Bridging Equivariant GNNs and Spherical CNNs for Structured Physical Domains*

Colin Kohler
Northeastern University
Boston, MA 02115
kohler.c@northeastern.edu

Purvik Patel
Northeastern University
Boston, MA 02115
pu.patel@northeastern.edu

Nathan Vaska
MIT Lincoln Laboratory
Lexington, MA 02421
nathan.vaska@ll.mit.edu

Justin Goodwin
MIT Lincoln Laboratory
Lexington, MA 02421
jgoodwin@ll.mit.edu

Matthew C. Jones
MIT Lincoln Laboratory
Lexington, MA 02421
matthew.jones@ll.mit.edu

Robert Platt
Northeastern University
Boston, MA 02115
rplatt@ccs.neu.edu

Rajmonda S. Caceres
MIT Lincoln Laboratory
Lexington, MA 02421
rajmonda.caceres@ll.mit.edu

Robin Walters
Northeastern University
Boston, MA 02115
r.walters@northeastern.edu

Abstract

Many modeling tasks from disparate domains can be framed in the same way, computing spherical signals from geometric inputs, for example, computing the radar response of different objects or navigating through an environment. This paper introduces G2Sphere, a general method for mapping object geometries to spherical signals. G2Sphere operates entirely in Fourier space, encoding geometric structure into latent Fourier features using equivariant neural networks and outputting the Fourier coefficients of the continuous target signal, which can be evaluated at any resolution. By utilizing a hybrid GNN-spherical CNN architecture, our method achieves a much higher frequency output signal than comparable equivariant GNNs and avoids hand-engineered geometry features used previously by purely spherical methods. We perform experiments on various challenging domains, including radar response modeling, aerodynamic drag prediction, and policy learning for manipulation and navigation. We find that G2Sphere outperforms competitive baselines in terms of accuracy and inference time, and we demonstrate that equivariance and Fourier features lead to improved sample efficiency and generalization. The source code is available at: https://github.com/ColinKohler/geometry2sphere.

*DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001 or FA8702-25-D-B002. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

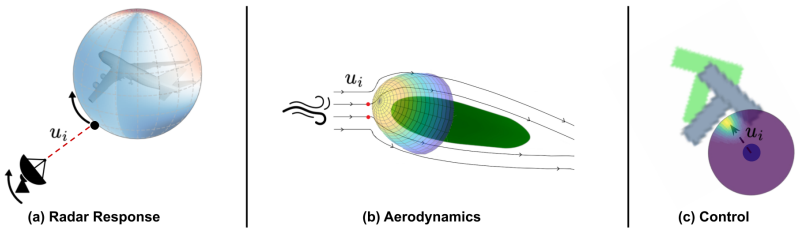


Figure 1: **Spherical Domains.** Across many different domains learning high-frequency spherical signals is a challenging task. (a) In the radar response domain, we learn a function $F_X(u_i)$, which maps the input geometry X and the radar viewing angle u_i to the radar response. (b) In the aerodynamics domain, we learn a function $F_X(u_i)$, which maps the input geometry X and the airflow direction u_i to the corresponding drag coefficient. (c) In the control domain, we learn a function $F_X(u_i)$, which maps the environmental geometry X and the control action u_i to their value.

1 Introduction

Many 3D modeling problems from different domains require solving a common problem, mapping detailed local geometric information to global spherical functions. In this paper, we consider several examples, such as simulating the radar response of an object, predicting the drag characteristics of a vehicle, or modeling an optimal control policy for manipulation (Fig 1). Modeling these types of signals is challenging because it requires a very fine discretization of the domain to accurately capture high-frequency information, such as sharp specular reflections in radar. This can make traditional numerical solvers and simulators expensive [2]. In practice, these approachs limits the potential applications as they are too slow for use in many domains such as inverse radar modeling or high-frequency robot controllers.

Deep learning is a potentially more efficient approach, but current methods also struggle with inefficiency and overfitting. For example, one method of using neural networks to model spherical output signals is to output a tensor of regularly sampled values across the sphere using a spatial grid such as Healpix [25]. However, a large number of samples is necessary to attain a high spatial resolution over the sphere and even then, the model can easily overfit to the specific sample locations. Furthermore, these large input and output spaces can lead to other compounding issues, such as training instability and computational costs [56]. An alternate strategy is to use implicit models, which represent functions in a continuous manner and can be queried at any coordinate rather than outputting values at predefined coordinates. Implicit models have the advantages of being resolution independent, but are relatively inefficient, requiring both a large amount of data and training to achieve good performance[22]. Furthermore, implicit models have been shown to be prone to overfitting to the training coordinates [18] and lose the geometric relationship between the input and output spaces.

In this work, we seek to address these challenges by introducing G2Sphere, a more accurate, general, and efficient approach to mapping object geometries to spherical functions. We model the output signal in Fourier space as a linear combination of spherical harmonic (SH) basis functions, enabling simultaneous prediction of the continuous output function at any resolution. Although previous work has modeled mappings from geometric inputs to spherical functions using spherical harmonics, they rely on predefined mappings to convert geometry to spherical input [20, 26]. In contrast, G2Sphere uses learned spherical embeddings for geometric inputs that are both more general and more flexible. Furthermore, G2Sphere maintains the geometric structure between the input geometry and the output signal by operating entirely in Fourier space and enforcing the end-to-end $\mathrm{SO}(3)$-equivariance. G2Sphere leverages equivariant graph convolutions [24, 40] to encode the geometry, spherical convolutions [15] to render the spherical signal, and pointwise MLP non-linearities to model high-frequency information in the output space. This combination alleviates issues with existing equivariant GNN approaches which have been limited to lower-frequency modeling, primarily due to computational constraints.

Our contributions are as follows:

- A novel $\mathrm{SO}(3)$-equivariant architecture, G2Sphere, for mapping complex geometric inputs to high frequency, continuous spherical functions, utilizing Fourier decomposition.

- Empirical demonstration of the improved accuracy, flexibility, and efficiency of G2Sphere across a wide range of challenging domains, including radar response and aerodynamic drag of 3D meshes, and policy learning for 2D and 3D control.
- Demonstration of zero-shot super-resolution ability and generalization to novel meshes.
- In policy learning, an inference time of only $9ms$ compared to the $156ms$ of Diffusion Policy, a leading state-of-the-art approach, enabling our policy to achieve a high-frequency controller.

2 Related works

Fourier Transform in Machine Learning The Fourier transform (FT) is an important tool in a wide range of mathematical and engineering applications ranging from solving differential equations [16] to quantum mechanics and signal processing [3]. Fourier transforms have also been extensively utilized in machine learning. In image processing and pattern recognition the FT has been used to speed up convolutional neural networks [42], to improve feature representation [47], and to increase the resolution of generative adversarial networks [31]. Lee-Thorp et al. [37] uses the FT in the more modern transformer architecture to improve tokenization and improve performance. Li et al. [38] combine neural operators with the Fourier transform to create the Fourier Neural Operator (FNO) to solve a number of challenging PDE systems and Bonev et al. [7] extend this work to the Spherical FNO, which they use to model weather systems. G2Sphere and Spherical FNO have a number of notable differences. First, the trainable layers of G2Sphere operate entirely in Fourier space, whereas SFNO also has parallel trainable layers operating in the spatial domain. Additionally, SFNOs are typically used to model dynamical systems which have similar continuous input and output spaces, while G2Sphere maps discrete geometric inputs to continuous outputs.

Equivariant Neural Networks Equivariant neural networks constrain their layers to respect transformations under a symmetry group [14, 23, 61]. Neural networks equivariant to the 3D rotation group $\mathrm{SO}(3)$ are used for classifying shapes [20, 15], classifying protein structures [62]. determining 3D orientation [36] , and predicting features of atomic systems [54, 8, 39]. Similar to our method, all these approaches use steerable kernel bases and perform equivariant operations, such as convolutions or tensor products, but almost all are limited to harmonics of low degree (≤ 10). As such, G2Sphere is capable of capturing significantly more detail and fine-grained structure due to our use of a much higher maximum frequency (≈ 40). Esteves et al. [20] and Cohen et al. [15] model data over the sphere, but use a predefined mapping to convert the input into a spherical signal. They also focus on learning functions with low-dimensional discrete outputs, e.g. classification. Ha and Lyu [26] also uses a predefined mapping to convert brain geometry to spherical signals and a spherical UNet with frequency up-sampling to do segmentation. Lee and Cho [36] extracts features from 2D images using a resnet and projects them onto a sphere before applying spherical convolutions. They also operate at a fixed low maximum frequency of 5. Unlike these methods, G2Sphere uses a learned mapping to convert object geometries to spherical signals, enabling the broad applicability of G2Sphere, enabling the use of the same model for a wide range of applications.

Spherical Domains Spherical functions naturally arise in a number of disparate domains including physical modeling and decision-making tasks. In the radar response domain (Fig. 1a), the reflected response from a 3D object is a function defined over the sphere, where each direction corresponds to an incident wave direction. In the aerodynamic domain (Fig. 1b), the drag function can be represented as a spherical signal, capturing how air resistance varies across orientations of a body in flow. Current machine learning methods for reconstructing radar and aerodynamic signatures do not explicitly model the inherent geometric and physical symmetries of 3D objects, e.g., rotational and permutation symmetries [63, 46, 64, 19]. Geometric deep learning has emerged as a class of algorithms that can address some of these challenges by directly encoding symmetry and other physical priors into neural networks [60, 9, 6, 23], but applications to settings that map fine 3D geometries to corresponding high-frequency signals like radar and aerodynamics has been lacking. Similarly, control policies can be framed as functions over spherical action domains (Fig. 1c), particularly when considering continuous action spaces and geometric state spaces. While several recent works have explored incorporating geometric structure into policy learning [58, 28, 29, 29], they primarily focus on extending prior methods using equivariant neural networks, whereas G2Sphere redefines the value function as a composition of harmonic basis functions.

3 Background and Problem Statement

Problem Setting We consider the problem of mapping 3D geometric data, such as object meshes or point clouds, to *spherical signals* representing physical systems, such as radar response or drag. We consider spherical signals as continuous functions $f\colon S^d \to \mathbb{R}^c$ where S^d is the 1-sphere or 2-sphere and c is the number of channels in the signal. Given some input geometry $X \subset \mathbb{R}^3$, the objective is to learn a functional mapping $F : X \mapsto f_X$. Since full observations of f_X are rare in practice, we assume a data set consisting only of partially observed output. That is, a dataset $\mathcal{D} = \{(X_i, \{u_{ij}\}, \{y_{ij}\})\}_{i=1,j=1}^{N,\ M}$, where samples consist of meshes or point clouds X_i, a set of viewing directions $\{u_{ij}\}$, where each $u_{ij} \in S^d$, and the value of the spherical signal $y_{ij} = f_{X_i}(u_{ij}) \in \mathbb{R}^c$ at each u_{ij}. Since the output signals are only partially observed, we consider generalization to new viewing directions u_{ij} and new geometries X_i.

