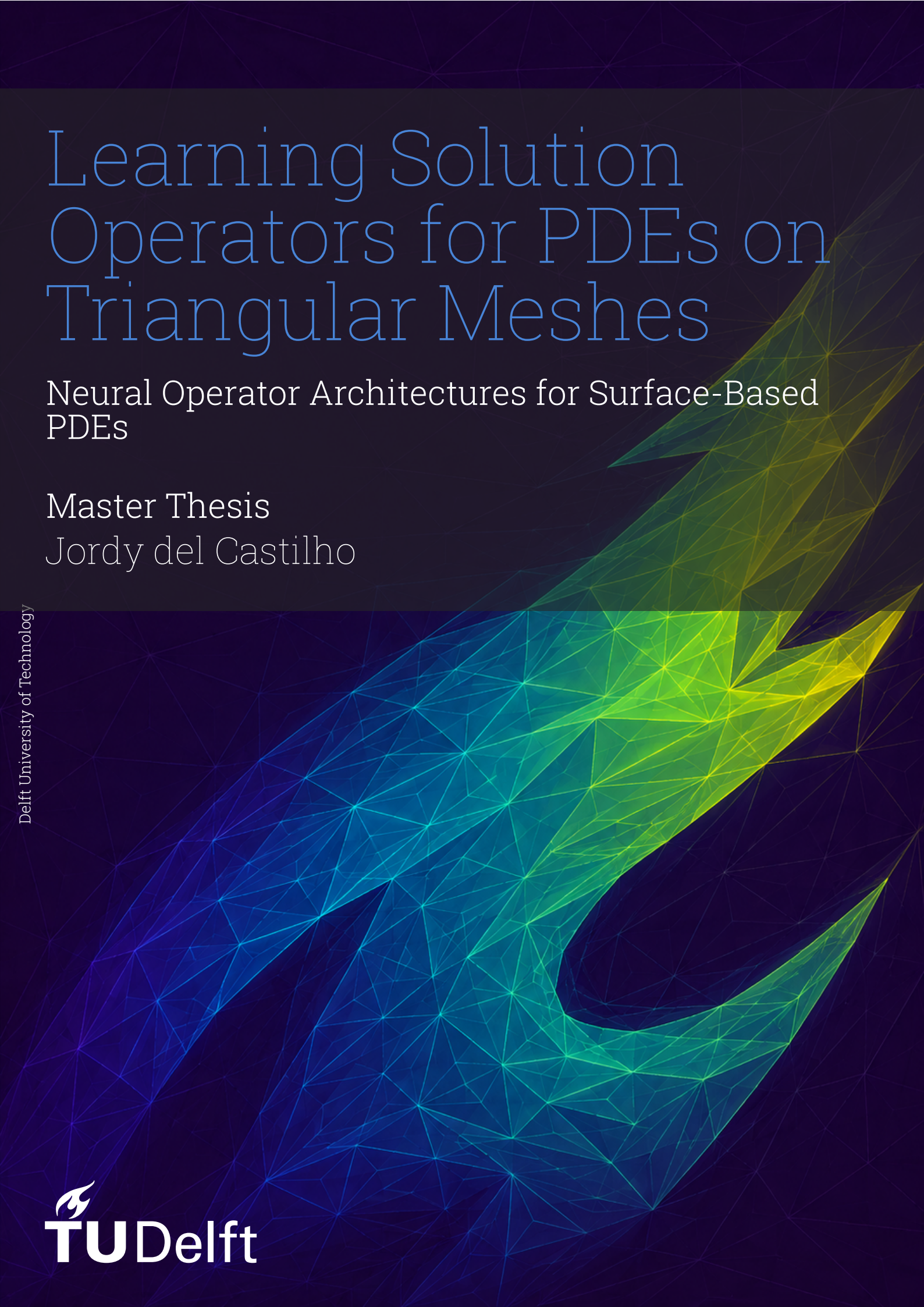
Learning Solution Operators for PDEs on Triangular Meshes

Neural Operator Architectures for Surface-Based PDEs

Master Thesis

Jordy del Castilho

Delft University of Technology



TU Delft

Learning Solution Operators for PDEs on Triangular Meshes

Neural Operator Architectures for Surface-Based PDEs

by

Jordy del Castilho

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday June 29, 2026 at 12:00 PM.

Student number: 5550394
Project duration: November 25, 2025 – June 29, 2026
Thesis committee:

Klaus Hildebrandt	Chair, Associate Professor
Jan van Gemert	Core Member, Associate Professor
Jackson Campolattaro	Additional member, PhD candidate

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.
Code will be available at `https://github.com/jofeltje2/TU-delft-msc-thesis/tree/main`.

TU Delft

Preface

This thesis was written as part of the Computer Science master's programme at TU Delft. My interest in computer graphics began during the computer graphics course in the Computer Science bachelor's programme. During the master's introduction day in the final year of the bachelor's programme, we were told that master's students choose two themes. We were also told that the computer graphics and machine learning themes complement each other well. Fortunately, these were also the two subjects that interested me most.

During the master's programme, I followed the computer graphics research course, where I worked on a project related to vector graphics. This project further increased my interest in the field, and I initially proposed a vector-graphics-related topic for my master's thesis. During a meeting with my supervisor, however, I was introduced to several other possible projects. One of these involved using neural networks to approximate solutions of partial differential equations, which are mathematical models commonly used to describe physical phenomena. This also strongly appealed to me.

At the start of the thesis, I was very motivated to define an ambitious research direction. However, as I had also experienced during my bachelor's thesis, a good research idea is rarely as simple to realise as it first appears. After several changes in scope and valuable feedback from my supervisor, the project was narrowed down to neural operators for solving PDEs on fixed triangular mesh surfaces. Throughout the project, I was reminded several times that concise explanations of difficult concepts can make the underlying theory appear easier than it actually is. Looking back, I feel that I have learnt a great deal about neural operators, artificial intelligence architectures, mathematics, and physics, especially partial differential equations. At the same time, I still consider myself a novice in this field, and I hope that more experienced researchers can further extend this work.

I would like to thank my parents, who have supported me throughout the thesis project. They gave up some quality time together so that I could work on my thesis, and they encouraged me to take the much-needed two-week delay before my green-light review, which helped reduce some of the stress.

Finally, I would like to thank my supervisors, Klaus and Jackson. They helped me refine the scope of the project, explained difficult concepts, shared reference code and papers, and gave valuable advice throughout the thesis process. Without their guidance, I would likely have chosen a topic that was too broad or too complex to complete within the available time.

