PAPER • OPEN ACCESS

Geometric neural operators (gnps) for data-driven deep learning in non-euclidean settings

MACHINE LEARNING Science and Technology

PAPER

OPEN ACCESS

RECEIVED
11 July 2024

REVISED
1 October 2024

ACCEPTED FOR PUBLICATION
21 October 2024

PUBLISHED
6 November 2024

Geometric neural operators (gnps) for data-driven deep learning in non-euclidean settings

B Quackenbush[1] and P J Atzberger[1,2,*]

[1] Department of Mathematics, University of California Santa Barbara (UCSB), Santa Barbara, CA, United States of America
[2] Department of Mathematics, Department of Mechanical Engineering, University of California Santa Barbara (UCSB), Santa Barbara, CA, United States of America
[*] Author to whom any correspondence should be addressed.

E-mail: atzberg@gmail.com

Keywords: neural operator, deep learning, geometric methods, partial differential equations, inverse problems, data-driven methods

Abstract

We introduce Geometric Neural Operators (GNPs) for data-driven deep learning of geometric features for tasks in non-euclidean settings. We present a formulation for accounting for geometric contributions along with practical neural network architectures and factorizations for training. We then demonstrate how GNPs can be used (i) to estimate geometric properties, such as the metric and curvatures of surfaces, (ii) to approximate solutions of geometric partial differential equations on manifolds, and (iii) to solve Bayesian inverse problems for identifying manifold shapes. These results show a few ways GNPs can be used for incorporating the roles of geometry in the data-driven learning of operators.

1. Introduction

Many data-driven modeling and inference tasks require learning operations on functions [1–7]. Problems involving mappings between function spaces include learning solution operators for partial differential equations (PDEs) and integral operators [8–10], estimators for inverse problems [3, 11, 12], and data assimilation [3, 13]. For many of these tasks, there are also significant geometric and topological structures [1, 2, 14, 15]. Sources of geometric contributions can arise both directly from the problem formulation [1, 5, 16, 17] or from more abstract considerations [4, 10, 15, 18]. For example, PDEs on manifolds or with domains having complicated shapes [5, 19]. More abstract sources of geometric structure also can arise, such as the subset of solutions of parametric PDEs arising from smooth parameterizations [4, 20–23] or from qualitative analysis of dynamical systems [24, 25]. Related geometric problems also arise in many inference settings, learning tasks, and numerical methods [14, 15, 26, 27].This includes approaches for handling point-cloud representations in shape classification [16], or in developing PDE solvers on manifolds [4].Deep neural networks hold potential for these problems by providing new approaches for non-linear approximations, learning representations for analytically unknown operations through training, providing accelerations of frequent operations, or discovering geometric structures within problems [1, 2, 4, 5, 26, 28, 29].

We introduce a class of deep neural networks for learning operators leveraging geometry referred to as ***Geometric Neural Operators (GNPs)***. The GNPs introduce capabilities for incorporating geometric contributions as features and as part of the operations performed on functions. This allows for handling functions and operations on arbitrary shaped domains, curved surfaces and other manifolds. This includes capturing non-linear and geometric contributions arising in computational geometry tasks, geometric PDEs, and shape reconstruction inverse problems.

Related work has been done on function operators and parameterized PDEs, but primarily on euclidean domains [6, 30–33]. Many of the methods also are based on using linear approaches, such proper orthogonal decomposition [34, 35] or dynamic mode decomposition [36–38]. More recently, neural networks have been used for developing non-linear approximation approaches. This includes the early work in [6], and more recent work on Neural Operators [30, 31] and incorporating geometry in [29]. A few more specialized realizations of this approach are the Fourier neural operator (FNO) [29, 32, 39], Deep-O-Nets [30], and graph neural operators [40]. GNPs handle the geometric contributions in addition to function inputs based on network architectures building on Neural Operators [31, 40].

We organize the paper as follows. We discuss the formulation of GNPs in section 2. We then develop methods for training GNPs for learning geometric quantities from point cloud representations of manifolds in section 3. We show the GNPs can be used to approximate Laplace–Beltrami (LB) operators and to learn solutions to Laplace–Beltrami–Poisson (LB-P) PDEs in section 4. We then show how GNPs can be used to perform inference in Bayesian Inverse problems for learning manifold shapes in section 5. Our results show how GNPs can be used for diverse learning tasks where significant contributions arise in operations from the geometry.

2. Geometric neural operators

For learning general non-linear mappings between infinite dimensional function spaces on manifolds that incorporate geometric contributions, we build on the neural operator framework [6, 31]. In contrast to more conventional neural networks which map between finite dimensional vector spaces, we use approaches that learn representations for operators that are not strictly tied to the underlying discretizations used for the input and output functions.

We consider geometric operators of the form $\mathcal{G}[w,\Phi] \to u$. This operator takes as input a function $w(\cdot)$, where $w : \mathbb{R}^{d_i} \to \mathbb{R}^{d_w}$ with $w \in \mathcal{W}$ for some function space $\mathcal{W}$, and a geometric description Φ with $\Phi : \mathbb{R}^{d_i} \to \mathbb{R}^{d_s}$, $\Phi \in \mathcal{S}$, and gives as output a function $u(\cdot)$ with $u : \mathbb{R}^{d_i} \to \mathbb{R}^{d_u}$, $u \in \mathcal{U}$. The operator can be expressed as a mapping $\mathcal{G} : \mathcal{W} \times \mathcal{S} \to \mathcal{U}$. This approach allows for flexibility in the formulation of the geometry representation Φ , and its influence on the output u. If the geometry is known beforehand, one could provide information using parameterizations. In this case, one could leverage one of the most common approaches using coordinate charts with $\Phi : \mathbb{R}^{d_i+1} \to \mathbb{R}^{d_g}$ where the extra component gives the chart index I. In the case of a surface, we would have $d_i = 2$ and $w(\mathbf{z}, I)$ and $\Phi(\mathbf{z}, I) \in \mathbb{R}^3$ with $\mathbf{z} \in \mathbb{R}^2$. As another approach, the geometry also could be described as the level set of some functions g_i, $\mathcal{M} = \{\mathbf{x} \mid g_i(\mathbf{x}) = 0,\ i = 1,\ldots,k\}$ embedded in $\mathbb{R}^{d_g}$. In the implicit level set case, $w = w(\mathbf{x})$ has inputs $\mathbf{x} \in \mathcal{M}$ and we could use $\Phi(\mathbf{x}) = \mathbf{x}$. In the special case when we already know the geometric quantities and contributions in advance, such as an operator that only depends on the local principle curvatures κ_1, κ_2, we can simplify learning by letting $\Phi(\cdot) = [\kappa_1(\cdot), \kappa_2(\cdot)]$. These cases illustrate within a common framework a few different ways to explicitly and implicitly incorporate the geometric contributions when learning operators. Here, we will primarily use spatial data given in the form of point clouds $\{\mathbf{x}_i\}_{i=1}^N \subset \mathbb{R}^3$ and use the geometric description $\Phi(\mathbf{x}) = \mathbf{x}$. In the notation, we denote the combination of input function and geometric description by $a(\cdot) = [w(\cdot), \Phi(\cdot)]$ with $a :\ \mathbb{R}^{d_i} \to \mathbb{R}^{d_a}$, $a \in \mathcal{A} = \mathcal{W} \times \mathcal{S}$.

We approximate the geometric operators $\mathcal{G} : \mathcal{A} \to \mathcal{U}$ by developing methods for learning a neural operator $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}$ with parameters θ. The $\mathcal{G}_\theta$ approximation consists of the following three learnable components (i) performing a lifting procedure $\mathcal{P}$ for $a \in \mathbb{R}^{d_a}$ to a higher dimensional set of feature functions $v_0 \in \mathbb{R}^{d_v}$ with $d_v \geqslant d_a$, (ii) performing compositions of layers consisting of a local linear operator W, integral operator $\mathcal{K}$, bias function $b(\cdot)$, and non-linear activation $\sigma(\cdot)$, to obtain $v_{i+1} = \sigma\left(Wv_i + \mathcal{K}[v_i] + b\right)$, and (iii) performing projection $\mathcal{Q}$ to a $\mathbb{R}^{d_u}$-valued function, see figure 1. The trainable components include the lifting procedure $\mathcal{P}$, kernel k, bias b function, and local operator W in the operator layers, and the projection $\mathcal{Q}$. We collect all of these parameters into θ. This gives a neural operator with T layers of the general form

$$\mathcal{G}_\theta^{(T)} = \mathcal{Q} \circ \sigma_T\left(W_T + \mathcal{K}_T + b_T\right) \circ \cdots \circ \sigma_0\left(W_0 + \mathcal{K}_0 + b_0\right) \circ \mathcal{P}. \tag{1}$$

The activation of the last layer σ_T is typically taken to be the identity. The special case of a neural operator with a single layer has the form $\mathcal{G}_\theta^{(1)} = \mathcal{Q} \circ \sigma\left(W + \mathcal{K} + b\right) \circ \mathcal{P}$. For the linear operators $\mathcal{K}$, we consider primarily integral operators of the form

$$\mathcal{K}\left[v\right](x) = \int_D k(x,y)\, v(y)\ \mathrm{d}\mu(y). \tag{2}$$

The μ is a measure on $D \subset \mathbb{R}^{d_v}$, $v : D \to \mathbb{R}^{d_v}$ is the input function, and k is a kernel $k(x,y) \in \mathbb{R}^{d_v} \times \mathbb{R}^{d_v}$. For each layer t, we consider a trainable kernel $k = k(x, y; \theta_t)$ parameterized by θ_t for fully connected neural networks having layer-widths $(d_a, n/4, n/2, n, d_v^2)$ for $n \in \mathbb{N}$, as in [40].