Equivariance The value $f_X(u)$ may be considered as a property of the geometry X defined with respect to viewing direction $u \in S^2$. We assume that both the geometry X and the direction u are defined with respect to the same coordinate frame. Thus if the coordinate frame is rotated, both X and u are rotated, and $f_X(u) = f_{RX}(Ru)$ should be invariant. This property is equivalent to the equivariance of the functional mapping F; if X is rotated to RX, the spherical signal f_X is rotated to $f_{RX} = Rf_X$, preserving the relationship between the input geometry and the output signal. That is, $F(RX) = RF(X)$. This property can be enforced in the model architecture to ensure that the learned model respects this property. Since the composition of equivariant functions is equivariant, it is sufficient to ensure that each component of the architecture is equivariant.

SO(3)-Equivariant Neural Networks. For physical 3D systems, since the orientation of the coordinate frame is arbitrary, many task functions f should be $\mathrm{SO}(3)$-equivariant. Broadly speaking, there are two types of $\mathrm{SO}(3)$-equivariant models; those which focus on modeling high frequency spherical signals such as Spherical CNN or Spherical Fourier Neural Operator [15, 7] which have been applied to weather prediction and GNN-based approaches such as Equiformer [39] which specialize in lower frequency signals over geometric graphs, e.g. in materials property prediction. In spherical CNNs, features are defined as signals $f\colon \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} = S^2$ or $\mathcal{X} = \mathrm{SO}(3)$. The model architecture consists of alternating layers of $\mathrm{SO}(3)$ group convolutions and pointwise activations applied on $\mathrm{SO}(3)$ or S^2. In GNN-based approaches, features are defined as node embeddings that transform according to $\mathrm{SO}(3)$-irreducible representations. Linear layers are implemented via messaging passing, where node features are tensored with spherical harmonic embeddings of their relative positions and then projected back to irreps using Clebsh-Gordan coefficients [23]. Non-linearities are typically implemented as radial or gated equivariant activations.

Spherical Fourier Features Cohen et al. [15] provide an efficient solution for handling $\mathrm{SO}(3)$-equivariant signals in the Fourier domain using the truncated basis of spherical harmonics Y_m^l for signals defined over S^2 and Wigner D-matrix coefficients D_{mn}^l for signals over $\mathrm{SO}(3)$. Writing $f\colon \mathrm{SO}(3) \to \mathbb{R}$ in terms of the D_{mn}^l and then truncating to a given frequency $l \leq L$ gives the approximation $f(g) \approx \sum_{l=0}^{L} \sum_{m=0}^{2l+1} \sum_{n=0}^{2l+1} c_{mn}^l D_{mn}^l(g)$. The $\mathrm{SO}(3)$ group convolution can be efficiently computed in the Fourier domain in terms of the coefficients $\{c_{mn}^l\}$ by convolution theorem. See Cohen et al. [15] for additional details on the $\mathrm{SO}(3)$ group convolution.

4 Method

A G2Sphere model consists of two key components: (1) an encoder, which compresses the input data into a compact latent representation, and (2) a decoder, which decomposes this latent representation into the spherical output using spherical harmonics (Fig. 2). Encoder-decoder architectures such as U-Net [65] and Transformers [55] are attractive methods for prediction tasks with high-dimensional inputs, such as 3D geometric data, due to latent space compression. The encoder-decoder design is a good fit for our problem since we map between two different types of geometric data. It also allows us to match the geometry of the input and output spaces to the architecture of the encoder and decoder. Using these learned and generalizable mappings from geometry to spherical signals, G2Sphere can be applied to tasks that were previously unattainable or computationally prohibitive

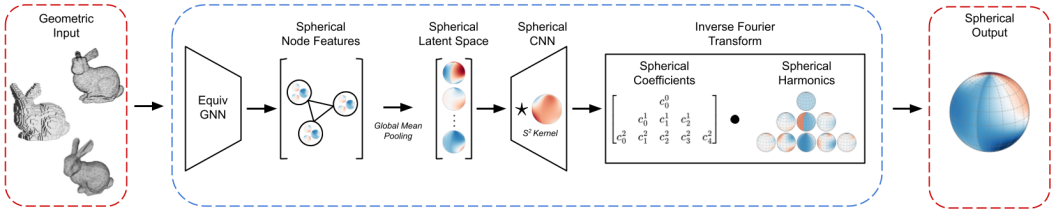


Figure 2: **G2Sphere Architecture.** Illustration of the G2Sphere (G2S) model. First, the geometric input is converted from real space (red dotted-line) to Fourier space (blue dotted line) using an $\mathrm{SO}(3)$-equivariant encoder. We decode this spherical latent representation into a set of Fourier coefficients c_m^l representing the continuous spherical signal. Finally, these coefficients are combined with a grid of pre-computed harmonic basis functions to convert back to real space.

under previous methods [20, 15, 40]. We detail the exact implementations utilized in our experiments in Appendix G.

4.1 Encoder

In a G2Sphere model, the first step is to encode the local and global information of the 3D input into Fourier space, thereby ensuring that Euclidean and permutation symmetries are correctly respected within our model. This is accomplished by utilizing an equivariant neural network encoder, with the exact architecture varying depending on the type of input data. Broadly speaking, we define 3D geometric data as information that describes the shape, position, and spatial properties of objects in three-dimensional space. This can include various types of data including point clouds, voxel grids, object meshes, and RGB-D images. When dealing with dense inputs, such as point clouds or object meshes, equivariant transformers such as Equiformer V2 [40] are used. However, when dealing with voxel grids or RGB-D images, $E(n)$-equivariant Steerable CNNs [61, 11] are a more natural choice. These types of models are constrained to use low-frequency spherical features, i.e. $l_{max} \leq 10$, due to the computational costs of higher-frequency features scaling significantly when applied on a per node basis. In each case, the input signal is initially spatially dispersed in $\mathbb{R}^3$ and at the end of the encoder is aggregated into a single feature vector representing the multi-channel spherical signal in Fourier space.

In Figure 2, we illustrate a G2Sphere model with mesh inputs which uses Equiformer V2 [40], a $\mathrm{SO}(3)$-equivariant transformer commonly used on 3D geometric data, as the encoder. The mesh is embedded as a geometric graph with node and edge features. The node feature at node i is the position $\mathbf{x}_i \in \mathbb{R}^3$ and the edge features are the spherical harmonic embedding of the relative positions $e_{ij} = Y^l(\mathbf{x}_j - \mathbf{x}_i)$, where Y^l: $\mathbb{R}^3 \to \mathbb{R}^{2l+1}$ are the spherical harmonics and i, j are vertices connected by an edge. Then a set of node-wise spherical features are learned over the graph which are mean pooled to form a multi-channel spherical latent space.

4.2 Decoder

Once the input has been projected into Fourier space using the encoder, we use operations that preserve the $\mathrm{SO}(3)$ symmetry of the latent representation. Specifically, we use $\mathrm{SO}(3)$-equivariant group convolutions to form a Spherical CNN [15] decoder. The output of the final layer of the decoder $\hat{f}$ represents an N-channel signal over S^2 in the Fourier domain. That is, $\hat{f}_k = (c_{km}^l)_{m=0,l=0}^{2l+1,L}$ are the coefficients of the spherical harmonics up to frequency L for $1 \leq k \leq K$. To convert the signal back to real space, we apply the inverse Fourier transform and evaluate these K signals on the 2-sphere, at a position specified by spherical coordinates (θ, ϕ). That is,

$$f(\theta, \phi) = \left(\sum_{l=0}^{L} \sum_{m=0}^{2l+1} Y_m^l(\theta, \phi) c_{km}^l \right)_{k=1}^{K},$$

where Y_m^l are the spherical harmonics of degree l and order m. In practice, we pre-compute the spherical harmonics basis functions at the desired grid resolution to allow for fast evaluation. See Appendix D for additional details on our harmonics implementations.

Table 1: **Mesh to Spherical Functions.** Mean square error of G2Sphere (G2S) and baselines on the radar and drag domains. Performance is averaged across 3 random seeds with ± standard error.

Domain	Mesh Type	Transformer	Equiformer	Spherical CNN	G2S	G2S+TSNL
Radar	Frusta	$0.201 \pm$ 2e-4	$0.271 \pm$ 3e-4	$0.438 \pm$ 1e-6	$0.221 \pm$ 1e-5	$\mathbf{0.195 \pm 9e\text{-}4}$
	Asym	$0.179 \pm$ 5e-4	$0.257 \pm$ 6e-4	$0.496 \pm$ 1e-6	$\mathbf{0.123 \pm 2e\text{-}4}$	$0.128 \pm$ 4e-4
Drag	Pods	$0.064 \pm$ 1e-2	$0.0672 \pm$ 4e-3	$0.075 \pm$ 3e-4	$\mathbf{0.062 \pm 2e\text{-}3}$	$0.064 \pm$ 6e-3

One drawback to using harmonics to approximate functions in this manner is that if the maximum frequency L is low, then their resolution will also be low. The SH excel at capturing smooth, low-frequency components, but struggle with sharp features or discontinuous behavior unless a large number of harmonics are used, i.e. high L. As a result, traditional equivariant decoder architectures, which operate with a relatively small number of harmonics ($L \leq 10$), struggle in domains where higher output fidelity is required. G2Sphere uses two techniques to improve output fidelity: *frequency up-sampling* where we gradually increase the frequency from the encoder's frequency L_{enc} to the output frequency L_{dec} and *trainable spherical non-linearities* (TSNL) [7], where an MLP is applied pointwise across the sphere resulting in arbitrarily high frequency. We implement a more general version of TSNL than Bonev et al. [7] by applying inverse Fourier transform for each frequency l separately, giving the MLP more input channels.

Trainable spherical non-linearity (TSNL) To further increase L in the output space, we apply a learned MLP non-linearity pointwise to the spherical output signal. That is, given decoder outputs c^l_{km}, we apply the FT to each frequency l separately yielding $f = \left(\sum_{m=0}^{2l+1} Y^l_m c^l_{km}\right)_{k,l}$, a $K \cdot L$–channel spherical signal $f\colon S^2 \to \mathbb{R}^{K \cdot L}$. Let M be a trainable MLP $M\colon \mathbb{R}^{K \cdot L} \to \mathbb{R}^c$. The final output is then evaluated at $u \in S^2$ as $M(f(u))$. This allows the maximum frequency of the output signal $M \circ f$ to be unbounded when using typical non-linearities inside M. While this could also be achieved by setting M to be a single fixed non-linearity, a trainable M is more controllable and can introduce higher frequencies without adding artifacts.

The TSNL is equivariant up to the sampling error introduced by the inverse Fourier Transform used to convert from the frequency domain features to real space signals on a fixed spherical grid. Since M is applied identically at each point in the sphere, it is $\mathrm{SO}(3)$ equivariant by construction. While the use of a finite spherical sampling can introduce minor numerical artifacts, these lead to negligible equivariance error in practice. This approximation is common in many equivariant architectures such as SFNO [7], Spherical CNN [15], and Gauge Equivariant Mesh CNNs [17]. Notably, Cohen et al. [15] explicitly measure this equivariance error and find it to be minimal ($< 2e - 6$).

