot (\mathbf{Q}\mathbf{y}) && (13)\\
&= \mathbf{R} \cdot (\mathbf{Q}\mathbf{Q}^m\mathbf{w})\\
&= \mathbf{R} \cdot \mathbf{Q}^{m+1}\mathbf{w},
\end{aligned}$$

where in (13) we use the inductive hypothesis with $\mathbf{y}$ in place of $\mathbf{w}$.

□

C Removing the sign ambiguity

The definition of diffusion wavelets relies on computing the SVD of the matrices $\mathbf{B}_i$. However, as noted above, the SVD is only unique up to sign-flips. That is, one may obtain a different SVD $\mathbf{B}_i' = \mathbf{U}_i'\mathbf{\Sigma}_i(\mathbf{V}')_i^\top$ by replacing both $\mathbf{u}_{i,k}$ and $\mathbf{v}_{i,k}$ with $\mathbf{u}_{i,k}' = -\mathbf{u}_{i,k}$ and $\mathbf{v}_{i,k}' = -\mathbf{v}_{i,k}$ for any fixed k. Therefore, we will utilize the sign-flipping trick described below in order to ensure that each of the $\mathcal{O}_{i,j}$ are well defined (and do not suffer from any sign ambiguity).

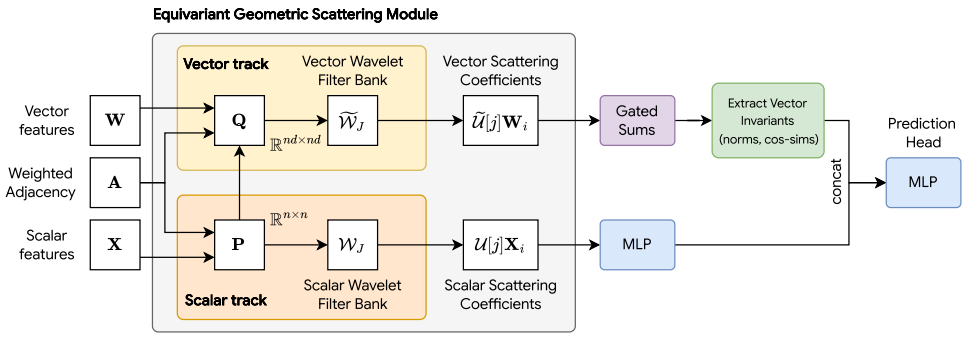

Figure 3: Architecture the equivariant scattering-based GNN. Given a geometric graph, scalar features $\mathbf{X}$ are passed into the scalar track while vector features $\mathbf{W}$ are passed into the vector track. Upon computing the respective scattering coefficients, the updated scalar and vector features are concatenated and projected to form the final node-level embeddings, which can be aggregated to obtain graph-level embeddings.

We first observe that the k, ℓ-th entry of $\mathcal{O}_{i,j}$ is given by

$$\mathcal{O}_{i,j}[k,l] = \sum_{m=1}^{d} \mathbf{U}_i[k,m]\mathbf{U}_j^\top[m,\ell] = \sum_{m=1}^{d} \mathbf{U}_i[k,m]\mathbf{U}_j[\ell,m] = \langle \mathbf{u}_{i,k}, \mathbf{u}_{j,\ell}\rangle.$$

Therefore, if we are able to ensure that $\langle \mathbf{u}_{i,k}, \mathbf{u}_{j,\ell}\rangle$ is always non-negative, then the definition of $\mathcal{O}_{i,j}$ will not suffer from the sign ambiguity.