In practice, we fix d_v to be the latent dimension of the nodal features within the hidden layers and we use Lebesgue measure for μ. For activation functions, we use the ReLU on all internal layers given their universal

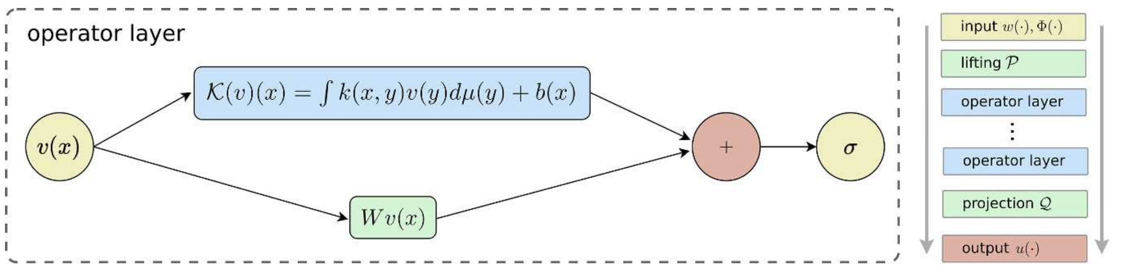

Figure 1. Deep Learning Methods for Operators. An operator layer is used as part of processing input functions. For a function $v(\cdot)$, an affine operation is performed based on integration against a kernel $k(x,y)$ and adding a bias $b(\cdot)$. An additional skip connection with a local linear operator W is also added to the pre-activation output of the layer. These linear operations are then followed by applying the activation function $\sigma(\cdot)$ *(left)*. In combination with the operations of lifting $\mathcal{P}$ and projection $\mathcal{Q}$, these layers are stacked to process input functions to obtain deep learning methods for approximating operators *(right)*.

approximation properties and widely-established use in deep learning [31, 41]). Other choices and functional forms for $\mathcal{K}$ and the kernels k also can be used to further adapt our techniques for special classes of problems.

2.1. Approximating the integral operations

For approximating the integral operations on general manifolds, we develop methods for approximating the integral operations required to evaluate $\mathcal{K}$ building on graph neural operators [31]. We further develop methods using sparse kernel evaluations and specialized constraints on the form of k in the learned kernels. As an initial approximation of the integral operator $\mathcal{K}$, consider using J sample points $\{x_k\}_{k=1}^J$ to obtain

$$\mathcal{K}[v](x_i) \approx \frac{1}{J}\sum_{j=1}^{J} k(x_i, x_j)\, v(x_j)\,,\ i = 1,\ldots,J. \tag{3}$$

Here, we assume the measure μ in $\mathcal{K}$ is normalized to have $\mu(D) = 1$. Direct evaluation of these expressions gives a computational complexity $O(J^2)$.

To help manage these computational costs, we will approximate $\mathcal{K}$ using a few approaches to control the size of J, and denote this approximation $\tilde{\mathcal{K}}$. This includes (i) using a sparse sub-sampling of the points $\{x_i\}$, and (ii) truncating the domain of integration to $S(x)$, such as a ball $B_r(x)$ of radius r, see figure 2. Using these approaches, we approximate the kernel operator $\mathcal{K}$ by

$$\tilde{\mathcal{K}}[v](x) = \int_{S(x)} k(x,y)\, v(y)\ \mathrm{d}y, \quad \forall x \in D. \tag{4}$$

The truncation $S(x) \subset D$ is given by $S: D \to \mathcal{B}(D)$ which maps x to the neighborhood $S(x)$. The $\mathcal{B}(D)$ denotes the subsets of D that are Lebesgue measurable. We denote the indicator function for such sets as $\mathrm{d}\mu(x,y) = \mathbb{1}_{S(x)}\mathrm{d}y$. While more general choices can also be made, we will primarily use $S(x) = B_r(x)$ for a ball of radius r centered at x. We remark that while the truncations may seem to reduce the domain of dependence of the operator to B_r, in fact, the overall operator and information flow can still have longer-range dependencies. In deep learning methods, the operator layers are stacked which successively in the depth from the previous layers increases the domain of dependence of the overall operator, see figure 2.

We compute our approximations using random sampling and the following steps. First, a graph is constructed using nodes $\{x_1,\ldots,x_D\} \subset D$ with features $a(x_i)$ at x_i. We construct directed edges to all nodes $y \in S(x)$ with edge features $k(x,y)$. For a given node x_j having value $v_t(x_j)$ and neighborhood $\mathcal{N}(x_j) = S(x_j) \cap \{x_1,\ldots,x_J\}$, we update the value at node x_j using the mean aggregation

$$\tilde{\mathcal{K}}[v_t](x_j) = \frac{1}{|\mathcal{N}(x_j)|}\sum_{x_k \in \mathcal{N}(x_j)} k(x_j, x_k)\, v_t(x_k)\,. \tag{5}$$

We remark that for such integration operations on manifolds, the accuracy of approximations depend on the sample points $\{x_i\}$, which must be taken sufficiently dense to capture both the local shape of the manifold and the local surface fields. These properties can be characterized using quantities, such as the fill-distance and estimated curvatures of the manifolds [5].

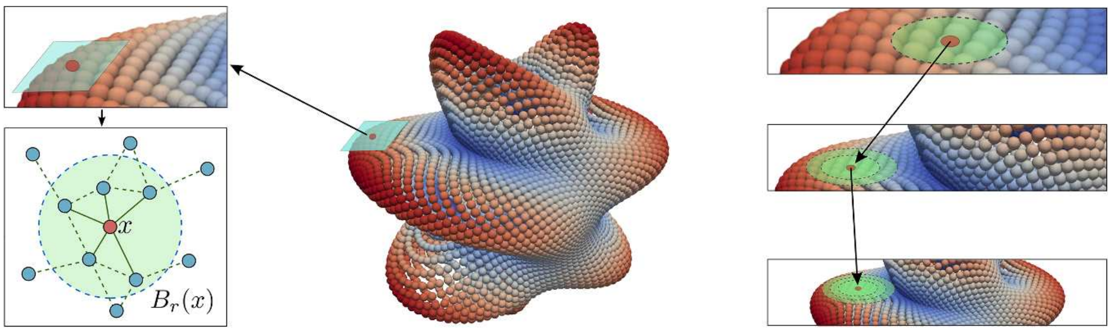

Figure 2. Approximating $\mathcal{K}[v]$: graph-based Approaches and Truncations. We develop methods for general manifolds, including with point-cloud representations *(middle)*. We approximate integral operations by using graph neural operators and message passing. To evaluate $\mathcal{K}[v](x)$ at node x, we use mean aggregation. The nodes x and y have an edge connection in the graph when $\|x-y\| < r$. The graph has node attributes $v(y)$ and edge attributes $k(x,y)$. We take the mean over all updates $k(x,y)v(y)$ for neighboring nodes $\mathcal{N}(x)$ to obtain the approximation of $\mathcal{K}[v]$. To help make computations more efficient, we also truncate the neighborhood to a ball $B_r(x)$ of radius r *(left)*. In deep learning methods, stacking the operator layers increases successively in depth from the previous layers the effective domain of dependence of the overall operator *(right)*.

2.2. Kernel restrictions with factorizations and block-reductions

The sampling of the kernel evaluations correspond to evaluating a linear operator equivalent to the action of a $d_v \times d_v$ matrix. In back-propagation, these calculations can readily exhaust memory during the gradient computations during training. To help mitigate such computational issues as the latent dimension d_v becoming large, we have developed further specialized functional forms and restrictions for the trainable kernels.

Since edges are more numerous than nodes, these contributions dominate the calculations, and we seek restrictions that limit their growth. For this purpose, we consider kernels that can be factored as $k(x,y) = W_k\tilde{k}(x,y)$ where $\tilde{k}(x,y)$ is block diagonal

$$\tilde{k} = \begin{bmatrix} B_1 & 0 & 0 & 0 \\ 0 & B_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & B_c \end{bmatrix}. \tag{6}$$

This consists of c blocks denoted by $B_i = B_i(x,y)$ each having the shape $d_{v'} \times d_{v'}$. We remark that a block form for kernels was also considered and found to be helpful in the setting of regular grids for a FNO in [42]. A notable feature of our factorized form is that the trainable W_k does not depend on the inputs (x,y). The integral operator with this choice for k can be expressed as

$$\tilde{\mathcal{K}}[v_t] = \int_D k(x,y)\,v(y)\;\mathrm{d}y = W_k\int_D \tilde{k}(x,y)\,v(y)\;\mathrm{d}y = \sum_{i=1}^{c} W_{k,i}\int_D B_i(x,y)\,v(y)\;\mathrm{d}y. \tag{7}$$

We have the weights $W_k \in \mathbb{R}^{d_v} \times \mathbb{R}^{cd_{v'}}$ and $W_{k,i} \in \mathbb{R}^{d_v} \times \mathbb{R}^{d_{v'}}$. Since the kernels are represented by fully connected neural networks, our factorized kernel form also provides further savings through the ability to use fewer total weights. Even if considering only the final layer of a general kernel k, this would consist of $w_n d_v^2$ parameters. The w_n is the width of the kernel network. When we use $\tilde{k}$ instead, the final layer becomes $w_n c d_{v'}^2$ parameters, which given the quadratic dependence can be a significant savings. Further, moving the matrix W_k outside of the integral avoids having to apply W directly to each edge weight $k(x,y)$. The factorization allows for only applying W_k on the nodes of the graph after the aggregation step of the edge convolution, resulting in further savings. These approaches taken together are used in performing the steps in the operator layer to obtain

$$v_{t+1}(x) = \sigma_t\left(Wv_t(x) + \tilde{\mathcal{K}}_t[v_t] + b_t\right). \tag{8}$$

The $\tilde{\mathcal{K}}_t$ uses the block diagonal kernel $\tilde{k}_t$. The factorized form for the kernel allows for use of larger latent-space dimensions and for deploying weights and computations into other parts of the neural operator approximation. The approach also allows for overall savings in memory and computational time during training.

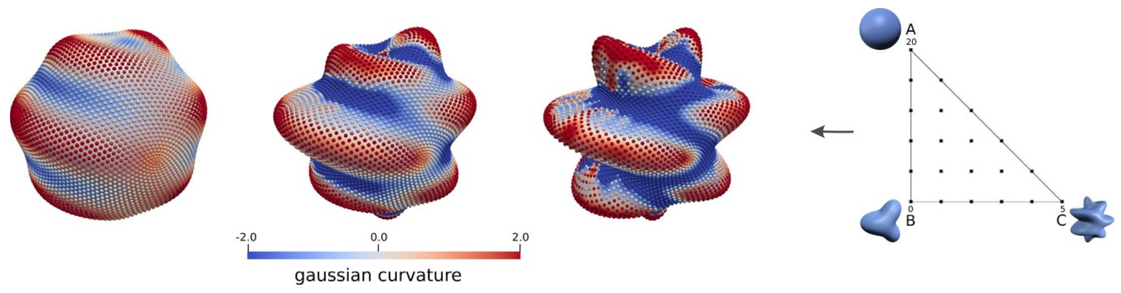

Figure 3. Geometric Quantities for Manifolds with Point-Cloud Representations.We develop geometric neural operators (GNPs) to estimate geometric quantities, such as the metric and curvatures, from manifolds with point-cloud representations. Shown are a few example shapes and their Gaussian curvatures *(left)*. The methods are trained on random shapes obtained by using barycentric coordinates to combine a collection of reference manifolds A, B, C depicted at the vertices with functional forms given in [45] *(right)*.