Jordy del Castilho
Delft, June 2026

Abstract

Partial differential equations are widely used to model physical and geometrical behaviour. Real-time graphics and interactive simulation applications often require fast approximations of these equations. Traditional solvers often require solving large linear systems, which can form a bottleneck for real-time applications. Neural operators aim to learn the direct mapping between input and solution functions. This thesis investigates whether neural operator architectures can learn solution operators for partial differential equations on fixed triangular mesh surfaces. Several model families are compared, including a standard Multilayer Perceptron, DeepONet, a Graph Neural Network, a Multigrid Graph Neural Network, a Spectral Graph Neural Network, and a Hodge Spectral Graph Neural Network. These models are trained and evaluated on three surface-based PDE tasks: the Poisson equation with different scalar right-hand sides, geodesic distance approximation using the heat method with a varying number of source points, and linear elasticity using a Reissner–Mindlin thin shell with different vector-valued force fields. The predictions are evaluated using relative L_2 loss, isoline comparisons for scalar fields, error visualisations for displacement fields, and inference speed measurements. The results show that different PDEs favour different architectures. Spectral models perform especially well on the Poisson problem, particularly for smooth right-hand sides. Geodesic distance approximation is harder to learn, although the Hodge Spectral Graph Neural Network performs best overall for this task. The linear elasticity experiments show a different trend: for smoother force fields, the standard Multilayer Perceptron is highly competitive and often achieves the lowest loss, while localised force fields remain more difficult to approximate. This work also shows that neural operators can achieve significant speed-ups over traditional solvers, especially for more complex problems and larger meshes, suggesting a valid use case for these models in real-time applications.

Contents

List of Figures

List of Tables

1 Introduction

1.1. Motivation

Partial differential equations (PDEs) are widely used to model real-world physical processes. These equations are essential for modelling processes such as fluid dynamics [1], deformation [2], electromagnetism [3], heat diffusion [4], and geodesics [5]. In computer graphics and geometry processing, these PDEs are often defined on triangular meshes and model physical behaviour along the mesh surface (2D) or through the mesh volume (3D).

Numerical methods, such as the Finite Element Method (FEM) and Finite Difference Method (FDM), are reliable ways to solve these problems. These methods often require solving a linear system to obtain the solution. Sometimes a single solve is enough, but in cases where many repeated solves with different input conditions are needed, these methods can form a bottleneck. For example, in fluid dynamics, storm trajectories can be predicted; however, it is necessary to perform many repeated solves with slightly different input conditions to ensure that, across all scenarios, a storm will not hit a city. This could require on the order of tens of thousands of solves [6].

Neural operators offer a possible solution to this computational bottleneck. By training a model on many input and output scenarios, the model can learn to map unseen input conditions to an accurate approximation of the solution.

Neural operators extend the field of neural networks by learning mappings between function spaces rather than finite-dimensional vectors. In the context of PDEs, this means learning the mapping between a forcing function or initial state and the solution function or final state. For example, in the context of elasticity, a trained neural operator can rapidly predict the displacement field of a mesh surface based on an initial force field.

A neural operator requires correct training data to be able to learn such mappings. Training data still has to be obtained through numerical methods. Because of this, neural operators can be seen as an investment for problems that are known to require many fast solves. Initial costs to generate training data may need to be on the order of thousands of solves. However, once trained, neural operators can produce fast approximate solutions, leading to a better amortised cost when many repeated evaluations are required.

One motivating application area is real-time computer graphics. Games and interactive simulations often require fast approximations of physical and geometrical processes. Solving one or multiple PDEs in every frame of a game or simulation can form a significant performance bottleneck. With a neural operator, these solves can be approximated quickly while remaining accurate. Even though this requires generating a lot of training data and training the models, this can all be done offline. For the end user, the only cost associated with increased performance would be storing the model parameters.

1.2. Problem

This thesis studies whether neural networks can learn solution operators for PDEs defined on surfaces of fixed triangular meshes. Given a fixed mesh and a forcing function defined on the mesh's vertices, the goal is to predict the corresponding solution function to the PDE problem. This should be done without running a classical PDE solver for each new input.

The fixed-mesh setting is central to this work. This research does not consider training a model that generalises between arbitrary meshes, but rather trains and evaluates a model for a specific problem on a fixed mesh. In games and interactive simulations, game objects often remain fixed but may require many repeated solves with high accuracy.

1.3. Research gap

Neural operators have shown very promising results for learning PDE solution operators in many different domains. Many successful methods, such as the Fourier Neural Operator (FNO) [7], perform well on structured equidistant grids. However, FNO and other spectral methods make use of spectral transforms, which can only be performed efficiently on these structured grids. Many geometric domains are not naturally represented by structured grids: triangular meshes are unstructured representations of irregular domains, and their geometry influences the physical behaviour of PDE solutions. This makes it less clear which methods are suitable for learning PDE solution operators on triangular meshes.

Recent work has introduced new model architectures that perform well on unstructured grids and irregular domains, utilising graph-based or transformer-based models. However, research on unstructured meshes and grids often aims to learn a mesh-agnostic mapping that can generalise across complex geometries. While this is an interesting goal, building mesh-agnostic models could introduce an accuracy cost. This research proposes to consider only fixed meshes. In a fixed-mesh setting, the model can fully specialise to a single geometry, and simpler models that have a fixed input size can also be considered.

This thesis therefore seeks to determine how simpler models compete with more complex models in a fixed-mesh setting, not only in terms of accuracy but also with regard to model size and inference speed.

1.4. Research questions

This thesis attempts to answer the following research questions:

1. **How accurately can neural operator models learn solution operators for PDEs on surfaces of fixed triangular meshes?**
   Using isolines, the accuracy of an average prediction made by the model can be compared to the ground truth. This is to see whether these models can actually make accurate predictions rather than simply achieve a low loss value.

2. **How do different models compare in terms of accuracy?**
   More specifically, this work analyses differences in accuracy between different kinds of model implementations. This includes MLP-based models like the standard MLP and DeepONet, spectral-based models like the Spectral Graph Neural Network and the Hodge Spectral Graph Neural Network, and graph-based models like the Multigrid Graph Neural Network and the Graph Neural Network.

3. **Which models provide the best trade-off between accuracy, inference speed, and model size for real-time applications?**
   This work compares speed-ups and model sizes to argue whether these models can be suitable as neural surrogates that can optimise real-time applications such as games.

1.5. Contributions

This thesis makes the following contributions to the study of neural operators for PDEs on triangular surface meshes. First, it defines a complete experimental framework for data generation, model initialisation, model training, and evaluation. This framework makes it possible to reproduce the results of this thesis and to extend the experiments to additional PDEs, meshes, right-hand sides, and model architectures.

Second, this work evaluates several neural operator models on multiple surface-based PDE tasks. The Poisson problem is evaluated on polynomial, delta, and sinusoidal right-hand side datasets. Geodesic distance approximation is evaluated using the heat method with different numbers of source points. In addition, this thesis evaluates surface linear elasticity, where the input is a vector-valued force field and

the output is the resulting displacement field. For the elasticity experiments, several force initialisations are considered, including normal delta forces, hydraulic press forces, sinusoidal forces, and smooth normal bumps.

Third, this thesis compares different types of neural operator implementations in a fixed-mesh setting. The evaluated models include a standard MLP, DeepONet, graph-based message-passing models, a Multigrid Graph Neural Network, and spectral graph neural networks. This comparison shows how the suitability of an architecture depends not only on the PDE itself, but also on the structure of the input function and the smoothness or locality of the resulting solution field.