Frequency Up-sampling Inspired by Ha and Lyu [26], while increasing L in the decoder, we gradually reduce the number of channels in our latent representation (See Appendix G). To maintain equivariance during this operation, we need to account for the relationship between the Fourier and real space. This is accomplished through the use of a regular non-linearity (as in [15] or [17]), where we map the signal to real space using the inverse Fourier transform (IFT), apply a point-wise non-linearity to the values of the spherical signal, and then perform the Fourier transform (FT), with higher frequency resolution, to convert back to the frequency domain. As with the TSNL, this operation is $\mathrm{SO}(3)$-equivariant up to the same minor sampling error introduced by the IFT, which is negligible in practice. By combining a number of these non-linearities in our decoder, we are able to achieve a much higher maximum frequency (up to $L = 40$) than previous works using equivariant architectures with dense geometric inputs, e.g. object meshes [5, 33]. This frequency upsampling mechanism is a key architectural component that enables our model to produce high-fidelity, fine-grained reconstructions, directly contributing to the improved high-resolution radar outputs shown in Figure 3 and discussed in Section 5.1.

5 Experiments

To demonstrate the generality of G2Sphere, we systematically evaluate across three different domains: two supervised learning domains, where we learn a mapping from object meshes to spherical radar and

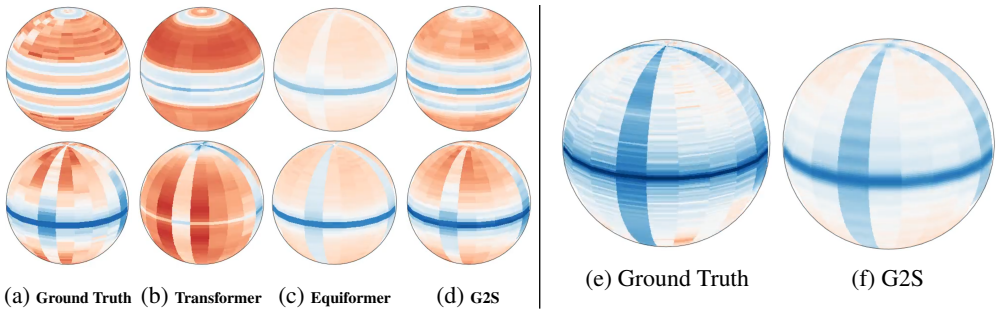


Figure 3: **Left: Radar Response.** Outputs generated by the ground truth first-principles simulator, two baseline methods and G2Sphere. G2Sphere does a good job in reconstructing the overall projected shape characteristics and correctly capturing both types of radar scattering responses whereas the Transformer model underfits and the Equiformer model overfits. **Right: Zero-Shot Super-Resolution.** G2Sphere trained on 60×20 and tested on 180×20.

drag signals, and a policy learning domain, where we learn a mapping between states and state-action values. These evaluations benchmark the performance of G2Sphere on dense (e.g., radar response) and sparse (e.g., drag coefficient) spherical signals and with complex (object mesh) and simple (object keypoint) input geometries. G2Sphere consistently outperforms the prior state-of-the-art in all domains. In the following, we provide an overview of each domain, our evaluation methodology, and our key findings.

5.1 Radar and Drag Modeling from Meshes

Many modeling tasks take an object mesh as input and predict a physical property of that object. If the property depends on a direction vector, that is, a spherical coordinate (θ, ϕ), then the output may be represented as a continuous function defined on the 2-sphere. We evaluate the performance of G2Sphere against baselines on two such tasks, predicting radar response and aerodynamic drag.

Radar Prediction Due to the expense of collecting real-world radar data, we simulate radar responses for a variety of different meshes using a physical optics method to approximate the electromagnetic waves [4]. Specifically, we generate two mesh datasets, *Asym* and *Frusta*. See Fig. 10 for examples. The *Asym* dataset is a collection of randomly scaled basic shapes (cubes, cylinders, and spheres), with randomly generated protrusions to make the radar response asymmetric. The *Frusta* dataset represents a collection of complex 3D shapes, where each mesh is a combination of several basic components stacked together to form roll-symmetric objects. Due to this symmetry, we evaluate each frusta mesh over 360 orientations on the non-symmetric axis, $\theta \in [0, 2\pi]$ to generate the associated radar responses. The *Asym* meshes are evaluated over both axes, $\theta \in [0, 2\pi], \phi \in [0, \pi]$, where θ is discretized over 60 values and ϕ over 20.

Drag Prediction We use a high-performance Computational Fluid Dynamics (CFD) simulator to generate a *Pods* dataset of mesh and drag coefficient samples of pod-like geometrical shapes. There are some differences relative to the radar dataset. First, as drag-coefficients are of most interest for flying objects (e.g. aircraft) the drag coefficients are only simulated for a cone in the front of the objects with $\theta, \phi \in [-20, 20]$. Secondly, a large contributing factor in the drag coefficients are the set of flight conditions (altitude, speed, etc.) which we treat as global parameters and append to the latent representation after encoding the mesh. The *Pods* dataset contains $10,000$ samples (mesh, flight conditions, drag) each with a single drag coefficient for a specific angle (θ, ϕ), which defines the orientation of the incoming flow relative to the centerline of the Pod shape. Additional details on the radar and drag datasets can be found in Appendix J.

G2Sphere Architecture We use Equiformer v2 [40] to encode the input mesh into a set of latent representations for each vertex in the mesh. These latent vectors are then combined into a single feature vector using a global average pooling layer. The decoder is a spherical CNN with several layers of spherical convolutions followed by the spherical non-linearity introduced in Sec. 4.2 to upsample the frequency from L_{enc} to L_{dec}. We use $L_{enc} = 5$ for both tasks but use $L_{dec} = 40$ for

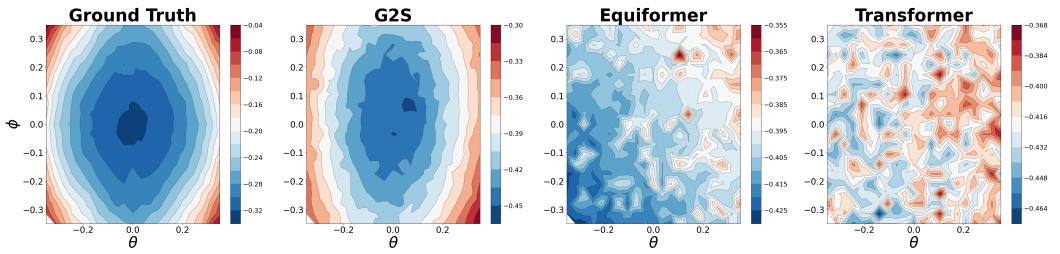


Figure 4: **Generalizing to Unseen Objects.** Full drag prediction on test sample not included in the *Pods* dataset. Unlike in our training dataset, we compute the entire drag prediction from -20 to 20 degrees (shown in radians here). Despite being trained on single coordinate points, G2Sphere is able to reconstruct the general shape of the drag function, whereas the baseline implicit models overfit to specific points.

the radar domain and $L_{dec} = 5$ for the drag domain. We examine two variants of G2Sphere, one with the trainable spherical non-linearity (G2S+TSNL), and one without (G2S).

Baselines We compare the performance of G2Sphere against three competitive baselines for processing meshes, *Transformer*, *Equiformer*, *Spherical CNN*. The *Transformer* model is inspired by other prominent mesh-based transformer architectures [52, 41, 21]. It tokenizes the mesh into spatial and structural descriptors as in [21], and uses a transformer encoder with an MLP decoder to generate the predicted response. The *Equiformer* [40] model resembles the G2Sphere, but directly maps the latent representation to the discrete set of spherical values. The *Spherical CNN* [20] model uses a ray-based method to map the input geometries onto the sphere, followed by a series of spherical convolutions. This can be considered a G2Sphere model where the learned encoder mapping is replaced with a predefined map. We use explicit models for the radar prediction task, where the baseline models output the full radar response for a given mesh. For the drag prediction task, we use implicit models where the coordinates are passed as input to the baseline models and appended onto the latent representation. This is necessary as, unlike in the radar domain, our drag dataset only has partially observed outputs and therefore lacks the dense targets required by explicit models. See Appendix G for additional information on model architectures and training.

G2Sphere performance We report the performance of G2Sphere and the baselines on both radar and drag tasks in Table 1. G2Sphere obtains the lowest mean squared error compared to any of the baseline methods. Further, as the complexity of the task increases from the roll-symmetric *Frusta* shapes to *Asym* shapes, the difference in performance between G2Sphere and the baselines increases. This implies that G2Sphere is more adept at predicting dense output spaces than methods that output discrete values. While exact error rates for radar prediction vary depending on the task specifications, the Transformer baseline and G2Sphere fall within the most commonly cited 10 to 20% error range [30]. Fig. 3 (Left) shows examples of the spherical radar signals generated by each model using meshes from the *Asym* dataset. When comparing the predictions generated by the equivariant models (Equiformer and G2Sphere), we can clearly see that G2Sphere captures significantly more detail and fine-grained structure which we attribute to the higher maximum frequency. Additionally, we can see that the Spherical CNN does very poorly, suggesting that the ray-based mapping from geometry to sphere does not capture the geometric information required for the radar prediction task. Similar to the radar task, we find that G2Sphere is able to achieve good performance on the drag task despite being trained on a sparse set of coordinate values. We note that all of our models achieve an error of around 6%, which falls within the single-digit percentage errors benchmark for aerodynamics [43]. In the following, we explore two interesting zero-shot capabilities of G2Sphere.

Zero-Shot Super-Resolution. G2Sphere outputs coefficients of Fourier basis functions and thus gives a continuous spherical signal which can be evaluated at arbitrary resolution. Therefore, G2Sphere can be trained on only low resolution radar data and evaluated at a higher resolution, i.e. a zero-shot super-resolution task. Fig. 3 (Right) shows an example where we train G2Sphere on 61×21 ($\theta \times \phi$) resolution radar data and transfer to 180×21 resolution. G2Sphere is the only model among the benchmarks (Transformer, Equiformer) that can do zero-shot super-resolution.

Table 2: **Policy Performance.** Comparison of G2Sphere (G2S) against baselines on the PushT and PyBullet Drone domains. We compare max performance and, in parentheses, the average of last 10 checkpoints, each averaged across 50 different initializations. G2Sphere significantly outperforms both an equivariant IBC model and the Diffusion Policy.

Domain	Task	IBC	E-IBC	Diffusion	NE-G2S	G2S
PushT	Fixed	0.85 (0.81)	0.98 (0.95)	0.95 (0.91)	0.97 (0.95)	**1.0 (0.98)**
	Random	0.62 (0.54)	0.81 (0.76)	0.71 (0.69)	0.74 (0.70)	**0.92 (0.89)**
PyBullet Drones	FlyToTarget	0.94 (0.87)	0.98 (0.92)	0.96 (0.94)	**1.00 (0.95)**	**1.00 (0.97)**
	FlyThroughGate	0.90 (0.83)	0.94 (0.87)	0.94 (0.92)	0.92 (0.90)	**0.98 (0.95)**

Generalization to unseen Object Geometries. Our drag dataset contains only a single drag coefficient corresponding to a single (θ, ϕ) for each object. In this experiment, we evaluate G2Sphere's ability to generalize to objects held out during training and predict the full drag response cone, $(\theta, \phi) \in [-20, 20]$. Fig. 4 shows the generalization capability of G2Sphere when compared to the baselines. When compared to these implicit baseline models, G2Sphere demonstrates a stronger ability to generalize, effectively capturing the overall shape of the drag cone and producing more accurate predictions. Although all the baseline models perform well on the training data, here we see that they have overfit to specific (θ, ϕ) coordinates, resulting in poor generalization.