3. Learning geometric quantities for manifolds with point-cloud representations

Important contributions are made by geometry in many machine learning tasks, such as classifying shapes or approximating solutions of PDEs on manifolds. We develop geometric neural operators (GNPs) for estimating geometric quantities, such as the metric and curvatures, from point-cloud representations of manifolds. To demonstrate the methods, we consider the setting of radial manifolds $\mathcal{M}$, and an embedding $\boldsymbol{\sigma}(\theta,\phi)$ taking values from a coordinate chart (θ,ϕ) into $\mathbb{R}^3$, [43]. We focus on learning the first $\mathbf{I}$ and second $\mathbf{II}$ fundamental forms of differential geometry [44]. These can be expressed in terms of the embedding map $\boldsymbol{\sigma}$ as

$$\mathbf{I} = \begin{bmatrix} E & F \\ F & G \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_\phi \cdot \boldsymbol{\sigma}_\phi & \boldsymbol{\sigma}_\phi \cdot \boldsymbol{\sigma}_\theta \\ \boldsymbol{\sigma}_\theta \cdot \boldsymbol{\sigma}_\phi & \boldsymbol{\sigma}_\theta \cdot \boldsymbol{\sigma}_\theta \end{bmatrix}, \qquad \mathbf{II} = \begin{bmatrix} L & M \\ M & N \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma}_{\phi\phi} \cdot \boldsymbol{n} & \boldsymbol{\sigma}_{\phi\theta} \cdot \boldsymbol{n} \\ \boldsymbol{\sigma}_{\theta\phi} \cdot \boldsymbol{n} & \boldsymbol{\sigma}_{\theta\theta} \cdot \boldsymbol{n} \end{bmatrix}. \tag{9}$$

The $\boldsymbol{\sigma}_\theta = \partial_\theta \boldsymbol{\sigma}$, $\boldsymbol{\sigma}_\phi = \partial_\phi \boldsymbol{\sigma}$, and similarly for higher-order derivatives. The outward normal $\boldsymbol{n}$ is given by

$$\boldsymbol{n}(\theta,\phi) = \frac{\sigma_\phi(\theta,\phi) \times \sigma_\theta(\theta,\phi)}{||\sigma_\phi(\theta,\phi) \times \sigma_\theta(\theta,\phi)||}. \tag{10}$$

These forms can be used to construct the Weingarten map as $\boldsymbol{W} = \mathbf{I}^{-1}\mathbf{II}$. We use $\boldsymbol{W}$ to compute the Gaussian curvature

$$K(\theta,\phi) = \det(\boldsymbol{W}(\theta,\phi)). \tag{11}$$

We consider the learning tasks for a collection of different radial manifold shapes given by combinations using barycentric coordinates arranged as in figure 3. Each of the radial manifolds are represented by a collection of spherical harmonic coefficients for the radial function $r(\theta,\phi)$, as in our prior work [43]. These manifold shapes are sampled by using uniform random variables $u_1, u_2 \sim \mathcal{U}[0,1)$, to obtain coefficients $\boldsymbol{d} = (1-\sqrt{u_1})\boldsymbol{a} + (1-u_2)\sqrt{u_1}\boldsymbol{b} + \sqrt{u_1}u_2\boldsymbol{c}$ where $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ are vectors for the collection of spherical harmonic coefficients. For our training dataset we sample 500 manifold shapes. We also sample 200 manifold shapes as a test dataset. We should mention these intrinsic manifold shapes from the spherical harmonics will primarily be used in our analysis and are treated distinct from the sampled point-cloud representations of the shapes used in training and testing.

To obtain point-cloud representations for each of the manifold shapes we use the vector of spherical harmonic coefficients $\boldsymbol{d}$ and a separate random sampling of $N = 1024$ points $\{\mathbf{x}_i\}_{i=1}^N$ on the sphere with $\mathbf{x}_i$ projected to the manifold surface. We then use the spherical harmonic interpolation to obtain training data for the geometric quantities $\mathbf{z} = \mathbf{z}(\mathbf{x}_i;\mathbf{d}) = (x,y,z,E,F,G,L,M,N,K)$ for the manifold surface with shape $\mathbf{d}$ and point-cloud sample at location $\mathbf{x}_i = (x,y,z)$, and E,F,G,L,M,N,K from the first and second fundamental forms $\mathbf{I}, \mathbf{II}$.

We also add noise to the datasets to obtain training samples $\tilde{\mathbf{z}} = \mathbf{z} + \boldsymbol{\xi} = (x,y,z,E,F,G,L,M,N,K) + \boldsymbol{\xi}$, where $\boldsymbol{\xi} \sim \mathcal{N}(0,\epsilon^2\Lambda)$ is Gaussian noise. The ϵ gives the relative noise for the reference scales Λ given by $\Lambda = \mathrm{diag}(\{\lambda_i\})$ with $\lambda_i = ||(x,y,z)||_2^2$, $i = 1,2,3$; $\lambda_i = ||(E,F,G)||_2^2$, $i = 4,5,6$; $\lambda_i = ||(L,M,N)||_2^2$, $i = 7,8,9$; $\lambda_i = |K|^2$, $i = 10$. To further test robustness of the methods, we also consider two additional datasets with (i) uniform noise at 1% added to all data components for each manifold shape, and (ii) by creating noisy perturbed outlier points by adding noise at 10% to 50 randomly chosen data points in each manifold shape.

We train the geometric neural operators (GNPs) to learn the geometric quantities from the point cloud representation $\{\boldsymbol{x}_j\}_{j=1}^{N} \subset \mathbb{R}^3$ with $N = 1024$ surface points for a manifold $\mathcal{M}^{(i)}$. The GNP $u = \mathcal{G}_\theta[\{\mathbf{x}_j\}]$ must learn from the point cloud $\{\mathbf{x}_j\}$ using coordinates from the embedding space $\mathbb{R}^3$. The task is to estimate the fundamental forms $\mathbf{I}, \mathbf{II}$ by learning a mapping u which can be expressed as

$$\boldsymbol{x}_j \mapsto \left(E\left(\boldsymbol{x}_j\right), F\left(\boldsymbol{x}_j\right), G\left(\boldsymbol{x}_j\right), L\left(\boldsymbol{x}_j\right), M\left(\boldsymbol{x}_j\right), N\left(\boldsymbol{x}_j\right), K\left(\boldsymbol{x}_j\right)\right). \tag{12}$$

In the training, we also include learning the Gaussian curvature K. We use a few different architectures for our GNPs. These are (i) use of a kernel approximation as in equation (5) which we call the *full kernel* and (ii) use of a factorization and block-reductions as in equation (7) which we call the *factorized block kernel*. In each case, we consider networks with $d_v = 64$, kernel width up to $w_n = 256$, and depth up to $d_n = 10$ integral operator layers. In our studies, we found that the latent dimension d_v and the depth d_n had the strongest influence on performance. Increasing the kernel width beyond $w_n = 256$ did not appear to lead to performance gains in general. We use for our loss function the L^2-norm for comparing the GNP predicted values of the model $\mathcal{G}_\theta$ with the true values from the shapes obtained from the spherical harmonics representations. We trained using optimization methods based on stochastic gradient descent with momentum using the Adam method [46].

We show results for training the GNPs with different choices for the neural network architectures and other hyper-parameters in table 1. We varied in the studies the width w_n of the neural networks between 128 and 256. We also varied the depth T of the network from 8 to 10 layers which from stacking increases the effective range of the domain of dependence of the overall operator, see figure 2. The results show the GNPs can learn from the point-cloud accurate representations simultaneously the metric components E, F, G of the first fundamental form $\mathbf{I}$ and the curvatures components L, M, N, K of the second fundamental form $\mathbf{II}$ and Gaussian curvature. This provides the basis for performing further many downstream tasks and analysis using concepts from differential geometry and the trained GNPs. The trained GNPs had an overall L^2-error around 5.19×10^{-2} when there were no noise perturbations. In the case of 1% noise perturbations in the training dataset, we find an L^2-error of around 9.55×10^{-2}. We find in the case of outliers the L^2-error becomes around 1.49×10^{-1}. While there is some degradation in accuracy when noise is added to the dataset, the results show the methods are overall robust providing decent estimates of the geometric quantities. We also find our factorized block-kernel methods can train to a comparable level of accuracy as the full kernel and run about three times faster during training. In particular, using 10 layers with a kernel width of $w_n = 256$, the full kernel GNPs trained over an average of 16 hours while the factorized-block GNPs trained over 7 hours. Further, the memory requirements were reduced from 24GB with the full kernel to 8GB with the factorized block-kernel. These results indicate the GNPs can be trained effectively to obtain robust methods for estimating the metrics and curvatures of manifolds from their point-cloud representations.

4. Learning solution maps of PDEs on manifolds: LB-P problems

We develop GNPs to learn the solution map for LB PDEs on manifolds. We consider LB-P problems of the form

$$\left\{\begin{array}{lll} \Delta_{\mathrm{LB}}\, u & = & -f \\ \int_{\mathcal{M}} u(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} & = & 0 \end{array}\right\}. \tag{13}$$

The $\mathcal{M}$ denotes the manifold, Δ_{LB} the LB operator, u the solution, and f the right-hand-side (rhs) data. In contrast to Euclidean operators, computing the forward or backward LB operator requires the geometric information such as the first fundamental form given in equation (9) and it is derivatives. Therefore, in approximating either of the operators it a necessary that the data-driven methods learn features capturing geometric properties of the manifold $\mathcal{M}$. When considering closed manifolds $\mathcal{M}$, there can be additional constants required in geometric PDEs related to the topology [43, 45]. For the spherical topology here, there is one additional constant which we determine by imposing that the solution integrates to zero. For additional discussions and details, see appendix.