Lastly, this thesis compares the inference speed of the trained models to traditional solvers and provides visualisations of the resulting predictions. This makes it possible to evaluate the models not only in terms of accuracy, but also in terms of their usefulness as surrogate solvers for real-time applications.

The results show that certain model architectures can learn the Poisson problem on triangular surface meshes accurately, especially for smooth right-hand sides. However, more localised right-hand sides, such as the delta right-hand side, are significantly harder to learn. The geodesic experiments show that more complex solution operators, such as geodesic distance approximation using the heat method, are harder for these models and likely require further architectural improvements.

The linear elasticity experiments show a different trend. For smoother force initialisations, such as hydraulic press loads, sinusoidal forces, and smooth normal bumps, the MLP baseline is highly competitive and achieves the lowest loss in several settings. For the more localised normal delta force, the behaviour is different, suggesting that localised elastic loads are harder to learn and may benefit more from architectures that encode global or spectral information. Overall, these results show that there is no single best architecture for all PDE tasks. Instead, model performance depends strongly on the PDE, the type of input field, and the structure of the resulting solution.

Finally, this thesis shows that neural operators can provide significant inference speed-up advantages over traditional solvers, especially when many repeated solves are required on larger meshes. This supports their potential use in real-time applications, such as games and interactive simulations, where approximate PDE solutions must be computed quickly.

2

Related Work

The research questions outlined in Chapter 1 require multiple neural operator architectures to be considered and compared. This chapter explains the motivation behind the selected architectures for the fixed triangular mesh setting.

2.1. Neural operators

Neural operators are architectures that enable learning mappings between function spaces. They are mainly used in the field of PDEs, where they learn the mapping between an input function and a solution function. This distinguishes neural operators from standard neural networks, which learn mappings between finite-dimensional vectors.

One of the foundational architectures in operator learning is the DeepONet architecture [8]. DeepONets approximate operators using two subnetworks: a branch network, which encodes the input function on the domain using a set of sensor points, and a trunk network, which encodes the query location. The final output is obtained by combining the outputs of the two subnetworks. This model works pointwise, predicting the value of the solution function at each query point while taking the whole input function and the three-dimensional coordinates of the query point as input. In this way, the branch network can specialise in function encoding, and the trunk network can specialise in query locations. Apart from being designed to follow the universal approximation theorem for operators [8, 9], this architecture also reflects a key property of the output: it is a function that can be queried at any point. However, this architecture still requires fixed sensor points during training and is therefore not considered discretisation-invariant. In a fixed triangular mesh setting, this model can still be utilised if it manages to implicitly learn the geometry of the mesh.

Other neural operator architectures often follow a kernel approach. In this approach, the input function is first lifted to a higher-dimensional feature space. Then, an integration over a learnt kernel is applied multiple times to the current latent representation. Lastly, the latent representation is projected back to the original function space.

This kernel-based approach is used in different neural operator architectures, each defining the integration over the kernel in a different way. This closely follows the Green's function interpretation of differential operators. For linear differential operators, the Green's function is the solution corresponding to a Dirac delta forcing at a specific point. Convolving the Green's function with a forcing function yields the corresponding solution. Because of this, neural operators that can approximate such global convolutions/integrations can learn the solution operator for linear differential operators. By including non-linear activation functions between these kernels, this can be extended beyond linear operators. For neural operator architecture research, it is essential to find accurate and inexpensive ways to compute this convolution.

2.2. Regular domains

A direct discretisation of a global integral kernel can be written as a sum over all pairs of discretisation points, which leads to $O(N^2)$ complexity. This motivates neural operator architectures that compute global interactions more efficiently, such as the Fourier Neural Operator.

2.2.1. Fourier Neural Operator

One of the most cited and influential works in the field of neural operators, the Fourier Neural Operator (FNO) [7], uses the Fourier domain. Global convolution/integration can be obtained via multiplication with the kernel in the Fourier domain. By restricting to a regular equidistant grid, the Fast Fourier Transform (FFT) can be utilised with a time complexity of $O(N \log(N))$. Because of this improvement in time complexity, while still being able to perform global convolutions, PDE problems can be solved orders of magnitude faster using a trained FNO architecture.

2.2.2. Multiscale extensions

The FNO architecture excels at capturing global relations but is less effective at capturing local relations. The Wavelet Neural Operator (WNO) [10] uses wavelet transforms instead of Fourier transforms, allowing the model to represent information at multiple scales and therefore capture both global and local structure. Despite being more expressive, it is still limited to regular, equidistant grids for fast spectral transforms.

Another way to extend FNO to handle local relations is to include integral and differential kernels alongside the spectral convolution and residual connection. This is exactly what is done by Liu-Schiaffini, Berner, and Bonev [11]. Using a small CNN as the differential kernel and a small-domain Riemann sum approximation for the local integration kernel, the model can learn local relations more easily than with only global convolutions in the spectral domain.

2.2.3. Domain extensions

All these spectral methods are limited to equidistant grids and cannot be used directly for irregular domains such as triangular meshes. However, Spherical FNO [12] extends the grid-based FNO to spheres. This is done by projecting the input function onto the eigenvectors of the Laplace operator of a spherical mesh. This is similar to a spectral transform, meaning that multiplying this projected representation by a kernel forms global convolutions. This is essential for the fixed-mesh setting described in this thesis.

2.3. Irregular domains and meshes

The methods discussed so far are mainly designed for regular and structured domains. However, many real-world problems are defined on irregular domains. This motivates research into neural operator architectures that work on arbitrary triangular meshes. Methods such as Geo-FNO and GINO extend neural operators towards more general geometries and large-scale PDE problems on irregular domains [13, 14]. However, these methods are typically motivated by generalisation across geometries or by handling complex 3D domains, whereas this thesis focuses on the fixed triangular mesh setting.

2.3.1. Graph Neural Operators

Graph Neural Operators [15] are one of the first proposed solutions for irregular domains. Graph Neural Operators utilise graph neural networks and message passing to capture local information propagation for PDEs. Despite the graph neural operator being a kernel-based neural operator, it can mostly only capture local relations due to the locality induced by message passing. Still, this architecture shows promising results, especially for time-dependent PDEs where the model predicts the next time step. It has also shown the ability to handle non-regular grids and even point-cloud data. However, there has not been much published work that demonstrates this model's capabilities on mesh surfaces. In fact, the Graph Neural Operator was designed to allow message passing based on Euclidean distance, not geodesic distance. This could mean that information is spread throughout the mesh volume rather than along the surface. This limitation is especially relevant for thin surface geometries, where two vertices can be close in Euclidean space but far apart along the surface. A similar issue is discussed by Pang et al. [16] who show that Euclidean grid-based pooling can incorrectly merge vertices on opposite sides of a tabletop, even though they are distant in geodesic distance.