5.2 Policy Learning

In order to demonstrate the flexibility of G2Sphere for high-frequency continuous spherical signal learning, we apply G2Sphere on four policy learning tasks from two benchmarks [22, 48]. Although there are many potential policy learning algorithms to which we could apply G2Sphere, in this work we explore the application of G2Sphere to behavioral cloning (BC) [49]. BC casts the policy learning task as a supervised learning problem where the agent learns to imitate a set of expert demonstrations. Following Florence et al. [22], we formulate BC as a conditional energy-based modeling (EBM) [35] problem. EMBs define a policy through an energy function which assigns an energy value to each state-action pair. We compare G2Sphere to two prominent EBM policy learning methods, Implicit Behavior Cloning (IBC) [22] and Diffusion Policy [13], and demonstrate that G2Sphere outperforms these baselines both in terms of final performance and sample efficiency.

Training spherical value functions We use techniques from the EBM literature to train our G2Sphere EBM. Given a dataset of state-action samples $\{s_i, a_i\}$, where $s_i \in \mathbb{R}^N$ and $a_i \in \mathbb{R}^M$, we aim to learn an energy function $E : \mathbb{R}^{N+M} \to \mathbb{R}$. We train using contrastive learning [34] by generating a set of non-expert actions a'_j and utilizing an InfoNCE-style loss function [45] where the model must predict the true expert action a_i from the non-expert actions a'_j.

The actions a are 2D or 3D velocity vectors normalized to be in the unit ball. In order to model the energy function in Fourier space, we decompose a into an action magnitude $\|a\|$ and an action direction $\hat{a} = a/\|a\| \in S^d$. For 2D actions, the energy function is evaluated

$$E_\theta(s, a) = \sum_{l,m} (F_\theta(s_i))^l_m B^l_m(a),$$

where F_θ is a G2Sphere model outputting Fourier coefficients c^l_m and B^l_m are polar harmonics (Appendix E), which gives a basis of functions over the unit ball. For 3D actions, the magnitude is treated as an implicit variable and $E_\theta(s, a) = \sum_{l,m} (F_\theta(s_i, \|a\|))^l_m Y^l_m(\hat{a})$.

G2Sphere architecture Following Chi et al. [13], we use keypoint observations which reduce the underlying geometries of objects in the scene to a set of keypoints that capture the spatial and structural information in the environment. Although we could still use an equivariant GNN encoder, given the small size and lack of variability of the input, an equivariant MLP is simpler. In the *PushT* domain, we have 2D keypoints and actions and therefore use $\mathrm{SO}(2)$-equivariant linear layers. The *PyBullet Drones* domain is a 3D navigation task, and we use $\mathrm{SO}(3)$-equivariant linear layers. The decoder remains a spherical CNN but with a small modification for the 2D domains to perform

Table 3: **Inference Speed (ms).** Comparison of inference speeds (ms) on an Nvidia Titan.

Domain	IBC	E-IBC	Diff	NE-G2S	G2S
Drones	32	83	156	3	9

convolutions on the 1-sphere. To separate the impact of equivariance and representing the output in Fourier space, we consider a non-equivariant version of G2Sphere (NE-G2S) where we replace the equivariant MLPs with normal MLPs but still output the Fourier coefficients. Additional architectural and training details for both G2Sphere and the baselines can be found in Appendix G.

PushT Adapted from [22, 13], the PushT task (Fig. 13a) requires pushing a T-shaped block (gray) to a fixed target (green) with a circular end-effector (blue). Object geometries are represented by a series of keypoints denoting the pose of the agent, block, and goal. We examine two variants: a fixed-goal variant where the target pose is always set to the pose in Fig. 13a and a randomized-goal variant where the target pose is randomly sampled within the workspace. The metric for performance is target coverage area between the block and the goal pose. We compare the performance of G2Sphere against two versions of IBC, standard (IBC) and equivariant (E-IBC), and Diffusion Policy. The results are shown in Table 2. We find that G2Sphere outperforms all other methods using the best checkpoints almost always achieves a perfect score in the fixed-variant. Additionally, we can see that the addition of equivariance stabilizes training instability leading to more consistent performance.

PyBullet Drones We use the PyBullet Drones benchmark Panerati et al. [48] to study the performance of G2Sphere in a 3D positional control domain. Specifically, we examine two tasks of varying difficulty, *FlyToTarget* (Fig. 13b) and *FlyThroughGate* (Fig. 13c). In the *FlyToTarget* task, the drone must take off from a landed configuration and fly to the target position. In the *FlyThroughGate* task, the drone is initialized in a flying configuration and must fly through a gate which is randomly posed around the drone. The observation space is composed of 4 keypoints for the drone, 4 keypoints for the gate, and a single keypoint for the target position. The action space is a delta motion from the current position which is passed to a PID controller. The agent operates at $30hz$ and the PID controller at $240hz$. The results are shown in Table 2. We see that similar to the PushT results, G2Sphere outperforms all other methods and provides increased training stability.

Inference Speed It is critical to have a fast inference speed for many closed-loop real-time control tasks, such as robotic control. One of the known drawbacks to diffusion is that the denoising process requires a number of steps to optimize the trajectory. This is somewhat mitigated in the Denoising Diffusion Implicit Models (DDIM) approach[13], which has enabled relatively fast inference times of $100ms$. However, this does limit diffusion policies from many high-frequency control tasks, such as policies enabled with force-feedback. G2Sphere, on the other hand, has extremely fast inference time due to our use of grids of pre-computed harmonic function values, which can accurately predict the energy distribution in a single forward pass. Table 3 demonstrates the differences in inference speed for G2Sphere, IBC, and diffusion, with equivariant G2Sphere taking only $9ms$ compared to $156ms$ for Diffusion Policy.

6 Conclusion

In this work, we present G2Sphere, a novel approach to learning mappings from complex geometric inputs to continuous spherical signals. We evaluated G2Sphere across a wide range of challenging domains including radar response prediction, drag modeling, and policy learning for navigation and demonstrated improved performance on all tasks. By employing a fully $\mathrm{SO}(3)$-equivariant architecture that learns the mapping from object geometry directly to the spherical Fourier domain, G2Sphere helps close the gap in learning high-fidelity spherical signal representations from geometric data. Additionally, we demonstrated G2Sphere is capable of zero-shot super-resolution, full-resolution generalization to unseen geometries, and vastly improved inference times.

Acknowledgments

This work is supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001 or FA8702-25-D-B002. We acknowledge the MIT SuperCloud, the Lincoln Laboratory Supercomputer Center, and the Northeastern University Discovery HPC cluster for providing resources that contributed to the research results in this work. R. Walters was supported by NSF awards #2442658, #2134178, #2314182, #2107256.

References

[1] Milton Abramowitz and Irene A Stegun. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, volume 55. US Government printing office, 1948.

[2] D. Andersh, J. Moore, S. Kosanovich, D. Kapp, R. Bhalla, R. Kipp, T. Courtney, A. Nolan, F. German, J. Cook, and J. Hughes. Xpatch 4: the next generation in high frequency electromagnetic modeling and simulation software. In *Record of the IEEE 2000 International Radar Conference [Cat. No. 00CH37037]*, pages 844–849, 2000. doi: 10.1109/RADAR.2000.851945.

[3] George B Arfken, Hans J Weber, and Frank E Harris. *Mathematical methods for physicists: a comprehensive guide*. Academic press, 2011.

[4] Constantine A Balanis. *Advanced engineering electromagnetics*. John Wiley & Sons, 2012.

[5] 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.

[6] Alexander Bogatskiy, Brandon M. Anderson, Jan T. Offermann, Marwah Roussi, David W. Miller, and Risi Kondor. Lorentz group equivariant neural network for particle physics. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 992–1002. PMLR, 2020. URL `http://proceedings.mlr.press/v119/bogatskiy20a.html`.

[7] Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. In *International conference on machine learning*, pages 2806–2823. PMLR, 2023.

[8] Johannes Brandstetter, Rob Hesselink, Elise van der Pol, Erik J. Bekkers, and Max Welling. Geometric and physical quantities improve E(3) equivariant message passing. *CoRR*, abs/2110.02905, 2021. URL `https://arxiv.org/abs/2110.02905`.

[9] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, abs/2104.13478, 2021. URL `https://arxiv.org/abs/2104.13478`.

[10] E.C. Burt and T.G. Moore. High frequency rcs prediction theory, 1991.

[11] Gabriele Cesa, Leon Lang, and Maurice Weiler. A program to build E(N)-equivariant steerable CNNs. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=WE4qe9xlnQw`.

[12] Z. Chance, A. Kern, A. Burch, and J. Goodwin. Differentiable point scattering models for efficient radar target characterization, 2022. URL `https://arxiv.org/abs/2206.02075`.

[13] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2024.

[14] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[15] Taco Cohen, Mario Geiger, Jonas Kohler, and Max Welling. Spherical cnns. In *International Conference on Learning Representations*, 2018.

[16] James W Cooley, Peter AW Lewis, and Peter D Welch. The fast fourier transform and its applications. *IEEE Transactions on Education*, 12(1):27–34, 1969.

[17] Pim De Haan, Maurice Weiler, Taco Cohen, and Max Welling. Gauge equivariant mesh cnns: Anisotropic convolutions on geometric graphs. In *International Conference on Learning Representations*, 2021.

[18] Juliette Decugis, Alicia Y Tsai, Max Emerling, Ashwin Ganesh, and Laurent El Ghaoui. The extrapolation power of implicit models. *arXiv preprint arXiv:2407.14430*, 2024.

[19] Bingchen Du, Ennan Shen, Jiangpeng Wu, Tongqing Guo, Zhiliang Lu, and Di Zhou. Aerodynamic prediction and design optimization using multi-fidelity deep neural network. *Aerospace*, 12(4), 2025. ISSN 2226-4310. doi: 10.3390/aerospace12040292. URL `https://www.mdpi.com/2226-4310/12/4/292`.

[20] Carlos Esteves, Christine Allen-Blanchette, Ameesh Makadia, and Kostas Daniilidis. Learning so (3) equivariant representations with spherical cnns. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 52–68, 2018.

[21] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. Meshnet: Mesh neural network for 3d shape representation, 2018.

[22] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[23] Mario Geiger and Tess Smidt. e3nn: Euclidean neural networks, 2022. URL `https://arxiv.org/abs/2207.09453`.

[24] Mario Geiger, Tess Smidt, Alby M., Benjamin Kurt Miller, Wouter Boomsma, Bradley Dice, Kostiantyn Lapchevskyi, Maurice Weiler, Michał Tyszkiewicz, Simon Batzner, Dylan Madisetti, Martin Uhrin, Jes Frellsen, Nuri Jung, Sophia Sanborn, Mingjian Wen, Josh Rackers, Marcel Rød, and Michael Bailey. Euclidean neural networks: e3nn, April 2022. URL `https://doi.org/10.5281/zenodo.6459381`.