Accordingly, when computing $\mathcal{O}_{i,j}$, we check whether or not each $\langle \mathbf{u}_{i,k}, \mathbf{u}_{j,\ell}\rangle$ is positive or negative. If $\langle \mathbf{u}_{i,k}, \mathbf{u}_{j,\ell}\rangle$ is negative, we replace $\mathbf{u}_{j,\ell}$ with $-\mathbf{u}_{j,\ell}$. We note that in this construction, the individual vectors, $\mathbf{u}_{i,k}$ still do suffer from the sign-flip ambiguity. However, if one replaces $\mathbf{u}_{i,k}$ with $-\mathbf{u}_{i,k}$, the sign flipping trick will also force us to replace $\mathbf{u}_{j,\ell}$ with $-\mathbf{u}_{j,\ell}$. Thus, the inner products stay the same since $\langle \mathbf{u}_{i,k}, \mathbf{u}_{j,\ell}\rangle = \langle -\mathbf{u}_{i,k}, -\mathbf{u}_{j,\ell}\rangle$.

D GNN Architecture

In this section, we provide details on the equivariant, scattering-based GNN we used for our experiments in Section 4. (Hyperparameter settings are given in Appendix E.) For simplicity, we will assume that we are given a single vector-valued signal, i.e., $F_{\text{vector}} = 1$, which is the case for all of our experiments.

Scalar- and vector-track geometric scattering layers. We first separately compute the first- and second-order scattering coefficients of each input signal, using (2) for the scalar-valued signals and (4) for the vector-valued signal, and concatenate these to the zeroth-order scattering coefficients (where the zeroth-order coefficients are simply the untransformed signals). We organize the scalar-valued coefficients and vector-valued scattering coefficients into tensors $\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{W}}$ with dimensions $n \times F_{\text{scalar}} \times S$ and $n \times d \times S$, where S is the total number of zeroth-, first-, and second-order used for each signal at each node. We note that in our experiments, we use the same value of S for all vector- and scalar-valued features. However, we could readily modify our architecture to use a different number of scales for each feature. We illustrate our architecture in Figure 3.

Within-track mixing layers. Second, for each track, the scattering coefficients are passed through within-track coefficient-mixing layers. For the scalar track, we learn new combinations of the scattering scales using a two-layer MLP, resulting in a new tensor $\widetilde{\mathbf{X}}' \in \mathbb{R}^{n \times F_{\text{scalar}} \times K}$ defined by

$$\widetilde{\mathbf{X}}'[i_1, i_2, :] = \text{MLP}(\widetilde{\mathbf{X}}[i_1, i_2, :]), \tag{14}$$

(where the MLP does not depend on i_1 or i_2).

For the vector track, we perform a weighted summation across the signals with learnable weights and also employ a gating procedure. This yields a new tensor $\widetilde{\mathbf{W}}' \in \mathbb{R}^{n \times d \times K}$ defined by

$$\widetilde{\mathbf{W}}'[i_1, i_2, i_3] = \sigma_w(\alpha_{i_3}) \sum_{s=1}^{S} \theta_{i_3,\ell} \widetilde{\mathbf{W}}[i_1, i_2, s], \tag{15}$$

where each $\alpha_{i_3} \in \mathbb{R}$ is a learnable gating parameter and σ_w is a nonlinear activation.

Vector invariants extraction layer. Third, for the vector track, we extract three invariant scattering vector features, the norm of each vector as well as mean and maximal cosine similarity between each of these vectors and those of its neighboring nodes. We then concatenate these invariant vector features with the scalar track hidden representations into a combined invariant hidden feature tensor $\mathbf{T} \in \mathbb{R}^{n \times (F_{\text{scalar}} + 3) \times K}$. For each node v_i, we then reshape $\mathbf{T}[i, :, :]$ to obtain a flattened hidden feature vector $\mathbf{t}_i \in \mathbb{R}^{K(F_{\text{scalar}} + 3)}$.

Task-specific prediction head. For scalar-valued target functions, we seek for our network to be rationally invariant. Therefore, for node-level tasks, we use a five-layer MLP to map each $\mathbf{t}_i$ to the target dimension. Alternatively, for graph-level tasks with scalar targets, we first pool the vectors $\mathbf{t}_i$ vectors across nodes within graphs (using, e.g., summation over i or max aggregations over i), and again use a five-layer MLP as the final readout head.