The task here is to learn an operator for the solution mappings $u = -\Delta_{\mathrm{LB}}^{-1} f$ for the PDE on the manifold surface for each of the reference shapes defined in figure 4. We will train separate GNPs for each manifold to effectively learn a solution operator $u = \mathcal{G}_\theta[\{\mathbf{x}_j\}, -f]$. For this geometric elliptic PDE, learning such an operator for a sufficiently rich set of source functions f can be combined with super-position principles or Green's function representation formulas to construct more general solutions on the manifold. For such non-constant coefficient linear PDEs, there often are not tractable analytic expressions for the Green's functions which pose practical challenges in obtaining representations for general solutions. We show how the data-driven methods of GNPs can be utilized to learning solution operators for f. For this purpose, we

Table 1. Results for Learning Joint Geometric Properties. The GNPs were trained with full and factorized kernels to estimate metrics and curvatures from manifolds with point-cloud representations. The methods were tested on random shapes obtained by sampling from shapes in figure 3. Results are reported in relative error for cases with and without noise ϵ, with the most accurate result in bold for each study. We report the mean ± the standard deviation of the results over 5 training runs. The sampling resolution for each shape was $N = 1024$ and a radius of $r = 0.4$ for the kernel integrations. The GNPs were trained for 500 epochs with an initial learning rate of 10^{-4} which was halved every 100 epochs. ReLU activations were used both in the kernel networks and in the operator layers.

Depth	Neurons	Training error	Test error	Training error	Test error
Architecture		Full kernel		Factorized block-kernel	
10	256	$2.52\times10^{-02}\pm6.5\times10^{-04}$	$\mathbf{3.82\times10^{-02}\pm1.7\times10^{-03}}$	$4.23\times10^{-02}\pm7.6\times10^{-04}$	$\mathbf{5.19\times10^{-02}\pm9.3\times10^{-04}}$
10	128	$4.36\times10^{-02}\pm6.0\times10^{-04}$	$5.57\times10^{-02}\pm5.9\times10^{-04}$	$7.73\times10^{-02}\pm3.9\times10^{-03}$	$8.54\times10^{-02}\pm3.1\times10^{-03}$
8	256	$2.83\times10^{-02}\pm2.3\times10^{-04}$	$4.13\times10^{-02}\pm1.3\times10^{-03}$	$5.08\times10^{-02}\pm1.0\times10^{-03}$	$6.00\times10^{-02}\pm9.9\times10^{-04}$
8	128	$5.08\times10^{-02}\pm1.3\times10^{-03}$	$6.29\times10^{-02}\pm1.7\times10^{-03}$	$8.94\times10^{-02}\pm2.3\times10^{-03}$	$9.59\times10^{-02}\pm1.8\times10^{-03}$
1% noise for all points					
10	256	$6.31\times10^{-02}\pm6.3\times10^{-04}$	$\mathbf{8.78\times10^{-02}\pm1.6\times10^{-03}}$	$8.11\times10^{-02}\pm1.1\times10^{-03}$	$\mathbf{9.55\times10^{-02}\pm9.8\times10^{-04}}$
10	128	$8.00\times10^{-02}\pm8.1\times10^{-04}$	$9.83\times10^{-02}\pm7.0\times10^{-04}$	$1.11\times10^{-01}\pm2.5\times10^{-03}$	$1.20\times10^{-01}\pm1.2\times10^{-03}$
8	256	$6.68\times10^{-02}\pm9.8\times10^{-04}$	$8.97\times10^{-02}\pm7.1\times10^{-04}$	$8.63\times10^{-02}\pm9.7\times10^{-04}$	$9.79\times10^{-02}\pm8.3\times10^{-04}$
8	128	$8.75\times10^{-02}\pm5.7\times10^{-04}$	$1.03\times10^{-01}\pm1.4\times10^{-03}$	$1.18\times10^{-01}\pm3.1\times10^{-03}$	$1.26\times10^{-01}\pm3.8\times10^{-03}$
10% noise for 5% of points					
10	256	$1.00\times10^{-01}\pm6.2\times10^{-04}$	$1.49\times10^{-01}\pm1.9\times10^{-03}$	$1.26\times10^{-01}\pm6.4\times10^{-04}$	$\mathbf{1.49\times10^{-01}\pm2.5\times10^{-03}}$
10	128	$1.22\times10^{-01}\pm1.6\times10^{-03}$	$1.54\times10^{-01}\pm2.5\times10^{-03}$	$1.49\times10^{-01}\pm1.1\times10^{-03}$	$1.67\times10^{-01}\pm2.5\times10^{-03}$
8	256	$1.06\times10^{-01}\pm5.1\times10^{-04}$	$\mathbf{1.48\times10^{-01}\pm7.2\times10^{-04}}$	$1.31\times10^{-01}\pm1.3\times10^{-03}$	$1.52\times10^{-01}\pm2.7\times10^{-03}$
8	128	$1.28\times10^{-01}\pm1.6\times10^{-03}$	$1.59\times10^{-01}\pm3.6\times10^{-03}$	$1.58\times10^{-01}\pm1.4\times10^{-03}$	$1.76\times10^{-01}\pm3.0\times10^{-03}$

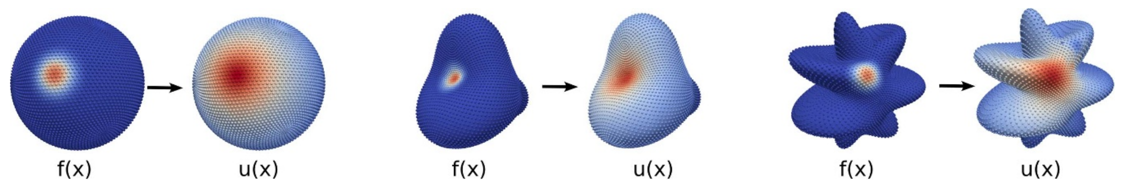

Figure 4. Training Geometric Operators for Solving PDEs on Manifolds. We train neural geometric operators (GNPs) for the solution map of the Laplace–Beltrami PDE. For the reference manifolds A, B, C, we use radial functions $r_A(\theta,\phi) = 1, r_B(\theta,\phi) = 1 + 0.4\sin(3\phi)\cos(\theta), r_C(\theta,\phi) = 1 + 0.4\sin(7\phi)\cos(\theta)$. For additional details on radial manifolds, see appendix. We show some example right-hand-sides (rhs) $f(x)$ and the corresponding solutions $u(x)$. We train with locations of the Gaussian for the rhs varied over the surface and over the collection of shapes.

will train by generating input functions f that approximate Dirac δ-functions sampled at a collection of randomized locations on the manifold. We use a Gaussian having the modified form

$$f(\boldsymbol{x};\bar{\boldsymbol{x}},\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{x}-\bar{\boldsymbol{x}}||^2\right) - c_0(\sigma). \qquad (14)$$

The f have parameters $\sigma = 0.15$ and $\bar{\boldsymbol{x}}$. For the training data, we interpolate f using spherical harmonics with 2030 Lebedev nodes and hyper-interpolation [43, 45]. To obtain random samples for f, we sample Gaussians $\boldsymbol{y} \sim \mathcal{N}(0, I)$ that are radially projected onto the manifold $\mathcal{M}$ to obtain samples for $\bar{\boldsymbol{x}}$. We further choose $c_0(\sigma)$ so that the spherical harmonic interpolations of f always have a zero spatial average $\frac{1}{|\mathcal{M}|}\int_{\mathcal{M}} f(\mathbf{x})\mathrm{d}x = 0$. To obtain the input-solution pairs (f, u) for the training data, we solve the LB equation (13). This is done numerically by building on our prior work on spectral methods for spheres based on Galerkin truncations over the spherical harmonics with $N = 2023$ Lebedev nodes [45, 47]. We remark that one could in principle start with u and compute the action of the differential operator $-\Delta_{\mathrm{LB}}[u]$ to obtain f. However, in practice, we find from the differentiations involved that this is much more sensitive to noise resulting in poorer quality training sets. This indicates when performing operator training, it is preferred to learn by sampling the underlying approximation of the solution map when it is available.

To train and test our GNPs, we use 1000 random samples of f to obtain the training set and another 200 samples for testing. We consider the final resolutions for the input-solution pairs (f, u) using point-clouds with $N = 1024$ spatial sample locations. Our loss function are based on the L^2-norm integrated over the manifold surface for the difference $u - \tilde{u}$ between the GNP predicted solution $\tilde{u}$ and the spectral solution of the PDE u. We trained using optimization methods based on stochastic gradient descent with momentum using the Adam method [46]. We used 300 epochs with a batch size of 1 along with an initial learning rate of 10^{-4} that was halved every 75 epochs. During training, using 10 layers with a kernel width of $w_n = 256$, the full kernel GNPs averaged 21 h utilizing approximately 24GB of GPU memory. The factorized block-kernel averaged 7 h for training and required approximately 8GB of memory on the same hardware.

We show results of our training of GNPs for each of the manifolds $\mathcal{M}$ in table 2 and figure 5. We considered GNPs with a few different choices for the neural network architectures. We find our factorized block-kernel methods perform comparable to the full kernel GNPs. The accuracy over all the manifolds is found to be 9.03×10^{-2}. The most accurate results of 1.07×10^{-2} were obtained for Manifold A. In this case, the manifold is a sphere and the GNPs appear to benefit from the symmetry. We further see in this case the depth and width of the neural networks do not have as much impact on the accuracy. As the shapes become more complicated, such as for Manifold B and Manifold C, the learning problem becomes more challenging. This would be expected given the more heterogeneous contributions of the curvature within the differential operators in the LB equation (13). Relative to Manifold A, the Manifold B is less symmetric and we find an accuracy of 3.75×10^{-2}. We see the depth and width of the GNPs has a more significant impact on the accuracy, especially for the factorized case. This could in part arise from the role depth plays in increasing the domain of dependence of the operator. For geometric quantities and computing differential operations, there could be benefits from having a wider range of points over which to make estimates. In the case of the more complicated Manifold C, we find an accuracy of 9.03×10^{-2}. We see the depth and width of the neural network again has a significant impact on the accuracy. Overall, the results indicate that GNPs can learn solution maps over a wide variety of shapes for geometric PDEs.

Table 2. Results for Learning Laplace–Beltrami Solution Operator. For the manifolds A, B, C, we show relative training and test errors of GNPs for the learned solution operator for the Laplace–Beltrami PDE in equation (13). We report the mean $\pm$ the standard deviation of the results over 5 training runs. We show a few different choices for the GNP architectures with the most accurate results shown in bold for each study.