[25] Krzysztof M Gorski, Eric Hivon, Anthony J Banday, Benjamin D Wandelt, Frode K Hansen, Mstvos Reinecke, and Matthia Bartelmann. Healpix: A framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622 (2):759, 2005.

[26] Seungbo Ha and Ilwoo Lyu. Spharm-net: Spherical harmonics-based convolution for cortical parcellation. *IEEE Transactions on Medical Imaging*, 41(10):2739–2751, 2022. doi: 10.1109/TMI.2022.3168670.

[27] Philippe Hansen-Estruch, Ilya Kostrikov, Michael Janner, Jakub Grudzien Kuba, and Sergey Levine. Idql: Implicit q-learning as an actor-critic method with diffusion policies. *arXiv preprint arXiv:2304.10573*, 2023.

[28] Haojie Huang, Dian Wang, Xupeng Zhu, Robin Walters, and Robert Platt. Edge grasp network: A graph-based se (3)-invariant approach to grasp detection. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3882–3888. IEEE, 2023.

[29] Haojie Huang, Owen L Howell, Dian Wang, Xupeng Zhu, Robert Platt, and Robin Walters. Fourier transporter: Bi-equivariant robotic manipulation in 3d. International Conference on Learning Representations, 2024.

[30] John A. Tabaczynski Joseph T. Mayhan. *Measurements-Based Radar Signature Modeling: An Analysis Framework*. The MIT Press, 2024.

[31] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. *Advances in neural information processing systems*, 34:852–863, 2021.

[32] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[33] Risi Kondor, Zhen Lin, and Shubhendu Trivedi. Clebsch–gordan nets: a fully fourier space spherical convolutional neural network. *Advances in Neural Information Processing Systems*, 31, 2018.

[34] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. Contrastive representation learning: A framework and review. *Ieee Access*, 8:193907–193934, 2020.

[35] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, Fujie Huang, et al. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

[36] Jongmin Lee and Minsu Cho. 3d equivariant pose regression via direct wigner-d harmonics prediction, 2024. URL `https://arxiv.org/abs/2411.00543`.

[37] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.

[38] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[39] Yi-Lun Liao and Tess Smidt. Equiformer: Equivariant graph attention transformer for 3d atomistic graphs. In *The Eleventh International Conference on Learning Representations*, 2022.

[40] Yi-Lun Liao, Brandon Wood, Abhishek Das, and Tess Smidt. Equiformerv2: Improved equivariant transformer for scaling to higher-degree representations. *arXiv preprint arXiv:2306.12059*, 2023.

[41] Kevin Lin, Lijuan Wang, and Zicheng Liu. Mesh graphormer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12939–12948, 2021.

[42] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[43] Kalinja Naffer-Chevassier, Florian De Vuyst, and Yohann Goardou. Enhanced drag force estimation in automotive design: A surrogate model leveraging limited full-order model drag data and comprehensive physical field integration. *Computation*, 12(10):207, 2024.

[44] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

[45] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[46] Othmane-Latif Ouabi, Radmila Pribić, and Sorin Olaru. Stochastic complex-valued neural networks for radar. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 1442–1446, 2021. doi: 10.23919/Eusipco47968.2020.9287425.

[47] Edouard Oyallon, Eugene Belilovsky, Sergey Zagoruyko, and Michal Valko. Compressing the input for cnns with the first-order scattering transform. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 301–316, 2018.

[48] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P Schoellig. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7512–7519. IEEE, 2021.

[49] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[50] Nathanaël Schaeffer. Efficient spherical harmonic transforms aimed at pseudospectral numerical simulations. *Geochemistry, Geophysics, Geosystems*, 14(3):751–758, 2013.

[51] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[52] Yawar Siddiqui, Antonio Alliegro, Alexey Artemov, Tatiana Tommasi, Daniele Sirigatti, Vladislav Rosov, Angela Dai, and Matthias Nießner. Meshgpt: Generating triangle meshes with decoder-only transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19615–19625, 2024.

[53] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[54] 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 preprint arXiv:1802.08219*, 2018.

[55] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Adv. in Neural Info. Proc. Sys.*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[56] Sudheendra Vijayanarasimhan, Jonathon Shlens, Rajat Monga, and Jay Yagnik. Deep networks with large output spaces. *arXiv preprint arXiv:1412.7479*, 2014.

[57] Bo Wang, Li-Lian Wang, and Ziqing Xie. Accurate calculation of spherical and vector spherical harmonic expansions via spectral element grids. *Advances in Computational Mathematics*, 44: 951–985, 2018.

[58] Dian Wang, Stephen Hart, David Surovik, Tarik Kelestemur, Haojie Huang, Haibo Zhao, Mark Yeatman, Jiuguang Wang, Robin Walters, and Robert Platt. Equivariant diffusion policy. In *8th Annual Conference on Robot Learning*, 2024. URL `https://openreview.net/forum?id=wD2kUVLT1g`.

[59] Qing Wang, Olaf Ronneberger, and Hans Burkhardt. Rotational invariance based on fourier analysis in polar and spherical coordinates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(9):1715–1722, 2009.

[60] Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=wta_8Hx2KD`.

[61] Maurice Weiler and Gabriele Cesa. General E(2)-Equivariant Steerable CNNs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. URL `https://arxiv.org/abs/1911.08251`.

[62] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. *Advances in Neural Information Processing Systems*, 31, 2018.

[63] Tim A. Wheeler, Martin Holder, Hermann Winner, and Mykel J. Kochenderfer. Deep stochastic radar models. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 47–53, 2017. doi: 10.1109/IVS.2017.7995697.

[64] Bo-Wen Zan, Zhong-Hua Han, Chen-Zhou Xu, Ming-Qi Liu, and Wen-Zheng Wang. High-dimensional aerodynamic data modeling using a machine learning method based on a convolutional neural network. *Advances in Aeronautics*, 4(1):39, December 2022. doi: 10.1186/s42774-022-00128-8.

[65] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: 4th*

International Workshop, DLMIA 2018, and 8th International Workshop, ML-CDS 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 20, 2018, Proceedings 4, pages 3–11. Springer, 2018.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The claims made in the abstract/intro are substantiated by the experimental section which demonstrates the performance of the proposed method compared to a number of baselines both for supervised learning and policy learning.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We include a limitations and discussion section in the appendix (B) alongside the discussion of the limitations of the components of the method in the main text. If accepted we plan on moving this into the main body of the work with the additional page.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: This work is focused on the applications of the proposed method and does not include theortical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide numerous high level details in the method section of the main body and then more details specific to the experiments in the experimental section. Additionally, in the appendix exact details regarding the models utilized for both our method and the baselines are given.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: While we are planning on releasing both the code and datasets in this work if accepted, they are subject to an internal review process which must be complete first which will began upon acceptance of this work (if accepted).

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide these details in the appendix for each of the experiments ran.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Our tables show the mean and standard error for a number of runs with different random seeds.

   Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We specify the CPUs/GPUs utilized for our training/testing in the Training details section of the appendix. We include the compute required for the experiments in this work and also an estimate of the overall compute used during the course of research.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and this work adheres to the relevant sections.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

Answer: [NA]

Justification: This work focused on the technical applications of a specific type of modeling and none of the application domains have broad societal impacts.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This work does not deal with either data or models that have a high risk for misuse as it is primally considered with the modeling of spherical signals via the Fourier-based methods.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The existing assets used in this work have both their original works cited and when code is used urls are provided to the repositories.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: While we are planning on releasing both the code and datasets in this work if accepted, they are subject to an internal review process which must be complete first which will began upon acceptance of this work (if accepted)

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This work does not involve either crowdsourcing or human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This work does not involve either crowdsourcing or human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: This work does not use LLMs.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Reproducibility statement

To ensure reproducibility, we have documented and made accessible all resources utilized by this work. Our source code, datasets, and visualization methods are provided for transparency. We have provided additional training and model architectural details for both G2Sphere and the various baselines in Appendix G. The techniques utilized to generate our high-quality synthetic data used in the experiments in Sec. 5.1 is detailed in Appendix J. Similarly, the details for the expert data used for policy learning experiments can be found in Appendix K alongside additional environmental details such as the sources for the environment implementations. Finally, the design decisions and implementation details of our harmonics implementation tailored to equivariant neural networks are discussed in Appendix D.

B Limitations and discussion

There are a number of limitations which future work could improve upon. First, in regards to the radar and drag domains, the output fidelity of G2Sphere is heavily dependent on the maximum harmonic frequency of the model. Unfortunately, current implementations of the spherical harmonics in equivariant network architectures result in an cubic increase in computational requirements as the harmonic frequency increases, thereby limiting the expressivity of our model. However, there do exist more efficient methods for calculating the spherical harmonics which would allow for increased maximum frequencies in future work [57, 50]. Similarly, in this work we model the radial component of the spherical coordinates by either outputting a discrete set of spheres for different radii or by including it as an implicit variable. An alternate method would be to incorporate the radial component into the Fourier basis functions allowing for efficient computation of the all spheres continuously within some defined radial bounds [59]. Finally, in this work we only explore 2D and 3D positional control policies and have omitted full 6DoF pose control. The model can be extended to 6DoF by applying inverse FT over $\mathrm{SO}(3)$ instead of S^2.

C Group theory

Equivariance A function is equivariant if it respects the symmetries of its input and output spaces. Specifically, a function $F\colon X \to Y$ is *equivariant* with respect to a symmetry group G, if it commutes with all transformations $g \in G, F(\rho_x(g)x) = \rho_y(g)F(x)$, where ρ_x and ρ_y are the *representations* of the group G that define how the group element $g \in G$ acts on $x \in X$ and $y \in Y$ respectively. An equivariant function is a mathematical way of expressing that F is symmetric with respect to G; if we evaluate F for various transformed versions of the same input, we should obtain transformed versions of the same output.

SO(3)-Equivariant Neural Networks For 3D physical systems, since the orientation of the coordinate frame is arbitrary, many task functions f should be $\mathrm{SO}(3)$-equivariant. For neural networks to incorporate this sort of geometric reasoning, it is necessary to parameterize signals $f\colon \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} = S^2$ or $\mathcal{X} = \mathrm{SO}(3)$ in a way that is both computationally efficient and easy to apply rotations to. Cohen et al. [15] provide an effective solution using the truncated basis of spherical harmonics Y_m^l for signals defined over S^2 and Wigner D-matrix coefficients D_{mn}^l for signals over $\mathrm{SO}(3)$. Writing $f\colon \mathrm{SO}(3) \to \mathbb{R}$ in terms of the D_{mn}^l and then truncating to a given frequency $l \leq L$ gives the approximation $f(g) \approx \sum_{l=0}^{L} \sum_{m=0}^{2l+1} \sum_{n=0}^{2l+1} c_{mn}^l D_{mn}^l(g)$. The $\mathrm{SO}(3)$ group convolution can be efficiently computed in the Fourier domain in terms of the coefficients $\{c_{mn}^l\}$ by convolution theorem. See Cohen et al. [15] for additional details on the $\mathrm{SO}(3)$ group convolution.