For vector-valued target functions, we seek for our network to be rotationally equivariant. Thus, our final prediction for each node v_i is given by

$$\widehat{\mathbf{y}}_i = \sum_{k=1}^{K} \sigma_w(\beta_i[k]) \, \widetilde{\mathbf{W}}'[i, :, k] \in \mathbb{R}^d \,,$$

where, as above σ_w is a nonlinear activation and each $\beta_i[k]$ is a gating parameter. Here, however, the gating parameters $\beta_i[k]$ are learned via a two-layer MLP that inputs the tensor $\mathbf{T}[i, :, :]$ and outputs $\boldsymbol{\beta}_i$, a vector of K gating weights for each node.

Further details. We also note that our implementation of the scattering coefficients differs slightly from the exposition in Section 3 for improved performance. Instead of dyadic wavelets of the form $\mathbf{P}^{2^{j-1}} - \mathbf{P}^{2^j}$, we instead use generalized diffusion wavelets of the form $\mathbf{P}^{t_{j-1}} - \mathbf{P}^{t_j}$, where $0 = t_0 < t_1 < \ldots, < t_J$ is an increasing sequence of diffusion scales which are selected by the InfoGain procedure introduced in Johnson et al. [2025b]. Additionally, we did not use a non-linearity while computing the scattering-coefficients since our network is able to learn non-linear relationships in the data via the MLP and the gating function used in (14) and (15). Following Bhaskar et al. [2023], in order to help the geometric scattering transform probe the graph geometry, we also used two Dirac (Kronecker) vectors as additional scalar-valued input signals. To place these Diracs, we first computed the centroid of the vertices in $\mathbb{R}^d$ (using the Euclidean distance). We then placed one Dirac at the vertex closest to this centroid and the other Dirac at the vertex furthest from the centroid (ties broken randomly). Importantly, we note that all of our theoretical guarantees may be readily adapted to this modified version of the vector-valued geometric scattering transform.

D.1 Complexity of diffusion wavelets

We compute the (scalar) diffusion matrix $\mathbf{P}$ and the vector diffusion matrix $\mathbf{Q}$ for each graph as a one-time data preprocessing step and cache their (sparse) tensor data in memory. Since operators are independent for each graph, for large data sets, this is an embarrassingly parallel task and so we also batch-parallelize this step across CPU workers.

For a single graph with n nodes, m edges, vector dimension d, and individual node degrees k_i:

- Constructing $\mathbf{P}$ as a sparse matrix involves building a sparse adjacency matrix, computing a node degree vector, adding a sparse identity matrix and rescaling entries. The total time complexity is $\mathcal{O}(n + m)$, and $\mathbf{P}$ has $n + m$ nonzero entries.
- Constructing $\mathbf{Q}$ as a sparse block-diagonal matrix requires first building a dictionary of neighbor sets ($\mathcal{O}(m)$), and then centering and kernel-rescaling vector node features ($\mathcal{O}(md)$).

Then, singular value decomposition is performed on each transformed node's vector feature matrix $C_i \in \mathbb{R}^{d \times k_i}$, which has complexity $\mathcal{O}(d \sum_{i=1}^{n} k_i^2)$ where $k_i \geq d$. Finally, computing $O_{ij} = O_i O_j^\top$ (with sign-alignment of nodes' left singular vectors) and rescaling by p_{ij} is $\mathcal{O}((n+m)d^3)$. The number of nonzero entries in $\mathbf{Q}$ is $(n+m)d^2$.