Neurons	Depth	Training error	Test error	Training error	Test error
Manifold A		Full kernel		Factorized block-kernel	
256	10	$7.50 \times 10^{-03} \pm 7.1 \times 10^{-04}$	$\mathbf{8.98 \times 10^{-03} \pm 8.5 \times 10^{-04}}$	$9.72 \times 10^{-03} \pm 4.5 \times 10^{-04}$	$\mathbf{1.07 \times 10^{-02} \pm 3.9 \times 10^{-04}}$
128	10	$8.25 \times 10^{-03} \pm 5.8 \times 10^{-04}$	$9.66 \times 10^{-03} \pm 6.5 \times 10^{-04}$	$1.29 \times 10^{-02} \pm 3.6 \times 10^{-04}$	$1.37 \times 10^{-02} \pm 4.3 \times 10^{-04}$
256	8	$9.42 \times 10^{-03} \pm 7.0 \times 10^{-04}$	$1.09 \times 10^{-02} \pm 7.5 \times 10^{-04}$	$1.13 \times 10^{-02} \pm 8.6 \times 10^{-04}$	$1.23 \times 10^{-02} \pm 9.3 \times 10^{-04}$
128	8	$1.03 \times 10^{-02} \pm 4.6 \times 10^{-04}$	$1.16 \times 10^{-02} \pm 4.7 \times 10^{-04}$	$1.50 \times 10^{-02} \pm 7.4 \times 10^{-04}$	$1.59 \times 10^{-02} \pm 7.7 \times 10^{-04}$
Manifold B					
256	10	$1.85 \times 10^{-02} \pm 1.1 \times 10^{-03}$	$\mathbf{3.32 \times 10^{-02} \pm 6.9 \times 10^{-04}}$	$2.74 \times 10^{-02} \pm 1.3 \times 10^{-03}$	$\mathbf{3.75 \times 10^{-02} \pm 1.4 \times 10^{-03}}$
128	10	$2.45 \times 10^{-02} \pm 1.2 \times 10^{-03}$	$3.74 \times 10^{-02} \pm 1.4 \times 10^{-03}$	$3.75 \times 10^{-02} \pm 1.0 \times 10^{-03}$	$4.40 \times 10^{-02} \pm 9.1 \times 10^{-04}$
256	8	$2.43 \times 10^{-02} \pm 9.7 \times 10^{-04}$	$3.84 \times 10^{-02} \pm 7.9 \times 10^{-04}$	$3.41 \times 10^{-02} \pm 1.0 \times 10^{-03}$	$4.24 \times 10^{-02} \pm 1.0 \times 10^{-03}$
128	8	$3.07 \times 10^{-02} \pm 4.5 \times 10^{-04}$	$4.19 \times 10^{-02} \pm 6.0 \times 10^{-04}$	$4.40 \times 10^{-02} \pm 8.7 \times 10^{-04}$	$4.93 \times 10^{-02} \pm 8.8 \times 10^{-04}$
Manifold C					
256	10	$3.93 \times 10^{-02} \pm 1.9 \times 10^{-03}$	$\mathbf{8.31 \times 10^{-02} \pm 2.1 \times 10^{-03}}$	$6.48 \times 10^{-02} \pm 2.8 \times 10^{-03}$	$\mathbf{9.03 \times 10^{-02} \pm 1.1 \times 10^{-03}}$
128	10	$5.31 \times 10^{-02} \pm 3.5 \times 10^{-03}$	$8.61 \times 10^{-02} \pm 2.2 \times 10^{-03}$	$8.58 \times 10^{-02} \pm 7.8 \times 10^{-04}$	$1.03 \times 10^{-01} \pm 1.3 \times 10^{-03}$
256	8	$5.61 \times 10^{-02} \pm 1.5 \times 10^{-03}$	$9.14 \times 10^{-02} \pm 1.3 \times 10^{-03}$	$8.02 \times 10^{-02} \pm 1.5 \times 10^{-03}$	$1.00 \times 10^{-01} \pm 1.4 \times 10^{-03}$
128	8	$6.84 \times 10^{-02} \pm 2.6 \times 10^{-03}$	$9.39 \times 10^{-02} \pm 2.0 \times 10^{-03}$	$9.97 \times 10^{-02} \pm 1.3 \times 10^{-03}$	$1.14 \times 10^{-01} \pm 1.7 \times 10^{-03}$

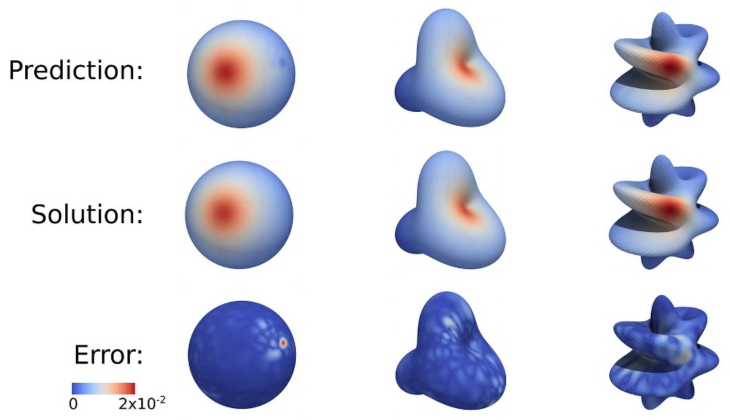

Figure 5. We show a few example predictions of the full kernel GNPs for the Laplace–Beltrami PDE solutions and their absolute errors for manifolds A, B, C.

5. Estimating manifold shapes: Bayesian inverse problems using observations of LB responses

We developed GNPs for estimating manifold shapes from observations of the action of the LB operator on the manifold. We use as observations the input-solution pairs (u, f) for the LB-P PDE in equation (13). For this purpose, we train GNPs to be used in conjunction with solving a Bayesian Inverse Problem. This consists of the following steps (i) formulate a prior probability distribution over the shape space, (ii) observe the LB responses for a collection of samples $\{(u^{(i)}, f^{(i)})\}_{i=1}^{m}$, (iii) use Bayes' rule to obtain a posterior probability distribution, (iv) perform optimization to find the most likely manifold shape under the posterior distribution. We train GNPs over both manifold shapes and LB responses. The GNPs serve to map manifold shapes to LB responses to compute likelihoods in place of a traditional solver. In comparison, GNPs are able to perform inference more rapidly, making them an efficient surrogate for obtaining a posterior distribution. As we shall discuss in more detail below, the likelihoods $p(\mathcal{M}|(u,f))$ are taken as Gaussians based on the L^2-norm for the difference $\tilde{u} - u$ between the GNP predicted solution $\tilde{u} = \mathcal{G}_\theta[\{\mathbf{x}_j^{\mathcal{M}}\}, f]$ and the data u. We use this in conjunction with Bayesian inference to assign a posterior distribution and obtain a maximum a posteriori (MAP) estimate for the shape $\mathcal{M}$. We can also use the GNPs to obtain an estimate of the full posterior distribution by using them for Monte-Carlo sampling.

We consider the inverse problem of using observation data $\mathcal{D}$ to recover $\mathcal{M}$. We consider radial manifolds $\mathcal{M}$ and the responses of LB operators $\Delta_{\mathrm{LB}}[u; \mathcal{M}]$ when applied to functions $u : \mathcal{M} \to \mathbb{R}$. The observation data is $\mathcal{D} = \{(u^i(\mathbf{x}_j), f^i(\mathbf{x}_j))\}_{i,j}$ for $j = 1, \ldots, N$ with $\{\mathbf{x}_j\}$ sampled uniformly on S^2 and radially projected to $\mathcal{M}$ and where $\Delta_{\mathrm{LB}}[u^i; \mathcal{M}] = -f^i$. The manifold shapes use the barycentric coordinates in figure 3. As discussed in section 3, the manifold shapes can be described by spherical harmonic coefficient vectors $\mathbf{d}$. We obtain responses by considering rhs-functions f as in equation (14). To obtain the response training data pairs $\{(u^i, f^i)\}$, we use the spectral solvers in our prior work [43, 45]. We obtain f^i by sampling $\bar{\boldsymbol{x}}^i$ in equation (14) at $M = 194$ Lebedev nodes for the order 12 hyper-interpolation [45]. Our spectral solver then yields a spherical harmonics representation of u^i. We further sample 21 manifold shapes from the barycentric coordinates as in figure 3. This is used to construct the full set of LB response pairs $\{(u^i, f^i)\}$ across all shapes. This yields a training set consisting of 4074 samples of LB response pairs. We remark that this problem requires that the GNP learn meaningful geometric features in order to approximate how the solution operator of equation (13) changes as the geometry is varied.

We train our factorized block-kernel GNPs by using data samples of the from $(\mathcal{M}^{(i)}, u^{(i)}, f^{(i)})$. This consists of a point-cloud sampling of the manifold geometry $\mathcal{M}^{(i)} = \{\mathbf{x}_j\}_{j=1}^{N}$ and a LB response pair $(u^{(i)}, f^{(i)})$. We sample the geometry and responses using a collection of $N = 1024$ points $\boldsymbol{x}_j^{(i)} \in \mathcal{M}^{(i)}$ to yield $(\boldsymbol{x}_j^{(i)}, u^{(i)}(\boldsymbol{x}_j^{(i)}), f^{(i)}(\boldsymbol{x}_j^{(i)}))$ and add 3% Gaussian noise relative to the function values. The GNPs are trained to learn the solution operator on the noisy data $(\boldsymbol{x}_j^{(i)}, \tilde{f}^{(i)}(\boldsymbol{x}_j^{(i)})) \mapsto \tilde{u}^{(i)}\left(\boldsymbol{x}_j^{(i)}\right)$. We use for the loss function the L^2-norm of the difference $\tilde{u} - u$ between the GNP prediction of the solution $\tilde{u}$ and the solution u obtained from the spectral solvers. For our GNP architectures we used $d_v = 64$ with 4 diagonal blocks in the kernel network output, kernel widths of $w_n = 256$, and depths of $d_n = 10$ for the integral operator layers. For the optimization we use stochastic gradient descent with momentum based on Adam [46]. We trained using 300

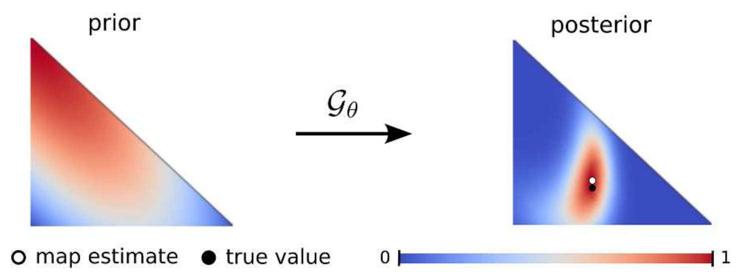

Figure 6. Manifold Shape Estimation: Prior Distribution. The prior distribution over manifolds $\mathcal{M}$ based on the radial shape covariances when $\beta = 1$. This is used in combination with Bayes' Rule and the geometric neural operators (GNPs) to obtain a posterior distribution over manifold shapes. We show density functions for our prior distribution given by equation (18), and posterior computed using our GNP and Bayes' Rule.

epochs with a batch size of 4. Our learning rate was set to 10^{-3} and was decreased to 10^{-6} over 60 epochs using cosine annealing [48]. In our cosine annealing, we cycled the learning rate every 60 epochs to be restarted back at 10^{-3}. Training required 25GB of GPU memory and an average of 18 h.