Graph neural networks have also been used in combination with other models to support arbitrary input sizes and more geometric encodings of the function spaces. Graph-Structured Physics-Informed DeepONet (GS-PI-DeepONet) [17] uses a DeepONet architecture where the branch and trunk networks are graph neural networks. Instead of encoding the input function with an MLP, the graph structure allows the model to incorporate information about the connectivity of the discretised domain. This is

especially relevant for surface PDEs, where the relationship between neighbouring vertices depends on the mesh structure. To ensure greater expressivity, the authors use attention-based message-passing layers. Although this architecture is not directly evaluated in this thesis, it is still relevant as it demonstrates that DeepONet architectures can be enhanced with geometry-aware subnetworks.

2.3.2. Spectral methods on graphs and meshes

The spectral methods discussed for regular domains rely on Fourier or wavelet transforms, which are naturally defined on equidistant grids. For graphs and meshes, an analogous spectral representation can be obtained through the eigenvectors of a mesh Laplacian. Using the same principles as FNO and WNO, the function space can be projected to a spectral space in which global convolutions can be performed efficiently. This is particularly relevant for fixed triangular meshes, as the eigenbasis only needs to be computed once. A learnt model can then operate on functions defined on that fixed mesh using spectral coefficients, avoiding the need to repeatedly approximate global interactions through many local graph convolution layers.

Spatio-Spectral Graph Neural Operator (Sp2GNO) [18] is an example of an architecture that explicitly combines both perspectives. It uses a spectral graph component to model global interactions and a spatial graph component to model local relations. The spectral component uses eigenvalues for global relations, while the spatial component adapts a graph neural network to model non-global relations.

3

Background

In this chapter, the reader will gain the required background knowledge to understand the rest of the paper. Furthermore, it serves as a foundation for the methods and choices made in this research. Although the chapter is written to be understood without a background in differential equations and machine learning, it does assume the reader has a computer science background. The chapter covers an explanation of differential equations, specific forms of differential equations and ways to solve them, and lastly, machine learning/neural operator architectures.

3.1. Differential Equations

A differential equation is an equation for an unknown function that contains derivatives of that function. The derivatives describe how the unknown function varies with respect to one or more independent variables.

Differential equations are widely used to model physical and geometric phenomena. In general, a differential equation can be written in the abstract form

$$\mathcal{L}u = f,$$

where $\mathcal{L}$ is a differential operator, u is the unknown solution function, and f is prescribed data, often called the right-hand side or forcing function. Differential equations can be categorised into two kinds: Ordinary differential equations (ODEs) involve derivatives with respect to a single independent variable, whereas partial differential equations (PDEs) involve partial derivatives with respect to multiple independent variables.

A standard example of an ordinary differential equation is the predator–prey model [19], which studies the interaction between a prey population and a predator population. It can be written as

$$\frac{dr}{dt} = r - 0.01rp, \qquad \frac{dp}{dt} = -0.5p + 0.0005rp.$$

Here, $r(t)$ and $p(t)$ denote the rabbit and predator populations at time t, respectively. The operator $\frac{d}{dt}$ acts on the unknown functions $r(t)$ and $p(t)$. Solving the system means finding functions r and p that satisfy both equations. In the predator–prey problem, the rate of change of the rabbit population depends on the number of rabbits and the number of predators. PDEs are suitable for modelling quantities that vary over spatial domains, such as temperature, displacement, or distance fields.

In this thesis, these quantities are defined on triangular surface meshes.

The continuous fields are represented by values at mesh vertices, and numerical solvers are used to compute the corresponding ground-truth solution fields.

The following sections introduce the specific PDE problems considered in this work.

3.2. Poisson problem

The Poisson equation is a partial differential equation. This equation is commonly written as $\Delta u = f$ or $\nabla^2 u = f$, where Δ and ∇^2 both represent the Laplacian differential operator. The Laplacian of a function

is the divergence of its gradient. Informally, the Laplacian measures how a function value differs from its local neighbourhood; for example, whether a point is locally higher or lower than the surrounding values.

The Poisson equation is an elliptic partial differential equation. Elliptic differential equations model steady states; they are not time-dependent. Solving a Poisson problem means finding a scalar field whose Laplacian matches the prescribed forcing function, subject to appropriate constraints or boundary conditions.

For a fixed triangular mesh, the Poisson equation can be discretised using a cotangent finite-element discretisation, after which the resulting sparse linear system is solved. In the common cotangent FEM discretisation, this system is built from the lumped mass matrix and the cotangent stiffness matrix (often referred to as the cotangent Laplacian). The mass matrix assigns each vertex an associated area or mass. Vertices surrounded by larger adjacent triangles therefore receive a larger weight in the discretised system.

A Poisson problem needs an additional constraint to make the solution unique. Without such a constraint, adding the same constant value to every vertex results in another equally valid solution, so there are infinitely many possible solutions that differ only by a constant offset. In this thesis, this ambiguity is removed by projecting out the constant component of the solution. This is done by subtracting the mean value, so that each Poisson solution has zero mean.

The Poisson equation appears in many areas of physics and geometric processing. This work explores the ability to learn the solution operator for a Poisson problem on a fixed triangle mesh.

3.3. Geodesics with heat method

Another problem considered in this work is the computation of geodesics on a mesh surface. Geodesic distance fields are scalar functions defined over a surface that describe, at each point, the distance along the surface to a source point. In contrast to Euclidean distances in 3D space, geodesic distances are solely based on the smallest distance that needs to be travelled along the mesh surface from the source point to the evaluated point.

Exact geodesics are expensive to compute. For that reason, accurate approximations have been used. One such approximation is the heat method. The heat method provides an efficient and accurate approximation of geodesic distance on meshes [5].

The pipeline used in this thesis is shown in Figure 3.1. First, the source set is encoded as a discrete source signal. A short heat diffusion step is then solved on the mesh. The gradient of the diffused heat field gives a direction field pointing away from the sources after normalisation. Finally, a Poisson problem is solved to recover a scalar distance field whose gradient is consistent with this direction field.

This thesis considers learning this heat-method approximation to further reduce the inference cost of geodesic-distance approximation.

Instead of computing geodesic distances based on one source point, it is also possible to introduce multiple source points. In this thesis, multiple experiments with different numbers of source points are considered.

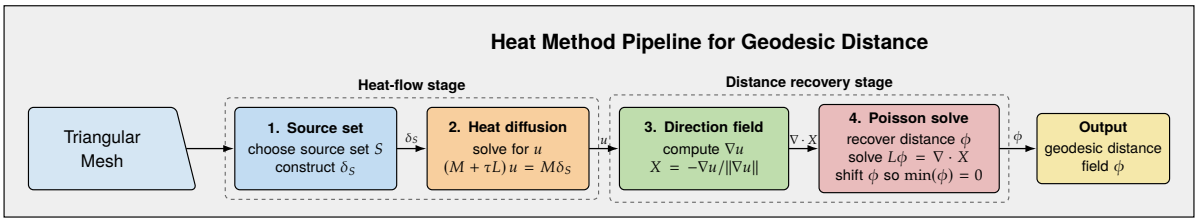


Figure 3.1: Pipeline of the heat method for approximating geodesic distance on a triangular mesh. A source set is encoded as a discrete source signal δ_S and diffused over the mesh. The normalised heat gradient defines a direction field, and a Poisson solve recovers the geodesic distance field ϕ.