Circular and spherical harmonics The *Circular Harmonics (CH)* describe functions on the 1-sphere (i.e. the circle S^1). They are solutions to Laplace's equation over S^1 defined $\psi_l(\theta) = e^{il\theta}$, where θ is the angular coordinate [59]. In terms of group theory, the circular harmonics define irreducible representations for the 2D rotation group $\mathrm{SO}(2)$. Therefore, they are useful for constructing $\mathrm{SO}(2)$-equivariant neural networks by representing circular signals as a combination of fixed CH basis functions and learnable CH coefficients [61].

The *Spherical Harmonics (SH)* extend the CH to describe functions on the surface of the 2-sphere S^2. Similar to the CH, they are orthonormal solutions of Laplace's equation on S^2, defined:

$$Y_m^l(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi} \frac{(l-m)!}{(l+m)!}} P_m^l(\cos\theta) e^{im\phi},$$

where θ is the polar angle, ϕ is the azimuth angle, l is the degree, m is the order and P_m^l is the associated Legendre function [1]. The SH describe irreducible representations of the group $\mathrm{SO}(3)$ of 3D rotations. They are useful for creating $\mathrm{SO}(3)$-equivariant features for $\mathrm{SO}(3)$-equivariant deep learning [23].

D Harmonics implementations

In this section, we describe the details of our PyTorch harmonics packages which implements the Polar and Spherical Harmonics used in this work. While there are a number of existing spherical harmonics implementations [7], they are tailored to the tasks that they solve, e.g. PDEs, and therefore do not integrate well with the equivariant neural networks used in this work (e.g. `e3nn` or `escnn`). Additionally, there are, to the best of our knowledge, no PyTorch implementations of the Polar Harmonics (detailed in Appendix E). To address these shortcomings, we developed our own differentiable implementations of the Polar and Spherical harmonics in PyTorch which we utilize throughout this work. The design goals of this package were two fold: (1) allow for the efficient integration of the harmonics with the output of `e3nn` or `escnn` and (2) enable the calculation of the harmonics on both a pre-defined grid of coordinates and at specific coordinates.

There are two considerations to make when tailoring our package to fit together with `e3nn` and `escnn`. First, these equivariant neural networks model the Fourier coefficients as the irreducible representations of the symmetry group and, therefor, our implementation needs to use the same representations. We note that this restriction makes a number of previous implementations such as Bonev et al. [7] a poor fit as they use different derivations of the harmonics with different Fourier coefficient specifications. Secondly, because these equivariant models typically operate in real space, we need the harmonics to be in real space as opposed to complex. For example, we use the $\sin$ and $\cos$ form of the circular harmonics (Appendix E) to match the dimensionality of the $SO(2)$ irreducible representations.

The second goal of our package is enable efficient batch computation of the harmonics over many values. We require the ability to both compute the harmonics at specific coordinates (commonly used during training to match the partial views u in our dataset) *and* to pre-compute a grid of harmonics values at pre-defined coordinates (typically used at inference time to decrease computation time). In order to achieve this, when instantiating the harmonics, a grid of harmonics values are pre-computed at the desired resolution and a number of intermediate variables are saved during this process. These intermediate variables are specific to the type of harmonics but, in general, they are components of the harmonics independent to the coordinates such as the normalization constants and Fourier mode components. This pre-computation allows for the efficient evaluation of the harmonics and reduces training times.

E Polar Fourier transform

In this section we derive the Polar Harmonics Basis functions used in our 2D policy learning experiments by performing Fourier analysis on the Polar coordinates. We would ideally like for these functions to be decomposed into radial and angular components such that we can view this decomposition as an extension of the normal Fourier transform. We can fulfill this requirement by requiring the basis functions to take the separation-of-variables form:

$$f(\rho, \phi) = \mathrm{P}(\rho)\Phi(\phi). \tag{1}$$

E.1 Basis functions

The angular component of the basis function is simply:

$$\Phi_l(\phi) = \frac{1}{\sqrt{2\pi}} e^{im\phi}, \tag{2}$$

where l is an integer. However, this definition resides in the complex domain, and when using equivariant neural networks, it is advantageous to be in the domain of real numbers. Therefore, we will instead use the sine-cosine form:

$$\Phi_l(\phi) = \frac{1}{\sqrt{2\pi}}\big(\cos(l\phi) + \sin(l\phi)\big), \tag{3}$$

where $l \geq 0$. The associated transform in angular coordinates is simply the normal 1D Fourier Transform.

We will rely on Bessel functions to match the functions of arbitrary spatial frequency required by the radial component of the Polar basis function. Using the Bessel function as a basis gives rise to the l-th order Fourier-Bessel series:

$$\mathrm{P}_{lm}(\rho) = \frac{1}{\sqrt{N_{lm}}} J_l(k_{lm}\rho), \tag{4}$$

where N_{lm} is a normalization constant and k_{lm} is the k^{th} zero of J_1.

E.2 Polar Fourier transform

By combining these radial and angular components we can form the complete basis function for the polar Fourier transform:

$$\Psi_{lm}(\rho, \phi) = \mathrm{P}_{lm}(\rho)\Psi_l(\psi), \tag{5}$$

where P_{lm} is defined by Eq. 4 and Ψ_l by Eq. 2. As Ψ_{lm} forms an orthonormal basis on the region $\rho < a$, we refer to it as the *Polar Harmonics*. A function defined over polar coordinates, $f(\rho, \phi)$, can be expanded using the polar Fourier transform with the Polar basis function, Ψ_{nm}, and Polar coefficients, P_{lm}:

$$f(\rho, \phi) = \int_0^{\inf} \int_0^{\inf} P_{lm}\Psi_{lm}(\rho, \psi) m dm, \tag{6}$$

where

$$P_{lm} = \int_0^{\inf} \int_0^{2\pi} f(\rho, \phi)\Psi^*_{lm}(\rho, \phi)\rho d\rho d\phi. \tag{7}$$

While this infinite transform is of theoretical interest, in practice we instead use the transform defined on a finite region, e.g. the unit-circle. That is, a function $f(\rho, \phi)$, where $\rho \leq a$ can be defined as:

$$f(\rho, \phi) = \sum_{m=1}^{\inf} \int_{l=0}^{\inf} P_{lm}\Psi_{lm}(\rho, \psi), \tag{8}$$

where

$$P_{lm} = \int_0^{a} \int_0^{2\pi} f(\rho, \phi)\Psi^*_{lm}(\rho, \phi)\rho d\rho d\phi. \tag{9}$$

F Spherical Fourier Transform

Functions defined on the 2-sphere S^2 are commonly expanded in a basis of spherical harmonics Y^l_m, yielding coefficients that compactly capture angular variation up to degree l_{max}. They are orthonormal solutions of Laplace's equation on S^2, defined:

$$Y^l_m(\theta, \phi) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} P^l_m(\cos\theta) e^{im\phi},$$

where θ is the polar angle, ϕ is the azimuth angle, l is the degree, m is the order and P^l_m is the associated Legendre function [1]. The SH describe irreducible representations of the group $\mathrm{SO}(3)$ of 3D rotations. They are useful for creating $\mathrm{SO}(3)$-equivariant features for $\mathrm{SO}(3)$-equivariant deep learning [23] and underlay many spherical convolution methods, where learnable filters act on these coefficients to extract multi-scale features from spherical signals.

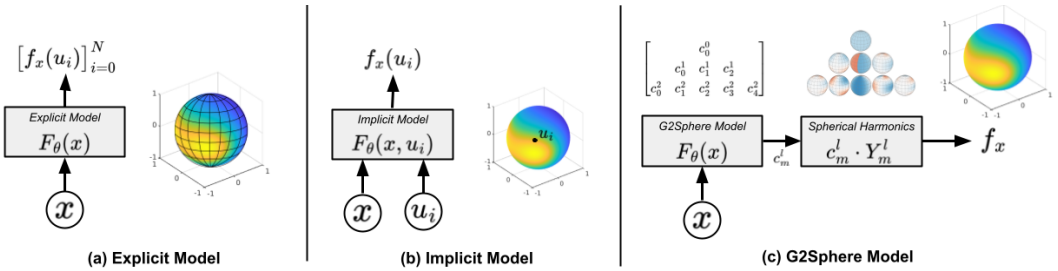


Figure 5: **Modeling Spherical Functions.** (a) The explicit model outputs a fixed grid of predictions to approximate the continuous function f_x. (b) The implicit model learns f_x by conditioning on both input x and coordinate point u. (c) G2Sphere decomposes f_x into a set of learned Fourier coefficients c_l^m and spherical harmonics Y_l^m. This enables prediction of f_x at any coordinate point u.

G Training details

In this section we provide additional details about the model architectures and training process for the experiments in Sec. 5. Broadly speaking our models fall into three categories: explicit models (Fig. 5a), implicit models (Fig 5b), and our G2Sphere model (Fig 5c). In the radar prediction domain, the baselines are explicit models using a grid of regularly spaced spherical viewing angles. As mentioned in Section 5.1, the drag prediction domain is limited by the data and, therefor, implicit models are used for the baseline methods. The same base architecture is used when modifying the models to be explicit or implicit with a few small changes to modify the input/ output spaces (see Fig. 5 for additional details). Finally, in the policy learning experiments, the IBC baseline uses implicit models whereas the diffusion policy uses an explicit model. All models were trained until convergence measured by loss in the supervised learning domains and policy performance in the policy learning domains. All of our experiments are run on a high-performance cluster using either one or two Nvidia Xeon-g6-volta GPUs with 32GB of memory. Each node has a Intel Xeon Gold 6248 2.4 GHz CPU with 128GB of RAM and 1TB of local storage. Each experimental run required approximately 8 GPU-hours, and the total compute for all reported experiments was around 300 GPU-hours. Including preliminary experiments and ablations not included in the final paper, the full research project used approximately 500 GPU-hours.

G.1 Mesh-to-Sphere

G.1.1 Transformer

The Transformer architecture is composed of a mesh face embedding layer, transformer encoder, and an MLP decoder. The mesh face embedding layer takes individual mesh faces as an input and uses geometric features like vertex position and normal direction to calculate an embedded representation of the face. The application of this layer to the mesh generates a sequence of embedded faces; a learned classification embedding is prepended to this sequence. A standard transformer encoder as introduced in Vaswani et al. [55] is then applied to the embedded sequence to transfer global information about the mesh faces to the classification embedding; after this operation, the MLP decoder is used to convert the classification embedding into the radar response. We train using the Adam optimizer [32] with the best learning rate and its decay were chosen to be $5e^{-4}$ and $1e^{-5}$ respectively. We use a batch size of 4.

G.1.2 G2Sphere

For the mesh-to-sphere domain, the G2Sphere model uses Equiformer v2 to encode the object mesh into Fourier space. We use the equiformer architecture from Liao et al. [40] with 4 layers where each layer a feature dimension of 128, a l_{max} of 5, and 4 attention heads. We then use a global mean pooling head to compress the node features returned by Equiformer into a single latent representation of size 128. The decoder is a 5 layer Spherical CNN [15] with the feature dimensions $[128, 64, 32, 16, K]$, where K is the number of output channels in the spherical signal, and l_{max} dimensions $[5, 10, 15, 20, 40]$. We use the ReLU [] activation in our $\mathrm{SO}(3)$-activation layers in the

Spherical CNN. We train using the Adam optimizer [32] with the best learning rate and its decay were chosen to be 1^{-4} and 0.95 respectively. We use a batch size of 4.