These constructions of $\mathbf{P}$ and $\mathbf{Q}$ enable highly efficient geometric scattering layers in ESc-GNN, through sparse matrix multiplication of (dense) scalar and vector feature vectors. Still, in terms of complexity of the forward pass of our model, this layer dominates for typical graphs. For each of the scalar and vector tracks, their first-order scattering has complexity $\mathcal{O}\left(S\, m\, (F+d)\right)$, where S is the number of wavelets used (typically four to ten), F is the number of signal channels (generally, $F = 1$ for the vector track), and d is the dimension ($d = 1$ for scalars). To obtain second-order scattering coefficients, we recursively reapply each (sparse) diffusion operator to all first-order scattering coefficients, and then discard coefficients resulting from higher-pass wavelets reapplied to lower-pass wavelets. Accordingly, for each track, second-order scattering has complexity $\mathcal{O}(n\, F\, d\, S^2)$, where n is the number of nodes.

E Experimental details

We use the PyTorch [Ansel et al., 2024] and PyTorch-Geometric [Fey and Lenssen, 2019] to build our codebase, which is available at `https://anonymous.4open.science/r/esc-gnn-407D`. For the EGNN and TFN models, we adapted code from the "Geometric GNN Dojo" [Joshi et al., 2023] under the MIT license, from their Github repository (`https://github.com/chaitjo/geometric-gnn-dojo`).

E.1 Cross-validation procedure

In our experiments, we split the data into five folds (each containing 20% of the data), and perform five-fold cross validation such that each fold is used as a test set and validation set exactly once, while the remaining three folds make up the training set in each cross-validation step.

E.2 Hyperparameter settings

E.2.1 General

For all models, we optimize for mean squared error (MSE) loss using PyTorch's `AdamW` optimizer, an initial learning rate of 0.001, and a train batch size of 32 (except for TFN, for which we use 16, to prevent out-of-memory errors). Models train for at least 50 burn-in epochs, before the following early stopping is enforced: if validation loss does not improve for 50 consecutive epochs (checking every five epochs), we (1) halve the learning rate and reload the best model weights achieved (by by validation loss), and then (2) quit training after a maximum of two such restarts or a maximum of 500 total training epochs. The final model weights are then extracted from the epoch in which the lowest validation loss was achieved.

For experimental consistency and compatibility with our regression tasks, we standardize some modules across models as appropriate. That is, first, for the graph-level diameter estimation task, we use `sum` and `max` pooling of hidden node features wherever such pooling is needed; for ESc-GNN, LEGS, GCN, GIN, and GAT, these pooled features are input into a five-layer readout MLP with hidden dimensions 128, 64, 32, and 16. Second, for all models (including ours), we use the sigmoid linear unit (SiLU, or 'swish') activation function [Elfwing et al., 2018] in the MLP (except for when another activation function is explicitly prescribed by the baseline model.) Third, when edge weights are used by a model, we compute these weights using a Gaussian kernel K_ϵ from Section 3, with scale parameter ϵ set equal to the squared mean of mean neighbor distances across the final ellipsoid data set for all models.

E.2.2 ESc-GNN (and ESc-GNN-non-equivariant)

For ESc-GNN, when using the InfoGain Wavelets procedure to select wavelet diffusion scales, we set $t_J = 16$, and use information cutoff quantiles of $[0.25, 0.5, 0.75]$ (see Johnson et al. [2025b] for details). We deviated slightly from the method presented in Johnson et al. [2025b] and measured

the convergence of each feature as quantified in the ℓ^1 norm rather than the KL divergence. For the scalar signals, we applied the $n \times n$ diffusion matrix $\mathbf{P}$ to each scalar signal $\mathbf{x}$. For the vector signals, we applied the $nd \times nd$ vector diffusion matrix $\mathbf{Q}$ to each vector-valued signal $\mathbf{w} \in \mathbb{R}^{nd}$. This procedure produced diffusion scales $\{t_j\}_{j=0}^{J} = \{0, 1, 2, 4, 6, 8, 16\}$ for the scalar features) and $\{0, 1, 2, 4, 6, 9, 16\}$ for the vector features (after taking the median across input signals as applicable). This means that that our filter bank consists seven total filters (including the low-pass filter $\mathbf{\Psi}^{t_J}$). In the within-track mixing layers, we set K equal to 16 and 32 for the scalar and vector tracks, respectively, and the hidden layer dimensions of their two-layer mixing MLPs to $[64, 64]$ and $[128, 128]$. (Note the vector track mixing MLP has no biases or activations.) We use sigmoid for the gate activation σ_w.