3.4. Linear elasticity

Linear elasticity models how an object deforms when forces are applied to it. In general, deformation can be temporary or permanent. In the elastic range, the object returns to its original shape after the

applied force is removed. If the applied load exceeds this range, the material may deform permanently; this behaviour is known as plastic deformation. More complex elasticity models can also describe non-linear behaviour, where the relationship between force and deformation is no longer proportional.

This thesis uses linear elasticity, where the material is assumed to remain in the elastic range and the resulting displacement is assumed to depend linearly on the applied force. This means that increasing the applied force increases the resulting displacement proportionally, and that different force contributions can be combined by addition. This thesis therefore focuses on reversible, proportional deformations and does not consider non-linear effects such as plasticity. The static linear elasticity problem considered here is also an elliptic boundary-value problem after suitable constraints or boundary conditions are imposed.

For a triangular mesh with N vertices, the displacement field is represented by one three-dimensional displacement vector per vertex:

$$u_i = (u_i^x, u_i^y, u_i^z), \qquad i = 1, \ldots, N.$$

Here, u_i is the displacement vector at vertex i. The three components of $\mathbf{u}_i$ describe how that vertex moves in the x, y, and z directions. In standard volume elasticity, the object is represented as a three-dimensional solid, and the displacement field is defined throughout the volume. This thesis, however, focuses on PDEs defined on triangular surface meshes. For this reason, the elasticity problem is also formulated as a surface-based problem: the unknown field is defined on the mesh surface, while each displacement vector still describes motion in three-dimensional space. After discretisation, both the input force field and the output displacement field are stored as one three-dimensional vector per mesh vertex.

Under the assumptions of linear elasticity, the problem can be written in the abstract form

$$\mathcal{L}u = f$$

where f is the applied force field and u is the resulting displacement field. In the discrete setting used in this thesis, this corresponds to a linear system of the form

$$Ku = f$$

where K is the stiffness matrix determined by the mesh geometry, the chosen elasticity model, and the material parameters. The neural operator task is therefore to learn the mapping

$$f \mapsto u$$

from force fields to displacement fields.

Surface elasticity can be modelled in several ways. A membrane model mainly captures in-plane stretching and shearing of the surface. A shell model extends this by also accounting for bending behaviour, which is important when forces cause the surface to move out of its tangent plane. In this thesis, the mesh is treated as the mid-surface of a thin shell, and a Reissner-Mindlin shell formulation is used to compute the deformation. This formulation is suitable for thin structures because it includes both membrane effects and bending behaviour.

The behaviour of the shell is controlled by material parameters. The Young's modulus E describes the stiffness of the material within the elastic range. For the same mesh and applied force field, a larger Young's modulus produces smaller displacements, while a smaller Young's modulus produces larger displacements. The Poisson ratio ν describes how much the material contracts or expands in directions perpendicular to the applied deformation. The shell thickness h influences the resistance to bending; thicker shells generally resist bending more strongly than thinner shells.

As with other static PDE problems, an elasticity problem requires boundary conditions or equivalent constraints to obtain a unique solution. Without such constraints, the stiffness matrix contains rigid-body modes: the object can translate or rotate without strain, so the static displacement solution is not unique. In practice, this issue is handled by applying boundary conditions or by fixing certain parts of the mesh. The experiments do not focus on modelling physical boundary conditions; instead, the solver setup applies the constraints required for a well-posed discrete system. This ensures that the computed displacement field represents actual elastic deformation rather than arbitrary translation or rotation of the entire object.

In this work, the Reissner–Mindlin thin-shell formulation used by the elasticity solver is used to generate ground-truth data for operator learning. Each input sample is a vector-valued force field defined on the mesh vertices, and each target output is the corresponding vector-valued displacement field. This makes linear elasticity more complex than the scalar Poisson and geodesic experiments, as both the input and output contain three channels per vertex.

3.5. Neural operators

Neural operators extend the idea of neural networks from learning functions to learning operators. A standard neural network usually learns a mapping between finite-dimensional vectors. For example, it may learn a function

$$F : \mathbb{R}^m \to \mathbb{R}^n,$$

where the input and output are represented by a fixed number of values. An operator, on the other hand, maps one function to another function. In the context of PDEs, this is a natural point of view: the input to a PDE problem is often a function, such as a forcing function or initial condition, and the output is the PDE solution function. A PDE solution operator can therefore be written as

$$\mathcal{G} : a \mapsto u,$$

where a is the input function and u is the corresponding solution function.

3.5.1. Discretisation invariance

A continuous function can be evaluated at infinitely many points, while a computer can only store finitely many values. In practice, functions are therefore represented through a discretisation. On a regular grid, this means storing function values at grid points. On a triangular mesh, this means storing scalar or vector values at mesh vertices. The sampled values are only a finite representation of the underlying function.

A central idea in neural operator learning is that the model should learn the mapping between the underlying functions, not only between one fixed set of sampled values. This property is usually called discretisation invariance. Informally, a discretisation-invariant model should represent the same underlying operator even when the input and output functions are sampled at different resolutions. If the discretisation is refined, the model output should converge to the same continuous solution operator. This separates neural operators from ordinary neural networks that only learn a mapping between fixed-size vectors.

Discretisation invariance is especially useful for PDEs because numerical solutions are often computed at different resolutions. A model that only works for one fixed grid or mesh has learnt a useful surrogate for that discretisation, but it has not necessarily learnt the continuous operator. In contrast, a true neural operator should be able to process functions sampled on different discretisations of the same domain, or at least be formulated in a way that is independent of the specific number of sample points.

Many neural operator architectures follow the same general structure. First, the input function is lifted to a higher-dimensional latent representation. Instead of working only with the raw input value at each point, which may be scalar-valued or vector-valued, the model lifts each point to a higher-dimensional feature representation. Then, several operator layers update this latent function. Different neural operator architectures approximate these interactions in different ways, for example through learnt integral kernels, Fourier or graph-spectral representations, or local message passing on a graph. Finally, a projection maps the latent representation back to the desired output function.

This structure can be written abstractly as

$$a \xrightarrow{\text{lifting}} v_0 \xrightarrow{\text{operator layers}} v_T \xrightarrow{\text{projection}} u.$$

The lifting and projection steps are often implemented with small neural networks, while the operator layers are the part that gives the architecture its operator-learning behaviour. In kernel-based neural operators, these layers can be interpreted as learnt integral operators. This is closely related to the way many PDE solution operators can be described: the solution at one point may depend on the input function over a large part of the domain.

Another important theoretical property is universal approximation. Standard neural networks have universal approximation results for functions between finite-dimensional spaces. Neural operators have analogous results for operators between function spaces. These results state, under suitable assumptions, that certain neural operator architectures can approximate a large class of continuous operators. This does not mean that every trained neural operator will be accurate in practice, but it provides theoretical motivation for using these architectures to approximate PDE solution operators.