G.1.3 Equiformer

The Equiformer baseline has the same encoder architecture as G2Sphere but a slightly different decoder architecture. The Spherical CNN decoder in Equiformer, has 6 layers where the feature dimensions are $[128, 64, 32, 16, K]$, where K is the number of output channels in the spherical signal, and l_{max} dimensions $[5, 5, 5, 5, 5]$. We use the ReLU [] activation in our $\mathrm{SO}(3)$-activation layers in the Spherical CNN. At the end of the Spherical CNN, a final spherical convolution converts the latent representation into the explicit grid of output values. We use the ReLU [] activation in our $\mathrm{SO}(3)$-activation layers in the Spherical CNN. We train using the Adam optimizer [32] with the best learning rate and its decay were chosen to be 1^{-4} and 0.95 respectively. We use a batch size of 4.

G.2 Policy learning

G.2.1 G2Sphere

In the 2D and 3D policy learning domains, our observations are a set of 2D/3D keypoints which describe the objects in the scene. Therefore, we use much simpler models than in the mesh-to-sphere domain. Specifically, we use $\mathrm{SO}(2)$ and $\mathrm{SO}(3)$-equivariant MLPs as both our encdoers and decoders. In the *PushT* domain, we use a 4-layer $SO(2)$-equivaraint MLP for our encoder with a 512-dimensional latent representation and a 4-layer, 512-feature dimension, $SO(2)$-equivariant spherical CNN with as the decoder. This $SO(2)$ spherical CNN is essentially a series of spherical convolutions on the 1-sphere as opposed to the 2-sphere. Both encoder and decoder use a L_{max} of 3. Similarly in the *PyBullet Drones* domain, we use a 4-layer $SO(3)$-equivaraint MLP for our encoder with a 256-dimensional latent representation and a 4-layer, 256-feature dimension, spherical CNN decoder. Both encoder and decoder use a L_{max} of 5. Both models use a dropout [53] of 0.1 while training. We train using the Adam optimizer [32] with the best learning rate and its decay were chosen to be 1^{-4} and 0.95 respectively. We use a batch size of 256 and train our contrastive loss using 256 negative samples.

G.2.2 IBC

We use two IBC models in our policy learning experiments: a non-equivariant version which uses standard MLPs and a equivariant version which uses $\mathrm{SO}(2)$ or $\mathrm{SO}(3)$-equivariant MLPs. The non-equivaraint IBC is comprised of a 4 layer MLP encoder and a 4 layer MLP decoder both with 1024 feature dimensions. The implicit actions are appended onto the latent representation after it has been encoded. The equivariant version of IBC, has the same structure as the non-equivariant model but uses a feature dimension of 512 and a L_{max} of 3 and 5 (for the 2D and 3D domains respectively). We train using the Adam optimizer [32] with the best learning rate and its decay were chosen to be 1^{-4} and 0.95 respectively. We use a batch size of 256. The contrastive loss is generated using 256 negative samples following Florence et al. [22]. We use a dropout [53] of 0.1 while training. At inference time, we perform 3 iterations of Derivative-Free Optimization [22] using 4096 samples to select the best action.

G.2.3 Diffusion

We use the implementations from [13] [MIT License] for our Diffusion models, specifically, we use the transformer-based architectures. The transformer has 8 layers with 4 heads where each token is embedded as a 256 feature vector. We use the Square Cosine Schedule proposed in iDDPM [44] which is cited as the best performing noise scheduler in Chi et al. [13]. We train using the Adam optimizer [32] with the best learning rate and its decay were chosen to be 1^{-4} and 0.95 respectively. We use a batch size of 256. We use 100 denoising diffusion iterations for both training and inference. At inference time the model predicts the next 16 actions of which 8 are executed.

Table 4: L_{dec} **Ablation.** Mean square error of G2Sphere using increasing maximum output harmonic frequencies. As the output frequency of the decoder increases the performance of the G2Sphere model improves. Performance is averaged across 3 training seeds.

Asym	$L_{dec} = 5$	$L_{dec} = 10$	$L_{dec} = 25$	$L_{dec} = 40$
G2S	0.383	0.218	0.148	**0.123**

(a) Ground Truth (b) $L_{dec} = 5$ (c) $L_{dec} = 10$ (d) $L_{dec} = 25$ (e) $L_{dec} = 40$

Figure 6: **Effect of L_{dec} on Fidelity.** As the maximum frequency of the decoder increases so does the accuracy of the G2Sphere prediction. At $L_{dec} = 5$ we see that most of the features of the signal are lost and we are only capable of modelling the general high/low signals. However, as we increase to $L_{dec} > 5$ we can see the features getting sharper as the output frequency increases.

H Maximum frequency ablations

A unique capability of G2Sphere is its ability to leverage the maximum harmonic frequency L to control bias of the model. A higher L captures more fine details and higher-frequency components of the underlying function which can lead to a more complex and accurate approximation. Therefore when trying to predict the high-frequency radar response function, we find that the higher the L the more accurate our predictions become (Table 4, Fig. 6). However, we note that this is only possible due to the large scale radar datasets which have responses over the entire sphere for each mesh. In contrast, in domains where we lack these dense outputs, such as the drag and policy learning domains, we find a lower L can combat overfitting by providing a smoother, lower-detailed approximation of the underlying function (Fig. 7).

I Multimodality

Learning from human demonstrations presents a significant challenge to BC due to the challenge of modeling the multi-modal distributions common in human actions [22, 51, 27, 13]. One advantage of G2Sphere is the ability to control the maximum multimodality of the policy via the maximum frequency. At a high level, the maximum frequency describes the amount of energy in the basis function associated with that maximum frequency. From a practical point of view, this means that the higher the frequency the more complicated a function you are able to model. G2Sphere takes advantage of this by setting the maximum angular frequency to at least the amount of multimodality

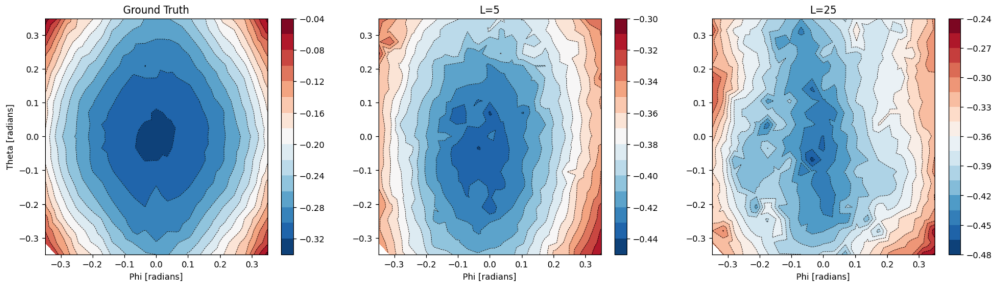
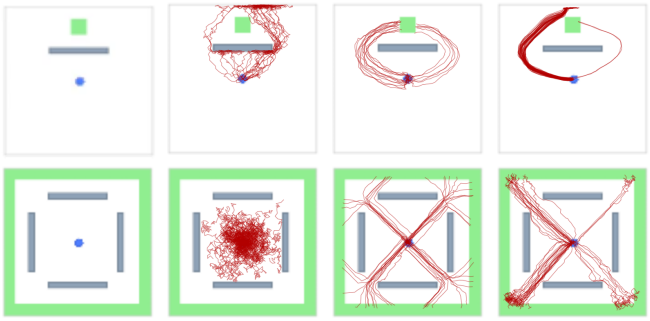
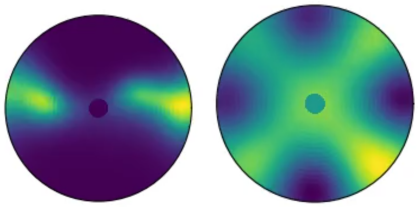


Figure 7: **Effect of L_{dec} on Generalization.** As the maximum frequency of the decoder increases, G2Sphere starts to overfit to the sparse data samples in the Drag domain. As a result, the G2Sphere model with a lower output frequency generalizes better to new data.



(a) **N-Paths** (b) **IBC** (c) **G2S** (d) **Diff**

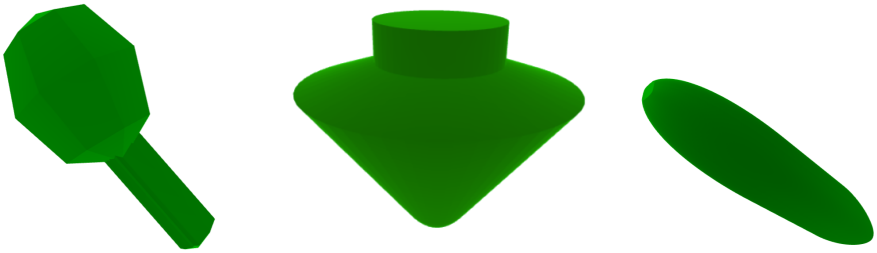
Figure 8: **Multimodal Behavior.** From the start state (blue) there are two paths (top row) and four paths (bottom row) to the goal state (green). When using high enough angular frequency ($L = 2$, $L = 4$), G2Sphere learns all paths and commits to a path at the start of each rollout. Diffusion Policy exhibits a similar behavior but shows some bias towards certain paths whereas IBC fails to learn due to an equal distribution of expert path data.



(a) **N=2, L=2** (b) **N=4, L=4**

Figure 9: **Energy Landscape.** We visualize the energy landscapes generated with the G2Sphere on the N-Paths domain with $N = 2$ and $N = 4$, respectively.

present in the task. We illustrate this behavior using the simple N-Paths task where the agent must navigate to the goal (green) using one of the N available paths. Fig. 9 shows that by setting the maximum frequency to the multimodality present in 2-Paths and 4-Paths, the energy landscape generated by G2Sphere is able to model each of the multimodal trajectories. Additionally, Fig. 8 shows example trajectories generated by G2Sphere, E-IBC, and Diffusion. We note that G2Sphere is the best at matching the multimodality present in the task and is equally likely to take any of the N-paths, while diffusion shows bias towards certain paths and IBC fails to commit to any of the paths.



(a) Asym Mesh (b) Frusta Mesh (c) Pods Mesh

Figure 10: **Object Meshes.** Example object meshes used in the mesh-to-sphere experiments. The Asym and Frusta meshes are used in the radar prediction domain and the pods meshes are used in the aerodynamics prediction domain.

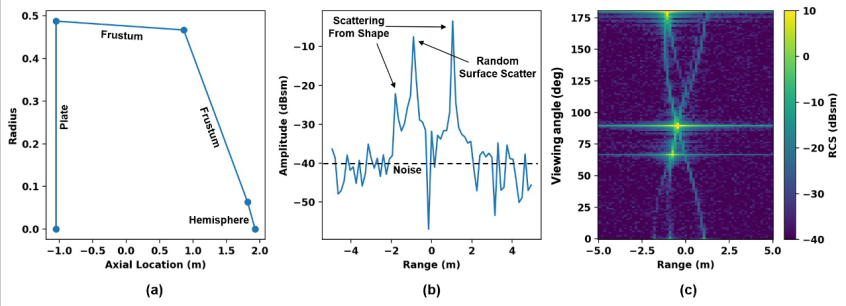


Figure 11: **Radar Simulator.** (a) Cross section of random 3D object mesh. (b) The range profile for 20° viewing angle. (c) The corresponding static radar pattern.