On the graph-level diameter estimation task, we mitigate overfitting by introducing random perceptron dropout between layers of the readout MLP, with probability 0.7. On the node-level vector target regression task, the final vector gating coefficients MLP has layer widths $[128, 128, K]$, including the output layer, and SiLU activations. As before, we apply sigmoid for the gates' activation function, σ_w.

When ablating the vector track in ESc-GNN to render it non-equivariant, we (1) concatenate the vector feature coordinates to the Dirac scalar features as d separate, additional scalar features, and (2) omit all vector feature operations. (For vector targets, this ablated model then uses the five-layer MLP prediction head defined previously, with an output layer width of the vector dimension d.)

E.2.3 LEGS

For LEGS, we use Dirac scalar node features, as in ESc-GNN, and concatenate the vector feature coordinates as d additional scalar features (analogous to ESc-GNN-non-equivariant). The network employs dyadic-scale wavelets with $J = 4$. Hidden representations are fed into a five-layer MLP with SiLU activations (and no dropout) to produce predictions. For graph-level predictions, we first pool across nodes using both `sum` and `max` operators. For vector targets, the prediction head outputs a vector of dimension d.

E.2.4 GCN, GIN, GAT

For GCN, GIN, and GAT, the input features consist of the vector feature coordinates encoded as d scalar features. Each model uses two layers, with hidden dimensions size 128. Predictions are obtained using the same MLP head strategy as in LEGS.

E.2.5 EGNN and TFN

For the node-level equivariant regression task, we tune the number of layers separately for each task by starting from one and increasing until performance degrades rather than improves. The best-performing configurations use two layers for EGNN and four layers for TFN on the graph-level task; and seven and four layers, respectively, on the node-level task. Prediction heads are implemented as two-layer MLPs (which operate on both `sum` and `max` pooled hidden node features for graph-level predictions).

EGNN expects a scalar node feature embedding; here, we use a single uniform feature with embedding size 128, which is equivalent to a learnable bias. TFN requires specification of the maximum irreducible representation order ℓ, which we tune up to $\ell = 2$ to prevent excessive parameter growth. We find that $\ell = 2$ was best for both tasks. TFN also expects radial edge weights; to this end, we implement Gaussian kernel edge weights, one per edge, for use in message passing.

E.3 Details on the node-level vector target

The direction of $\mathbf{h}(v)$ at each point $v = (x, y, z)$ is chosen to be the outward normal vectors, which is given by the normalized gradient of the functions $f(x, y, z) = x^2/a^2 + y^2/b^2 + z^2/c^2$, i.e.,

$$\mathbf{n} = \frac{\nabla f(v)}{\|\nabla f(v)\|}, \qquad \nabla f(v) = 2\Big(\tfrac{x}{a^2}, \tfrac{y}{b^2}, \tfrac{z}{c^2}\Big).$$

The magnitude of $\mathbf{h}(v)$ is computed using (an approximation of) the $K = 16$ nontrivial eigenfunctions of the Laplace-Beltrami operator on the underlying ellipsoid. (The first eigenfuction is considered

trivial because it is constant and has eigenvalue 0.) To compute these eigenfunctions, we sample 896 more points per point cloud (so that there is a total of 1024 points on each point cloud). We then construct a symmetric, unweighted, k-NN graph, $G^{\text{large}} = (V^{\text{large}}, E^{\text{large}})$ from each of these enlarged point clouds (with $k = 10$) and compute the symmetric-normalized graph Laplacian $\mathbf{L}_{\text{sym}}^{\text{large}} = \mathbf{I} - (\mathbf{D}^{\text{large}})^{-1/2}\,\mathbf{A}^{\text{large}}\,(\mathbf{D}^{\text{large}})^{-1/2}$. We then approximate the first 16 non-trivial eigenfuctions of the Laplace Beltrami operator by the first eigenvectors of $\mathbf{L}_{\text{sym}}^{\text{large}}$.