In this thesis, the term neural operator is used in a practical operator-learning sense. The goal is to learn mappings from input fields to solution fields for PDEs on triangular surface meshes. Several models in this thesis are evaluated in a fixed-mesh setting and therefore do not demonstrate discretisation invariance in the strongest experimental sense. Graph-based architectures can in principle be applied to different meshes when their input features and connectivity are defined consistently. Spectral graph models can also be evaluated on another mesh, but require the eigenbasis of that mesh to be computed and stored, and care is needed because the spectral basis is mesh-dependent. These models are still relevant because the thesis focuses on applications where the same object may require many repeated PDE solves. In that setting, specialising to a fixed mesh can be useful, even if it sacrifices some of the generality expected from a fully discretisation-invariant neural operator.

3.5.2. Graph and spectral neural architectures on meshes

Triangular meshes are irregular domains. Unlike images or regular grids, mesh vertices are not arranged in rows and columns. Each vertex can have a different number of neighbours, and the geometry of the triangles influences how physical quantities vary over the surface. This makes standard grid-based neural operators, such as the Fourier Neural Operator, difficult to apply directly to mesh-based PDE problems. For this reason, architectures for triangular meshes often use either the graph structure of the mesh or a spectral representation derived from the mesh.

A triangular mesh naturally defines a graph. The vertices of the mesh become graph nodes, and the mesh edges become graph edges. Graph neural networks use this structure to update the information stored at each vertex based on information from neighbouring vertices. This process is known as message passing. In one message-passing layer, each vertex receives information from its immediate neighbours. By stacking multiple message-passing layers, information can propagate over longer distances across the mesh, allowing each vertex to be influenced by vertices that are several edges away.

This makes graph-based models a natural choice for surface PDEs, since they respect the local connectivity of the mesh. They can learn how the function values and features of nearby vertices influence each other through message passing and are especially relevant when local structure is important, such as for localised source terms or force fields. However, standard message passing is inherently local: information only travels a limited number of edges per layer. This can make it difficult to capture global dependencies unless the model uses many layers or additional mechanisms for long-range communication.

Spectral neural operators take a different approach. Instead of passing information locally over mesh edges, they represent functions in a spectral basis. On regular grids, Fourier neural operators use Fourier modes for this purpose. On triangular meshes, a comparable spectral representation can be obtained from the eigenvectors of the mesh Laplacian. A function on the mesh can be projected into this spectral space, modified by learnt spectral operations, and transformed back to vertex values.

The advantage of this approach is that spectral operations can model global interactions over the mesh more directly than purely local message passing. This is useful for PDE solution operators, where the solution at one vertex may depend on input values over a large part of the surface. In the fixed-mesh setting considered in this thesis, the spectral basis only needs to be computed once for each mesh before training. The same basis can then be reused for all training and test samples on that mesh.

Graph and spectral models therefore provide two different ways to learn PDE solution operators on triangular meshes. Graph-based models operate directly on local mesh connectivity, while spectral models operate in a global basis associated with the mesh geometry. The experiments in this thesis compare these approaches to study their accuracy, inference speed, and model size for PDEs on fixed triangular mesh surfaces.

4

Methods

4.1. Problem setup

This work considers operator learning on fixed triangular meshes. Let $\mathcal{M} = (V, E, F)$ denote a fixed triangular mesh, where V is the set of vertices, E is the set of edges, and F is the set of triangular faces. The number of vertices is denoted by

$$N = |V|.$$

For each PDE task, the goal is to learn a mapping from an input field defined on $\mathcal{M}$ to the corresponding solution field defined on the same mesh.

Let $\mathcal{A}_{\mathcal{M}}$ denote the space of input fields and let $\mathcal{U}_{\mathcal{M}}$ denote the space of solution fields on the fixed mesh $\mathcal{M}$. A dataset consists of m input-output pairs

$$\{(a_i, u_i)\}_{i=1}^{m},$$

where $a_i \in \mathcal{A}_{\mathcal{M}}$ is an input field and $u_i \in \mathcal{U}_{\mathcal{M}}$ is the corresponding ground-truth solution field. Depending on the PDE task, these fields can be scalar-valued or vector-valued. For scalar problems, such as the Poisson problem or geodesics, the field is represented by one value per vertex. For vector-valued problems, such as linear elasticity, the field is represented by three values per vertex, corresponding to the x, y, and z components of the force applied and the displacement.

In the discrete setting used by the models, a field on a mesh with N vertices and C channels is represented as

$$a_i \in \mathbb{R}^{N \times C_{\text{in}}}, \qquad u_i \in \mathbb{R}^{N \times C_{\text{out}}}.$$

For the Poisson and geodesics tasks, $C_{\text{in}} = C_{\text{out}} = 1$. For the linear elasticity task, $C_{\text{in}} = C_{\text{out}} = 3$. The neural operator G_θ, with trainable parameters θ, is trained to predict

$$\hat{u}_i = G_\theta(a_i),$$

where $\hat{u}_i$ is the predicted solution field.

The models are trained by minimising a relative L_2 loss over the training set:

$$\mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{\|G_\theta(a_i) - u_i\|_2}{\|u_i\|_2 + \epsilon}.$$

Here, the norm is computed over all vertices and channels of the output field, and ϵ is a small constant used for numerical stability. This loss measures the prediction error relative to the magnitude of the target solution, rather than only measuring the unnormalised L_2 error.

The fixed-mesh assumption is central to this thesis. For each experiment, the mesh geometry, connectivity, number of vertices, and vertex ordering are fixed during training and testing. The goal is therefore not to learn a single model that generalises across arbitrary meshes, but to learn mesh-specific surrogate solvers for PDEs on a given triangular surface. This setting is relevant for real-time graphics and interactive simulations, where many repeated solves may be required for the same object.

4.2. Dataset generation

Since this research is about learning PDE solution operators, ground-truth data can be generated by solving the PDE. In this work, training data is generated by sampling the right-hand side input function based on a specific strategy per PDE family, and the ground truth solution function is computed either analytically or numerically. The resulting data is then split into training, validation, and test sets. The following sections explain the right-hand side strategies chosen.

4.2.1. Poisson

For the surface Poisson problem (Section 3.2), three right-hand side strategies are considered. Each strategy first defines a raw forcing field f_{raw}. Since the meshes used in this thesis are closed surfaces, the Poisson problem has a constant null space. Therefore, before storage and solving, the raw forcing field is projected to have zero mass-weighted mean and is normalised. The three raw right-hand side strategies are:

1. Delta functions: a random vertex is set to 1 while all other vertices are set to 0. For a uniformly chosen vertex index $k \in \{0, \dots, N-1\}$, the raw forcing field is defined as

$$f_{\text{raw},i} = \begin{cases} 0, & \text{if } i \neq k, \\ 1, & \text{if } i = k. \end{cases}$$

   After the zero-mean projection and normalisation, the stored model input is no longer exactly one-hot, but remains a delta-like localised right-hand side.