We used our trained GNPs $\mathcal{G}_\theta$ to perform Bayesian inference. This requires developing a prior distribution $p(\mathcal{M})$ and likelihood distribution $p(\mathcal{D}|\mathcal{M})$. We then seek to estimate a posterior distribution $p(\mathcal{M}|\mathcal{D})$ from the observation data $\mathcal{D}$ to assign a probability that the manifold shape was $\mathcal{M}$. We use Bayes' Rule to obtain

$$p(\mathcal{M}|\mathcal{D}) = \frac{1}{Z} p(\mathcal{D}|\mathcal{M})\, p(\mathcal{M}), \tag{15}$$

where Z is a normalization factor so the probabilities total to one.

We solve the inverse problem in practice by using our GNPs $\mathcal{G}_\theta$ as surrogate models for the LB responses. We obtain likelihoods by considering

$$\xi^2 = \frac{1}{M}\sum_{i=1}^{M} \frac{\left|\left|u^{(i)} - \mathcal{G}_\theta\left[\left\{\mathbf{x}_j^{(i)}\right\}, f^{(i)}\right]\right|\right|^2}{||u^{(i)}||^2}. \tag{16}$$

The sum here is taken over the M samples of the responses. We approximate the L^2-norms by performing further summation over the point-cloud samples at $\{\mathbf{x}_j\}$. This yields the likelihood

$$p(\mathcal{D}|\mathcal{M}) = \left(2\pi\sigma^2\right)^{-1/2} \exp\left(-\frac{\xi^2}{2\sigma^2}\right). \tag{17}$$

We take here $\sigma^2 = 10^{-3}$. We develop a prior distribution for use over the manifolds $\mathcal{M}$ of the form

$$p(\mathcal{M}) \propto \exp(-\beta F(\mathcal{M})). \tag{18}$$

We choose F to characterize the complexity of the manifold geometry by using the radial shape functions $r_{\mathcal{M}}(\theta,\phi)$. We use the covariance of the radial function $r_{\mathcal{M}}$ to obtain

$$F(\mathcal{M}) = \int \left(r_{\mathcal{M}}(\theta,\phi) - \mu_r\right)^2 \mathrm{d}\theta \mathrm{d}\phi.$$

The mean radius is given by $\mu_r = \int r_{\mathcal{M}}(\theta,\phi)\mathrm{d}\theta\mathrm{d}\phi$. The F can be thought of as a free energy for the manifold shapes. The β acts like an inverse temperature that controls the characteristic scales at which to put emphasis on the differences in the radial covariances. We use as our default value $\beta = 1$. This provides a prior distribution that serves to regularize results toward simpler shapes, such as a sphere which has the smallest radial covariance. We show our prior distribution over the barycentric interpolated shape space in figure 6.

We test our GNP-Bayesian methods by considering an underlying target true manifold $\mathcal{M}^*$ and constructing M observation samples for the LB responses $\mathcal{D} = \left\{\left(u^{(i)}, f^{(i)}\right)\right\}_{i=1}^{M}$, for $M = 3, 5$. For each sample $i = 1, \ldots, M$, we sample the manifold geometry to obtain a point-cloud representation $\left\{\boldsymbol{x}_j\right\}_{j=1}^{N}$ with $N = 1024$ points. These were used to construct observation data $\mathcal{D}$ for testing our inference methods.

We show results of our GNP-Bayesian methods for manifold shapes in table 3 and figure 7. We considered the cases for both 5 and 3 samples of the LB response pairs (u, f). We report in our results the top two predictions $P1$, $P2$ for the manifold. We also give the Bayesian posterior likelihood associated with the

Table 3. Results for Manifold Shape Identification: Bayesian Inference based on Laplace–Beltrami Responses.We show GNPs can be used for Bayesian estimates of manifold shapes using observations of the Laplace–Beltrami responses. We show results for both 5 and 3 samples of Laplace–Beltrami pairs (u, f). The first column M gives the true manifold, and the $P1$ and $P2$ give the top two predictions for the manifold. The columns $L1$ and $L2$ give the Bayesian posterior likelihood associated with the predictions of the manifold shape. We report the mean ± the standard deviation of the results over 5 training runs. We highlight in bold the case where the first prediction disagreed with the true manifold. We see the case with only 3 samples had only one error. The case with 5 samples did not have any errors when using the GNPs to predict the shape.

M	$P1$	$P2$	$L1$	$L2$	$P1$	$P2$	$L1$	$L2$
			5 Samples				3 Samples	
0	0	6	$2.82 \times 10^{-01} \pm 4.7 \times 10^{-02}$	$2.43 \times 10^{-01} \pm 3.5 \times 10^{-02}$	**6**	**0**	$2.65 \times 10^{-01} \pm 4.9 \times 10^{-02}$	$2.51 \times 10^{-01} \pm 4.2 \times 10^{-02}$
1	1	7	$2.64 \times 10^{-01} \pm 1.4 \times 10^{-02}$	$1.73 \times 10^{-01} \pm 7.7 \times 10^{-03}$	1	7	$3.13 \times 10^{-01} \pm 1.7 \times 10^{-02}$	$1.87 \times 10^{-01} \pm 1.5 \times 10^{-02}$
2	2	8	$3.19 \times 10^{-01} \pm 1.9 \times 10^{-02}$	$2.60 \times 10^{-01} \pm 1.3 \times 10^{-02}$	2	8	$3.47 \times 10^{-01} \pm 1.2 \times 10^{-02}$	$2.43 \times 10^{-01} \pm 7.4 \times 10^{-03}$
3	3	9	$5.18 \times 10^{-01} \pm 1.8 \times 10^{-02}$	$3.29 \times 10^{-01} \pm 1.9 \times 10^{-02}$	3	9	$4.48 \times 10^{-01} \pm 2.5 \times 10^{-02}$	$3.75 \times 10^{-01} \pm 1.5 \times 10^{-02}$
4	4	10	$5.67 \times 10^{-01} \pm 2.1 \times 10^{-02}$	$3.93 \times 10^{-01} \pm 1.6 \times 10^{-02}$	4	10	$5.40 \times 10^{-01} \pm 4.1 \times 10^{-02}$	$4.30 \times 10^{-01} \pm 2.9 \times 10^{-02}$
5	5	10	$6.97 \times 10^{-01} \pm 8.6 \times 10^{-02}$	$2.69 \times 10^{-01} \pm 7.2 \times 10^{-02}$	5	10	$6.53 \times 10^{-01} \pm 2.0 \times 10^{-01}$	$3.18 \times 10^{-01} \pm 1.7 \times 10^{-01}$
6	6	11	$2.57 \times 10^{-01} \pm 2.5 \times 10^{-02}$	$1.96 \times 10^{-01} \pm 3.2 \times 10^{-03}$	6	7	$2.44 \times 10^{-01} \pm 2.5 \times 10^{-02}$	$1.82 \times 10^{-01} \pm 1.8 \times 10^{-02}$
7	7	12	$2.14 \times 10^{-01} \pm 9.4 \times 10^{-03}$	$1.51 \times 10^{-01} \pm 1.1 \times 10^{-02}$	7	12	$2.57 \times 10^{-01} \pm 1.1 \times 10^{-02}$	$1.41 \times 10^{-01} \pm 1.2 \times 10^{-02}$
8	8	13	$3.16 \times 10^{-01} \pm 6.0 \times 10^{-03}$	$2.32 \times 10^{-01} \pm 1.3 \times 10^{-02}$	8	13	$2.96 \times 10^{-01} \pm 1.2 \times 10^{-02}$	$2.31 \times 10^{-01} \pm 8.6 \times 10^{-03}$
9	9	3	$4.61 \times 10^{-01} \pm 2.8 \times 10^{-02}$	$2.72 \times 10^{-01} \pm 4.5 \times 10^{-02}$	9	3	$4.73 \times 10^{-01} \pm 2.5 \times 10^{-02}$	$3.01 \times 10^{-01} \pm 3.7 \times 10^{-02}$
10	10	4	$5.24 \times 10^{-01} \pm 3.2 \times 10^{-02}$	$3.86 \times 10^{-01} \pm 2.6 \times 10^{-02}$	10	4	$5.61 \times 10^{-01} \pm 5.3 \times 10^{-02}$	$3.42 \times 10^{-01} \pm 3.5 \times 10^{-02}$
11	11	12	$2.17 \times 10^{-01} \pm 1.7 \times 10^{-02}$	$1.58 \times 10^{-01} \pm 2.4 \times 10^{-03}$	11	15	$2.23 \times 10^{-01} \pm 2.6 \times 10^{-02}$	$1.76 \times 10^{-01} \pm 1.6 \times 10^{-02}$
12	12	16	$2.45 \times 10^{-01} \pm 1.4 \times 10^{-02}$	$1.91 \times 10^{-01} \pm 1.1 \times 10^{-02}$	12	16	$2.44 \times 10^{-01} \pm 1.3 \times 10^{-02}$	$1.94 \times 10^{-01} \pm 1.0 \times 10^{-02}$
13	13	17	$3.01 \times 10^{-01} \pm 1.4 \times 10^{-02}$	$2.09 \times 10^{-01} \pm 2.0 \times 10^{-02}$	13	17	$3.25 \times 10^{-01} \pm 3.0 \times 10^{-02}$	$2.40 \times 10^{-01} \pm 2.9 \times 10^{-02}$
14	14	9	$3.90 \times 10^{-01} \pm 7.1 \times 10^{-02}$	$3.31 \times 10^{-01} \pm 3.8 \times 10^{-02}$	14	9	$3.70 \times 10^{-01} \pm 1.1 \times 10^{-01}$	$3.34 \times 10^{-01} \pm 6.6 \times 10^{-02}$
15	15	16	$2.43 \times 10^{-01} \pm 1.3 \times 10^{-02}$	$1.67 \times 10^{-01} \pm 7.0 \times 10^{-03}$	15	18	$2.55 \times 10^{-01} \pm 1.2 \times 10^{-02}$	$1.82 \times 10^{-01} \pm 2.3 \times 10^{-02}$
16	16	19	$2.09 \times 10^{-01} \pm 9.0 \times 10^{-03}$	$1.75 \times 10^{-01} \pm 5.3 \times 10^{-03}$	16	19	$1.95 \times 10^{-01} \pm 7.1 \times 10^{-03}$	$1.69 \times 10^{-01} \pm 2.1 \times 10^{-03}$
17	17	13	$3.19 \times 10^{-01} \pm 1.2 \times 10^{-02}$	$2.78 \times 10^{-01} \pm 1.5 \times 10^{-02}$	17	13	$3.51 \times 10^{-01} \pm 1.9 \times 10^{-02}$	$3.06 \times 10^{-01} \pm 2.4 \times 10^{-02}$
18	18	20	$3.05 \times 10^{-01} \pm 1.8 \times 10^{-02}$	$2.38 \times 10^{-01} \pm 1.1 \times 10^{-02}$	18	20	$3.00 \times 10^{-01} \pm 1.4 \times 10^{-02}$	$2.38 \times 10^{-01} \pm 1.5 \times 10^{-02}$
19	19	16	$2.08 \times 10^{-01} \pm 3.4 \times 10^{-03}$	$1.92 \times 10^{-01} \pm 5.6 \times 10^{-03}$	19	16	$1.98 \times 10^{-01} \pm 6.3 \times 10^{-03}$	$1.81 \times 10^{-01} \pm 8.0 \times 10^{-03}$
20	20	18	$3.44 \times 10^{-01} \pm 1.4 \times 10^{-02}$	$2.82 \times 10^{-01} \pm 1.3 \times 10^{-02}$	20	18	$3.60 \times 10^{-01} \pm 1.2 \times 10^{-02}$	$2.98 \times 10^{-01} \pm 8.8 \times 10^{-03}$