We then let $\mathbf{g}$ be a real-valued signal defined by V^{large} by

$$\mathbf{g}(v) = \sum_{j=1}^{K} c_j\,\phi_j(v),$$

and rescale $\mathbf{g}$ so that its entries have magnitudes in $[0, a]$, for some fixed $0 < a < 1$, by setting $\widetilde{\mathbf{g}} = a\frac{\mathbf{g}}{\|\mathbf{g}\|_\infty}$. Finally, we define the magnitude of $\mathbf{h}$ by $\|\mathbf{h}(v)\|_2 = 1 + \widetilde{\mathbf{g}}(v)$. We note that by construction, we will have $0 < 1 - a \leq \|\mathbf{h}(v)\|_2 \leq 1 + a$ for all v.

When training and evaluating our network, we use only the 128 points from the original data set. The additional points were merely introduced so that the eigenvectors of the graph Laplacian would be a good approximation of the Laplace-Beltrami operator on the underlying ellipsoid.

E.4 Model runtimes

The run times for each model are for the diameter prediction task and the node-level vector prediction task are displayed in Tables 3 and 4 respectively. We note that all models were trained using one NVIDIA L40S 48 GB GPU, with one exception. Due to its high parameter count and larger memory requirements, we trained TFN using distributed data parallel (DDP) with four such GPUs. Hence we conjecture that, if the model fit on one GPU, its runtimes would roughly quadruple.

Table 3: Epoch number of best validation loss, average training epoch runtime, and validation set inference time on the diameter prediction task. TFN is marked with **** since it was trained on multiple GPUs (while all other models used a single GPU), as explained at the beginning of section E.4.

Model	**Best epoch**	**Avg. train time (s/epoch)**	**Avg. inference time (s)**
ESc-GNN (Ours)	461 ± 56	0.0952 ± 0.0008	0.0168 ± 0.0001
ESc-GNN (non-equivariant) (Ours)	96 ± 93	0.0792 ± 0.0039	0.0072 ± 0.0034
LEGS	356 ± 155	0.0946 ± 0.0014	0.0164 ± 0.0002
GCN	335 ± 132	0.0568 ± 0.0009	0.0038 ± 0.0000
GAT	62 ± 31	0.0662 ± 0.0013	0.0048 ± 0.0000
GIN	130 ± 111	0.0576 ± 0.0004	0.0027 ± 0.0000
EGNN (2-layer)	338 ± 178	0.0856 ± 0.0009	0.0074 ± 0.0001
TFN (4-layer)****	453 ± 72	2.6965 ± 0.0044	0.0880 ± 0.0007

Table 4: Epoch number of best validation loss, average training epoch runtime, and validation set inference time on the node-level vector prediction task. TFN is marked with **** since it was trained on multiple GPUs (while all other models used a single GPU), as explained at the beginning of section E.4.

Model	**Best epoch**	**Avg. train time (s/epoch)**	**Avg. inference time (s)**
ESc-GNN (Ours)	488 ± 4	0.0945 ± 0.0003	0.0164 ± 0.0000
ESc-GNN (non-equivariant) (Ours)	355 ± 184	0.0793 ± 0.0034	0.0101 ± 0.0042
LEGS	402 ± 128	0.0980 ± 0.0016	0.0163 ± 0.0001
GCN	356 ± 176	0.0583 ± 0.0007	0.0035 ± 0.0000
GAT	356 ± 179	0.0624 ± 0.0004	0.0042 ± 0.0000
GIN	330 ± 171	0.0562 ± 0.0004	0.0021 ± 0.0000
EGNN (7-layer)	325 ± 77	0.1552 ± 0.0015	0.0183 ± 0.0001
TFN (4-layer)****	364 ± 123	2.6912 ± 0.0042	0.0884 ± 0.0017