2. Polynomial function: To generate smooth global patterns, polynomial functions in $\mathbb{R}^3$ are evaluated at the (normalised) mesh vertex positions. The chosen polynomial right-hand side is defined as

$$p(x, y, z) = c_0x^3 + c_1y^3 + c_2z^3 + c_3x^2 + c_4y^2 + c_5z^2 + c_6x + c_7y + c_8z + c_9xy + c_{10}xz + c_{11}yz + c_{12},$$

   with uniformly chosen coefficients $c_0, \dots, c_{12} \sim \mathcal{U}(0, 0.5)$. For each vertex v_i with normalised position $\mathbf{x}_i = (x_i, y_i, z_i)$, the raw forcing value is set as $f_{\text{raw},i} = p(x_i, y_i, z_i)$.

3. Sinusoidal function: For a smooth yet more complex pattern on the triangle mesh, a sinusoidal function in 3D space is also defined.

$$f_{\text{raw},i} = \frac{1}{10} \sum_{r=1}^{10} \left(\frac{1}{2}\right)^r \sin\left(2^r\, \mathbf{d}_r \cdot \mathbf{x}_i + \phi_r\right),$$

   where $\mathbf{x}_i = (x_i, y_i, z_i)$ is the normalised position of vertex v_i, $\mathbf{d}_r \in \mathbb{R}^3$ is a random vector sampled from a normal distribution, and ϕ_r is a random phase. With 10 sine terms with powers of 2 as frequencies and powers of 0.5 as amplitudes, this yields a more complex but still smooth pattern.

4.2.2. Geodesics with the heat method

For geodesics using the heat method (Section 3.3), this work employs input fields with one or multiple source points and learns the mapping to the corresponding heat method geodesic distance field. In this research, testing is mainly conducted with 1, 3, or 5 source points. Increasing the number of source points makes the problem more challenging but also more interesting.

For a source set $S \subset V$, the input field is represented as

$$\delta_S(v_i) = \begin{cases} 1, & \text{if } v_i \in S, \\ 0, & \text{otherwise.} \end{cases}$$

The corresponding target output is the geodesic distance field ϕ_S, computed using the heat-method pipeline introduced in Section 3.3 and illustrated in Figure 3.1.

4.2.3. Elasticity

For each elasticity force-field initialisation (Section 3.4), a raw vector-valued force field f_{raw} is first generated. Before storage and solving, this force field is projected to remove rigid-body mode components and is normalised. Therefore, the models are trained on the processed force fields rather than directly on the raw initialisations. As a result, local force initialisations, such as the normal delta force, are not necessarily perfectly local after preprocessing. Whereas the Poisson and geodesic experiments use scalar right-hand sides, the elasticity problem uses three-dimensional vectors

$$\mathbf{f}_i \in \mathbb{R}^3$$

at every vertex v_i. The neural operator therefore learns a mapping from an applied force field to the resulting displacement field of the mesh.

For all linear elasticity force-field initialisations, the Reissner–Mindlin shell solver is used with a fixed shell thickness $h = 0.05$, Young's modulus $E = 1.0$, and Poisson ratio $\nu = 0.3$. The value $E = 1.0$ is used in nondimensional units determined by the mesh scaling and solver setup. These parameters are kept constant across all elasticity experiments, so the models learn the mapping from varying force fields to displacement fields for one fixed material and shell configuration.

Four different right-hand side initialisations are considered. Three of these resemble the choices used for the Poisson right-hand sides.

The considered right-hand sides are

1. **Normal delta force:** A single vertex is chosen uniformly at random, and a force is applied in the direction of the vertex normal. All other vertices receive zero force. More formally, for a randomly chosen vertex k,

$$\mathbf{f}_i = \begin{cases} \mathbf{n}_i, & \text{if } i = k, \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

   where $\mathbf{n}_i$ is the normal vector at vertex v_i. This creates a highly localised load and tests whether the models can learn the global deformation representing a pulling force at one point.

2. **Smooth normal bumps:** Multiple random vertices are selected as bump centres. Around each centre, a Gaussian weight is computed based on the Euclidean distance between the centre and the surrounding vertices. These weights are combined with random amplitudes to form a smooth scalar field over the mesh. The final force is obtained by multiplying this scalar field with the vertex normals:

$$\mathbf{f}_i = s_i \, \mathbf{n}_i.$$

   This creates smooth normal forces that are less localised than delta functions but still spatially varying. These loads are useful for testing whether the models can learn smooth deformations caused by distributed pressure-like forces. This setting plays a similar role to the smoother Poisson right-hand sides: it tests whether the models can learn broad, smoothly varying input-output behaviour rather than highly localised responses.

3. **Sinusoidal:** To generate smooth global vector fields, low-frequency trigonometric functions are evaluated at the three-dimensional vertex coordinates. Each force component is defined as a random combination of sine and cosine functions:

$$\begin{aligned} f_x(x, y, z) &= a_1 \sin(\pi x) + a_2 \cos(\pi y) + a_3 \sin(\pi z) \\ f_y(x, y, z) &= b_1 \cos(\pi x) + b_2 \sin(\pi y) + b_3 \cos(\pi z) \\ f_z(x, y, z) &= c_1 \sin(\pi x) + c_2 \cos(\pi y) + c_3 \sin(\pi z) \end{aligned}$$

   where the coefficients a, b, and c are sampled randomly. This produces a smooth global force field and is expected to be easier to learn than highly localised forces, since the input varies smoothly over the whole mesh.

4. **Hydraulic press:** This strategy creates a more physically motivated loading case. A random coordinate axis is selected, and Gaussian-like force weights are placed near the minimum and/or maximum boundary of the object along that axis. The raw force is applied in the selected coordinate-axis direction:

$$\mathbf{f}_{\text{raw},i} = \alpha_i \, \mathbf{d},$$

where **d** is the selected coordinate-axis direction, and α_i is large near the selected boundary region and small elsewhere. Depending on the random sign, and on whether one or both sides of the object are selected, the force can approximate a compression, tension, or one-sided press-like load along the chosen axis. This should therefore be interpreted as an axis-aligned hydraulic-press-like force, not as a pressure force applied in the local vertex-normal direction. As with the other elasticity right-hand sides, the raw force field is projected to remove rigid-body mode components and normalised before storage and solving.

4.3. Model architectures

This work compares several model architectures for learning solution operators. Each model has its own benefits and drawbacks. This section presents the models considered for this research. These models can be categorised into standard neural network baselines, graph-based models, and spectral models.

4.3.1. Multilayer perceptron

The standard Multilayer Perceptron (MLP) is included as a baseline. This model is not a discretisation-invariant neural operator; instead, it is a fixed-dimensional baseline that learns a mesh-specific approximation of the underlying operator. Nevertheless, since the setting under consideration involves fixed objects, it is important to see how this simple model compares to more complex neural operator architectures and to evaluate whether more structured architectures are necessary for operator learning on triangle meshes.