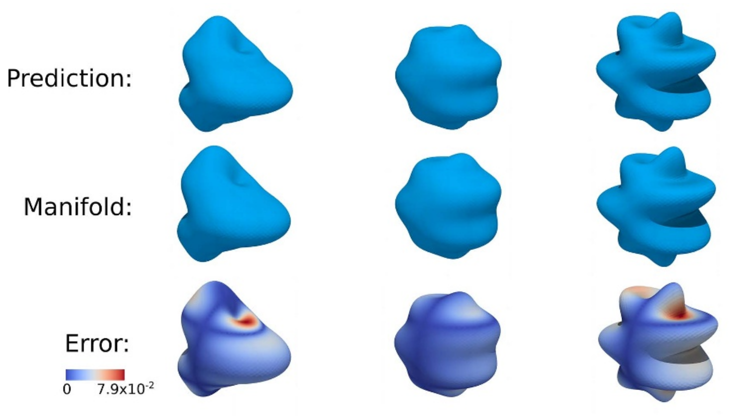

Figure 7. We show the GNP predictions based on MAP estimates for the shape along with the true manifold shape. We also show how the local absolute errors of the predictions vary over the surface of the manifold.

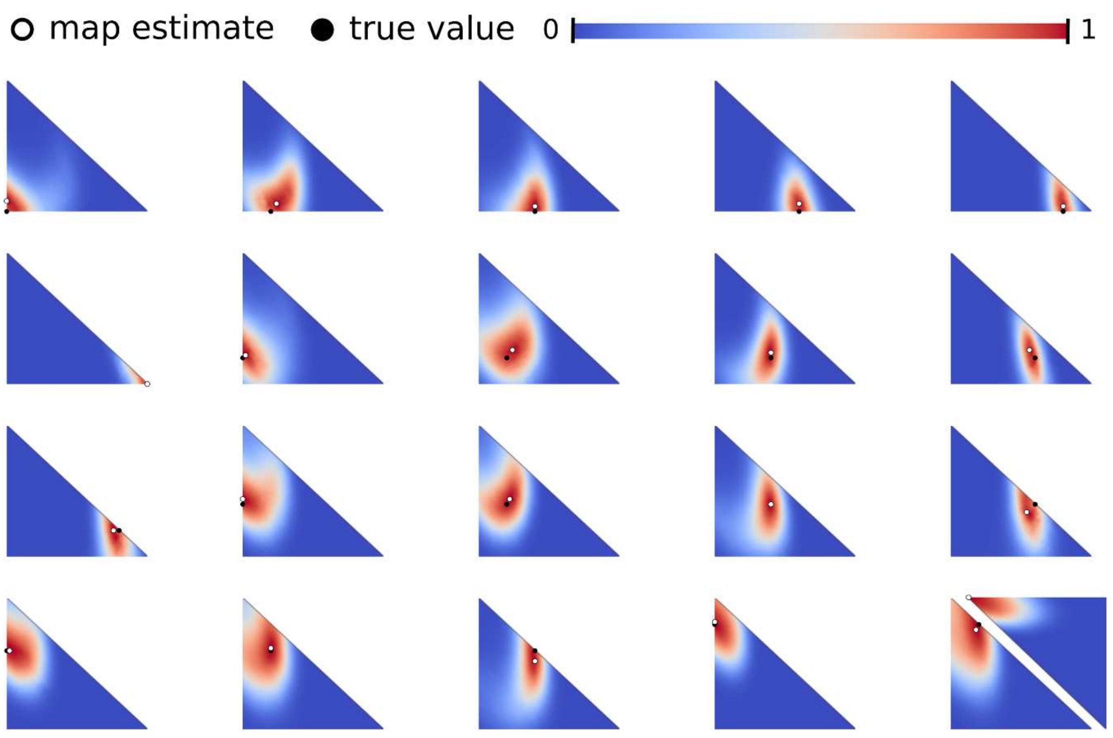

Figure 8. Manifold Shape Estimation: Solutions of the Bayesian Inverse Problem. We estimate the shape of manifolds by solving a Bayesian inverse problem based on observation of Laplace–Beltrami (LB) responses. Shown is the density of the posterior distribution over the shape space when using for the empirical data 5 samples of the LB-responses. The results are shown when the ground-truth shape has index i ranging from $i = 0, 1, 2, \ldots, 20$ with ordering top-to-bottom and left-to-right. The combined results for the cases $19, 20$ are shown on the lower right.

predictions of the manifold shape, denoted by $L1$ and $L2$. We report the mean $\pm$ the standard deviation of the results over 5 training runs. We highlight in bold the case where the first prediction disagreed with the true manifold. We find in the case of using 5 samples for the LB responses, we are able to identify shapes well. In this case, there were no errors over the 21 test shapes. We can see in some cases, such as $16, 19$ the posterior likelihoods are similar, indicating potentially similar LB responses. Interestingly, we do see in both sample cases the same two manifolds appear in the top-two predictions, including the correct manifold. We find when reducing the number of response samples to 3, one manifold is misidentified, but the case with 5 samples did not have any errors when using the GNPs to predict the shape. These results show that GNPs are capable of learning surrogate models for Bayesian inference tasks involving significant geometric contributions.

We also show the full posterior distribution in figure 8. The full posterior distribution can be used to obtain some further insights into the predictions. We find for many shapes there is a wide range in the posterior distribution for several shapes. This indicates these likely have similar responses. These results reinforce that for some predictions, even if correctly identified, there may be less overall certainty in the predicted shape. This highlights the need for enough sampling and for sufficient richness of the responses to obtain correct shape identification. The overall results show GNPs can be used within Bayesian inverse problems to capture the roles played by geometry.

6. Conclusions

We have shown how deep learning can be leveraged to develop methods for incorporating geometric contributions into data-driven learning of operators. Further, we have developed new architectures using factorizations and block-reductions to make training more efficient without degrading performance. We showed how geometric operators can be learned from manifolds including with point-cloud representations. We showed how geometric operators can be developed for estimating the metric and curvatures of surfaces, differential operators on manifolds, and solution maps for geometric PDEs. We also showed how GNPs can be combined with Bayesian inference to solve inverse problems for shape identification. In the context of linear PDEs, our methods also can be used readily to obtain more general families of solutions from the super-position principle or data-driven learning of approximations to the collection of Green's functions. The presented results provide a few ways GNPs can be used for learning tasks in non-euclidean settings where there are significant contributions from geometry.

Data availability statement

All data that support the findings of this study are included within the article (and any supplementary files).

Acknowledgments

Authors research supported by grant NSF Grant DMS-2306101. Authors also would like to acknowledge computational resources and administrative support at the UCSB Center for Scientific Computing (CSC) with Grants NSF-CNS-1725797, MRSEC: NSF-DMR-2308708, Pod-GPUs: OAC-1925717, and support from the California NanoSystems Institute (CNSI) at UCSB. P.J.A. also would like to acknowledge a hardware grant from Nvidia.

Appendix. Differential geometry of radial manifolds

A radial manifold is defined as a surface having the property that each point on the surface can be connected by a line segment to the origin only intersecting the surface at one point. This provides for a convenient parameterization $\boldsymbol{x}$ of the surface in spherical coordinates as a function of the azimuth angle θ and the polar angle ϕ with

$$\boldsymbol{x}(\theta,\phi) = \boldsymbol{\sigma}(\theta,\phi) = r(\theta,\phi)\,\boldsymbol{r}(\theta,\phi). \tag{19}$$

The r is a positive scalar function and the $\boldsymbol{r}$ is the unit vector from the origin to the point on the sphere with spherical coordinates (θ,ϕ).