J Dataset generation

J.1 First-Principle RF model

Due to the scarcity of available real-world and simulated radar data for training radar models, we simulate our own benchmark dataset. To generate ground truth data, we use the physical optics approximation method [4], which provides a linear approximation of the more general and highly non-linear scattering formulation for electromagnetic waves. A simple operator that describes physical optics response across the illuminated section of an object for perfectly reflecting material as,

$$F(x,k) = \frac{ik}{2\pi} \int_{R^3} e^{-i2k\langle x,y \rangle} dy,$$

where the incident wave number is $k = 2\pi/\lambda$, λ is the wavelength, and the observation unit vector is,

$$x = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta),$$

for $\theta \in [0,\pi]$ and $\phi \in [0,2\pi]$. We note that the approximation is valid only in high frequency regions such that $k << 2\pi/D$, where D is the length of the longest side of the object.

As we are particularly interested in far-field sensing, the physical optics approximation is useful as a fairly accurate and flexible simulation tool for training data generation. The simulation input is a parameterized mesh object (an example cross section shown in Fig. 11(a)). The simulation calculates the the physical optics response for given observation line-of-sight and frequency and the total radar response of the object is equal to the sum of the individual triangle responses that are visible to the radar. Simulations for this work required generating the response across a linear set of frequencies to emulate a Linear Frequency Modulated (LFM) waveform, where the center frequency is $3e9$Hz and bandwidth is $4e8$Hz using circular polarized waves with orientation RL.

The Radar Cross Sec. (RCS) for each triangle is calculated using legacy software [10]. The simulation produces a radar observation, $r \in \mathbb{R}^{N_r}$ (Fig. 11(b)) for a given viewing angle. The observation is the normalized magnitude of the range-profile as described in Sec. III of Chance et al. [12]. All the normalized range profiles are then stacked across varying radar viewing angles to generate what is referenced as a radar static pattern (Fig. 11(c)). Note that fixing the radar line-of-sight and rotating the object would generate the same radar static response.

We utilize two different mesh datasets for our experiments. The first is the *Frusta* dataset. Each object in this dataset is defined by a series of stacked frusta objects, in which adjacent frusta share the same radius to ensure a continuous object. To generate a diverse set of objects, we vary both the number of frusta components and the radial parameters for each frusta. To introduce additional object and radar response diversity, we also vary the ends of the frusta object to either be flat planes, half spheres, or one of each.

While the *Frusta* dataset provides objects with complex forms, it is also rotational symmetric along the roll axis. To address this issue, we also generate the *Asym*. This dataset consists of three different base shapes; cubes, cylinders, and spheres. Each object is randomly scaled across it's independent dimensions; to ensure diversity, no random scaling is repeated within object class. After the scaling, the mesh is duplicated and subdivided to decrease the relative size of mesh faces. A group of adjacent

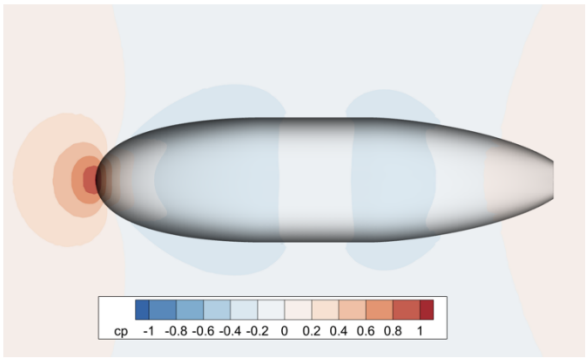


Figure 12: **Aerodynamics Simulator.** Ground truth drag coefficients generated by the numerical CDF simulator.

faces is selected at random from the overall set of mesh faces on the subdivided copy; these faces are then extruded by a random amount in the average normal direction of the selection. This extruded protrusion is then added to the original mesh using the union operation, resulting in an object without any rotational symmetries.

Using these methods, we generate a dataset of X objects for the *Frusta* dataset, and 30,000 objects for the *Asym* dataset, evenly distributed across the underlying classes. Each dataset is divided 90/10 between train/validation and test sets. Since the meshes resulting from the Frusta generation method are large relative to the capacity of neural network models (maximum size of 4500 faces), frusta meshes are decimated to 50% of their face count using quadratic decimation for training and testing.

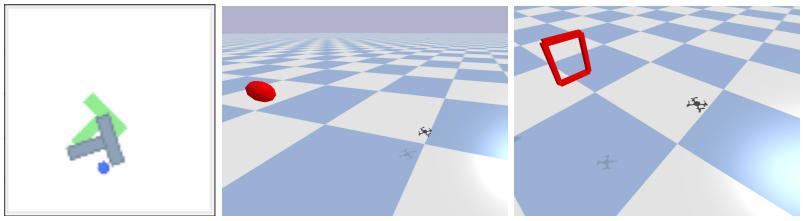
J.2 Aerodynamic simulation and data

The aerodynamic dataset consists of "pod-like" shapes defined parametrically by varying shape parameters such as: length, diameter, bluntness, and cross-section asymmetry using Latin Hypercube Sampling. Flight conditions including altitude, Mach number and Reynolds number were also selected using Latin Hypercube Sampling, where altitude ranges from 0 to 50 kilofeet, Mach number ranges from 0.05 to 0.5, and Reynolds number ranges from 10^6 to 5×10^8. For each shape permutation a high-quality computational grid is algorithmically generated. The minimum cell size is defined to provide over 100 cells across the length of each shape, and 50-100 volume cells to resolve the boundary layer down to the viscous sublayer, i.e. $y^+ = 1$, where y^+1 is the wall coordinate commonly used in defining the law of the wall. Generation of drag force data is based on a Computational Fluid Dynamics (CFD) simulator for which we solve the Reynolds-Averaged Navier-Stokes (RANS) mass, momentum and energy equations through the NASA FUN3D simulation framework. Turbulence is modeled using the Spalart-Allmaras model with a freestream turbulence intensity of 3 percent. The van Albada flux limiter with the low-diffusion flux splitting "LDFSS" flux construction method. Simulations are run until a relative residual tolerance of 10^{-5} is met for the mass, momentum, energy, and turbulence governing equations.

The Pods dataset contains $10,000$ samples of (mesh, flight conditions, drag) tuples, each with a single drag coefficient for a specific angle $\theta, \phi \in [-20, 20]$, which defines the orientation of the incoming flow relative to the centerline of the Pod shape.

K Policy learning environments

PushT We generate 100 expert demonstrations for both the fixed and randomized PushT tasks. All demonstrations were generated by a single human operator familiar with the task. Each episode terminates either after the task is completed or the maximum number of steps (500) is reached. We use the PushT implementation found in Chi et al. [13] which can be found here: `https://github.com/real-stanford/diffusion_policy` [MIT License].



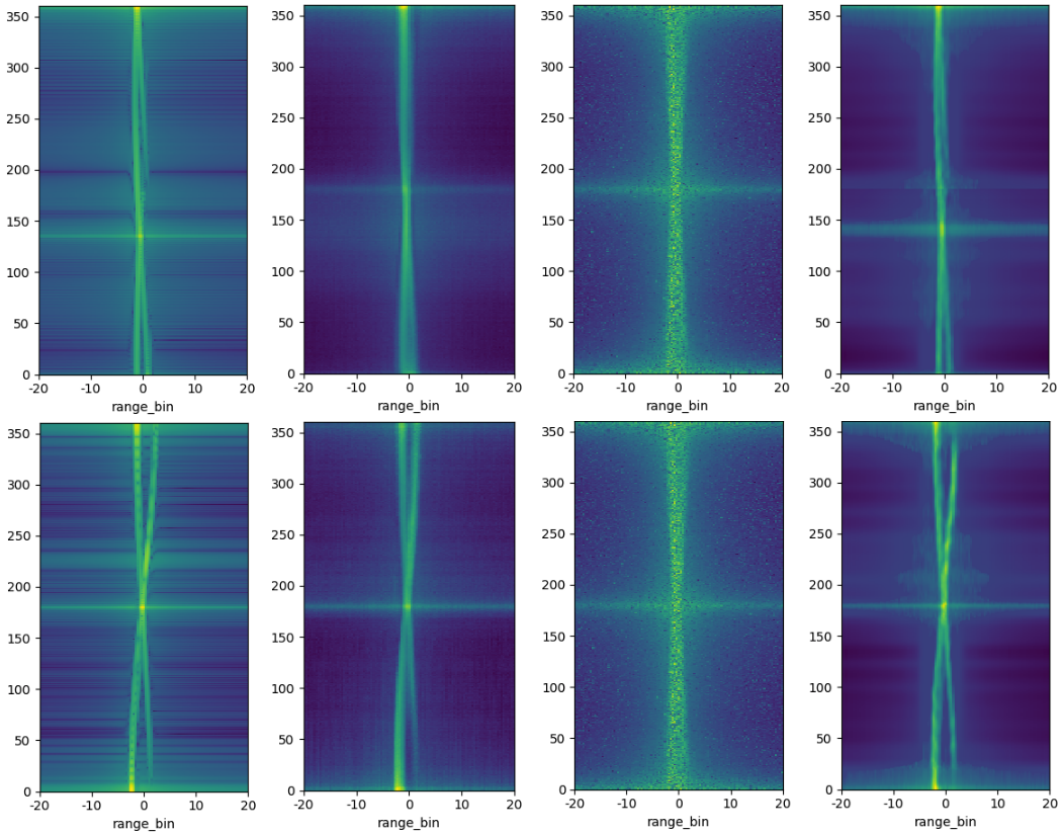
(a) **PushT.** (b) **FlyToTarget.** (c) **FlyThroughGate.**

Figure 13: **Policy Learning Domains.** The tasks from our policy learning benchmarks. The fixed-goal variant of PushT is shown in (a). The target position to fly to is shown as a red ball in (b) and the gate to fly through is shown in red in (c).

PyBullet Drones We generate 50 expert demonstrations for both the *FlyToTarget* and *FlyThroughGate* tasks. All demonstrations were generated by an expert policy which has access to the underlying simulation state. We add a small amount of uniformly sampled noise, $\epsilon \sim U(0, 1)$, to each action from the expert policy. Each episode terminates either after the task is completed or the maximum number of steps (300) is reached. We use the PyBullet Drones implementation found in Panerati et al. [48] which can be found here: `https://utiasdsl.github.io/gym-pybullet-drones/` [MIT License].

L Example radar response outputs

In Fig. 14 we highlight representative examples generated by the ground truth simulator, transformer, equiformer, and G2Sphere on the Frusta dataset. We note that although G2Sphere still produces the best predictions, the difference between G2Sphere and the transformer baseline is much less pronounced on this dataset. This aligns with the result in Table 1 which shows that the performance of the transformer baseline and G2Sphere to be similar.



(a) Ground Truth (b) Transformer (c) Equiformer (d) G2Sphere

Figure 14: **Frusta Radar Predictions.** Radar response predictions on the Frusta dataset. Due to the roll symmetry in the Frusta dataset we plot the spherical outputs as static patterns. G2Sphere is best able to capture both the overall structure of the response and also the speculars (bright horizontal lines).