In practice, when performing coordinate-based calculations at least two charts are required for radial manifolds in order to overcome singularities that occur from the topology in the surface coordinate frames, such as poles when using spherical angles. To this end, when using our numerical methods we consider two coordinate charts as in [43, 45]. The first chart we call Chart A and has coordinate singularities at the north and south pole, while the latter chart we call Chart B which has coordinate singularities at the east and west poles, see [43, 45]. In practice, for a given $(\theta,\phi) \in [0,2\pi) \times [0,\pi]$, we typically restrict usage of a chart for $\phi \in [\frac{\pi}{5}, \frac{4\pi}{5}]$. For chart A, we parameterize the manifold in the embedding space $\mathbb{R}^3$ as

$$\boldsymbol{x}\left(\hat{\theta},\hat{\phi}\right) = r\left(\hat{\theta},\hat{\phi}\right)\boldsymbol{r}\left(\hat{\theta},\hat{\phi}\right), \qquad \boldsymbol{r}\left(\hat{\theta},\hat{\phi}\right) = \left[\sin\left(\hat{\phi}\right)\cos\left(\hat{\theta}\right), \sin\left(\hat{\phi}\right)\sin\left(\hat{\theta}\right), \cos\left(\hat{\phi}\right)\right]. \tag{20}$$

For Chart B, we use

$$\boldsymbol{x}\left(\bar{\theta},\bar{\phi}\right) = r\left(\bar{\theta},\bar{\phi}\right)\boldsymbol{r}\left(\bar{\theta},\bar{\phi}\right), \qquad \boldsymbol{r}\left(\bar{\theta},\bar{\phi}\right) = \left[\cos\left(\bar{\phi}\right), \sin\left(\bar{\phi}\right)\sin\left(\bar{\theta}\right), \sin\left(\bar{\phi}\right)\cos\left(\bar{\theta}\right)\right]. \tag{21}$$

Using these parameterizations, we can compute the basis $\partial_\phi, \partial_\theta$ for the tangent space as

$$\boldsymbol{\sigma}_\phi(\theta,\phi) = r_\phi(\theta,\phi)\,\boldsymbol{r}(\theta,\phi) + r(\theta,\phi)\,\boldsymbol{r}_\phi(\theta,\phi), \tag{22}$$

$$\boldsymbol{\sigma}_\theta(\theta,\phi) = r_\theta(\theta,\phi)\,\boldsymbol{r}(\theta,\phi) + r(\theta,\phi)\,\boldsymbol{r}_\theta(\theta,\phi). \tag{23}$$

Expressions for $\boldsymbol{r}_\phi, \boldsymbol{r}_\theta$ can be found using equations (20) and (21) depending on which chart is being used. These can also be used to compute the metric and shape tensors as in equation (9) and the Gaussian curvature using equation (11). The metric tensor **I** is often denoted interchangeably with **g**, and is used to compute differential operators like the scalar LB used in section 4. It can be computed in coordinates using Einstein summation as

$$\Delta_{\mathrm{LB}} = \frac{1}{\sqrt{|g|}}\partial_i\left(g^{ij}\sqrt{g}\partial_j\right). \tag{24}$$

The g_{ij} denotes the metric tensor and g^{ij} denotes the terms of the inverse metric tensor for $i,j \in \{\phi,\theta\}$. In this way, we can parameterize our manifolds for coordinate-based calculations during training and numerical methods.

ORCID iDs

B Quackenbush https://orcid.org/0009-0005-5414-9008
P J Atzberger https://orcid.org/0000-0001-6806-8069

References

[1] Bronstein M M Bruna J Cohen T and Veličković P 2021 Geometric deep learning: grids, groups, graphs, geodesics, and gauges (arXiv:2104.13478)
[2] Izenman A J 2012 Introduction to manifold learning *Wiley Interdisciplinary Reviews: Computational Statistics* vol 4 (Wiley) pp 439–46
[3] Stuart A M 2010 Inverse problems: a Bayesian perspective *Acta Numer.* **19** 451–559
[4] Lopez R and Atzberger P J 2022 GD-VAEs: geometric dynamic variational autoencoders for learning nonlinear dynamics and dimension reductions (arXiv:2206.05183)
[5] Gross B J, Trask N, Kuberry P and Atzberger P J 2020 Meshfree methods on manifolds for hydrodynamic flows on curved surfaces: a generalized moving least-squares (GMLS) approach *J. Comput. Phys.* **409** 109340
[6] Chen T and Chen H 1995 universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems *IEEE Trans. Neural Netw.* **6** 911–7
[7] Kovachki N, Li Z, Liu B, Azizzadenesheli K, Bhattacharya K, Stuart A and Anandkumar A 2021 Neural operator: learning maps between function spaces (arXiv:2108.08481)
[8] Strauss W A 2007 *Partial Differential Equations: An Introduction* (Wiley)
[9] Pozrikidis C 1992 *Boundary Integral and Singularity Methods for Linearized Viscous Flow* (Cambridge University Press)
[10] Audouze C, De Vuyst F and Nair P B 2009 Reduced-order modeling of parameterized PDEs using time–space-parameter principal component analysis *Int. J. Numer. Methods Eng.* **80** 1025–57
[11] Kaipio J and Somersalo E 2006 *Statistical and Computational Inverse Problems* vol 160 (Springer)
[12] Gelman A, Carlin J B, Stern H S and Rubin D B 1995 *Bayesian Data Analysis* (Chapman)
[13] Asch M, Bocquet M and Nodet M 2016 *Data Assimilation: Methods, Algorithms and Applications* (SIAM)
[14] Bronstein M M, Bruna J, LeCun Y, Szlam A and Vandergheynst P 2017 Geometric deep learning: going beyond euclidean data *IEEE Signal Process. Mag.* **34** 18–42
[15] Wasserman L 2018 Topological data analysis *Annu. Rev. Stat. Appl.* **5** 501–32
[16] Hackel T Savinov N Ladicky L Wegner J D Schindler K and Pollefeys M 2017 Semantic3d. net: a new large-scale point cloud classification benchmark (arXiv:1704.03847)
[17] Lassila T and Rozza G 2010 Parametric free-form shape design with PDE models and reduced basis method *Comput. Methods Appl. Mech. Eng.* **199** 1583–92
[18] Fefferman C, Mitter S and Narayanan H 2016 Testing the manifold hypothesis *J. Am. Math. Soc.* **29** 983–1049
[19] Nguyen V P, Anitescu C, Bordas S P A and Rabczuk T 2015 Isogeometric analysis: an overview and computer implementation aspects *Math. Comput. Simul.* **117** 89–116
[20] Haasdonk B 2017 Reduced basis methods for parametrized PDEs–a tutorial introduction for stationary and instationary problems *Model Reduct. Approx.: Theory Algorithms* **15** 65
[21] Quarteroni A, Manzoni A and Negri F 2015 *Reduced Basis Methods for Partial Differential Equations: An Introduction* vol 92 (Springer)
[22] Lassila T, Manzoni A, Quarteroni A and Rozza G 2013 Generalized reduced basis methods and n-width estimates for the approximation of the solution manifold of parametric PDEs *Analysis and Numerics of Partial differential Equations* (Springer) pp 307–29
[23] Bhattacharya K, Hosseini B, Kovachki N B and Stuart A M 2021 Model reduction and neural networks for parametric PDEs *The SMAI J. Comput. Math.* vol 7 (Société de Mathématiques Appliquées et Industrielles) (available at: http://www.numdam.org/articles/10.5802/smai-jcm.74/) pp 121–57
[24] Hirsch M W, Smale S and Devaney R L 2012 *Differential Equations, Dynamical Systems and An Introduction to Chaos* (Academic)
[25] Arnold V I, Afrajmovich V S, Il'yashenko Y S and Shil'nikov L P 2013 *Dynamical Systems V: Bifurcation Theory and Catastrophe Theory* vol 5 (Springer)
[26] Meilă M and Zhang H 2023 Manifold learning: what, how and why *Annu. Rev. Stat. Appl.* **11** 393-417
[27] Kirchdoerfer T and Ortiz M 2016 Data-driven computational mechanics *Comput. Methods Appl. Mech. Eng.* **304** 81–101
[28] Chui C K and Mhaskar H N 2018 Deep nets for local manifold learning *Fron. Appl. Math. Stat.* **4** 12

[29] Li Z, Kovachki N, Choy C, Li B, Kossaifi J, Otta S, Nabian M A, Stadler M, Hundt C and Azizzadenesheli K 2024 Geometry-informed neural operator for large-scale 3d pdes *Advances in Neural Information Processing Systems* vol 36
[30] Lu L, Jin P, Pang G, Zhang Z and Karniadakis G E 2021 Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators *Nat. Mach. Intell.* **3** 218–29
[31] Kovachki N, Li Z, Liu B, Azizzadenesheli K, Bhattacharya K, Stuart A and Anandkumar A 2023 Neural operator: learning maps between function spaces with applications to PDEs *J. Mach. Learn. Res.* **24** 97
[32] Li Z Kovachki N Azizzadenesheli K Liu B Bhattacharya K Stuart A and Anandkumar A 2020 Fourier neural operator for parametric partial differential equations (arXiv:2010.08895)
[33] O'Leary-Roseberry T Chen P Villa U and Ghattas O 2022 Derivate informed neural operator: an efficient framework for high-dimensional parametric derivative learning (arXiv:2206.10745)
[34] Chatterjee A 2000 An introduction to the proper orthogonal decomposition *Curr. Sci.* **78** 808–17
[35] Schmidt O T and Colonius T 2020 Guide to spectral proper orthogonal decomposition *AIAA J.* **58** 1023–33
[36] Schmid P J 2010 Dynamic mode decomposition of numerical and experimental data *J. Fluid Mech.* **656** 5–28
[37] Kutz J N, Brunton S L, Brunton B W and Proctor J L 2016 *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems* (SIAM)
[38] Schmid P J 2022 Dynamic mode decomposition and its variants *Annu. Rev. Fluid Mech.* **54** 225–54
[39] Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A and Anandkumar A 2021 Fourier neural operator for parametric partial differential equations (arXiv:2010.08895)
[40] Anandkumar A, Azizzadenesheli K, Bhattacharya K, Kovachki N, Li Z, Liu B and Stuart A 2019 Neural Operator: graph kernel network for partial differential equations *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations* (available at: https://openreview.net/forum?id = fg2ZFmXFO3)
[41] Hornik K 1991 Approximation capabilities of multilayer feedforward networks *Neural Netw.* **4.2** 251–7
[42] Guibas J, Mardani M, Li Z, Tao A, Anandkumar A and Catanzaro B 2021 Adaptive fourier neural operators: efficient token mixers for transformers (arXiv:2111.13587)
[43] Gross B J and Atzberger P J 2018 Hydrodynamic flows on curved surfaces: spectral numerical methods for radial manifold shapes *J. Comput. Phys.* **371** 663–89
[44] Pressley A N 2010 *Elementary Differential Geometry* (Springer)
[45] Gross B and Atzberger P J 2018 Spectral numerical exterior calculus methods for differential equations on radial manifolds *J. Sci. Comput.* **76** 145–65
[46] Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)
[47] Lebedev V I 1976 Quadratures on a sphere *USSR Comput. Math. Math. Phys.* **16** 10–24
[48] Loshchilov I and Hutter F 2016 Sgdr: Stochastic gradient descent with warm restarts (arXiv:1608.03983)