The MLP consists of linear, fully connected layers and non-linear GELU activation functions. The input of size $N \times C$ (where N is the number of vertices and C is the number of input channels) is first flattened to a vector of size NC. The first linear layer maps from NC to H, where H is the hidden size. The final linear layer maps from H back to NC_{out}, which is then reshaped to an output of size $N \times C_{\text{out}}$. For scalar PDEs, the configuration uses $C = C_{\text{out}} = 1$, whereas for elasticity it uses $C = C_{\text{out}} = 3$.

4.3.2. DeepONet

DeepONet models consist of two parts: the branch network and the trunk network.

The branch network encodes the input function, while the trunk network encodes the spatial location of the point to be evaluated. Both of these networks are implemented with an MLP.

The branch network consumes the full flattened input field over all vertices and channels, i.e., it takes as input a vector of size $N\,C$ containing all scalar values of all vertices and all channels. The trunk network consumes a 3D vertex coordinate at which the model should predict the solution.

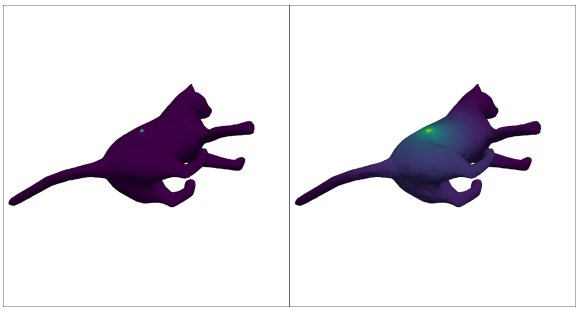
For vector-valued outputs, the branch and trunk network outputs are arranged by output channel and combined with an einsum-style dot product over their latent dimensions to produce all output channels simultaneously. Concretely, for each output channel the dot product is taken between the corresponding branch and trunk feature vectors.

This architecture is therefore a per-point operator rather than a full solution operator. By evaluating the trunk network at all mesh vertices and combining it with the (shared) branch representation via the einsum-style contraction, the full solution function over the mesh is predicted and then compared with the ground-truth solution.

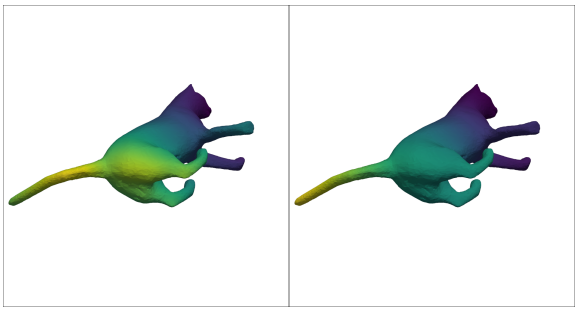
4.3.3. Graph neural network

Graph neural networks are a natural choice for learning solution operators on triangular meshes, since the mesh already defines a graph structure. In this graph, vertices correspond to nodes and mesh edges correspond to graph edges. This allows the model to update each vertex feature vector or scalar value using information from neighbouring vertices while respecting the local connectivity of the surface.

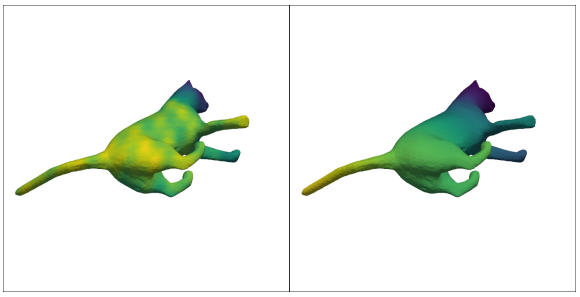
In this work, the graph neural network is implemented as a message-passing model on the mesh graph. For every undirected mesh edge, two directed edges are added so that information can be passed in both directions. In addition, a self-loop is added to every vertex. Each directed edge is assigned geometric edge features consisting of the edge vector and its Euclidean length. The input field values are also used as features: each node receives its input value as its initial node feature, and each edge feature is extended with the input value at the source vertex, the input value at the target vertex, and



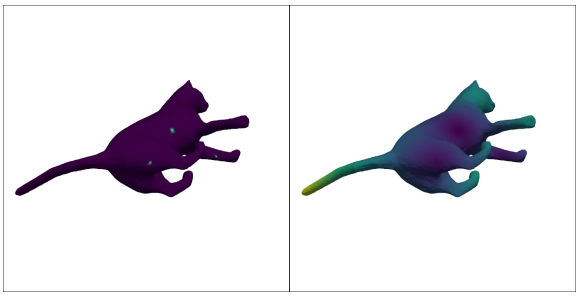
(a) Poisson delta function right-hand side initialisation (left) and ground truth solution (right)



(b) Poisson polynomial function right-hand side initialisation (left) and ground truth solution (right)

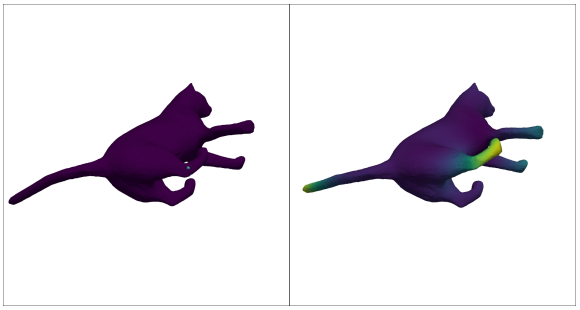


(c) Poisson sinusoidal function right-hand side initialisation (left) and ground truth solution (right)

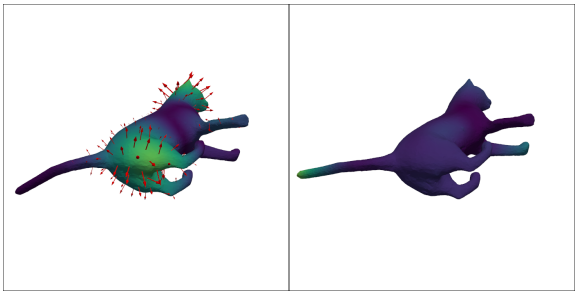


(d) Geodesics function right-hand side initialisation with 5 source points (left) and ground truth solution (right)

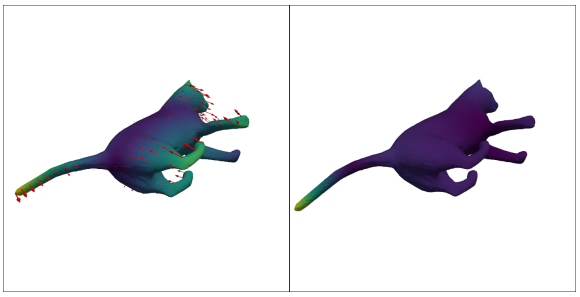
Figure 4.1: Poisson right-hand-side initialisations and geodesic source initialisations, together with their corresponding ground-truth solution fields. The geodesic source initialisation is shown only for five source points to avoid redundancy.



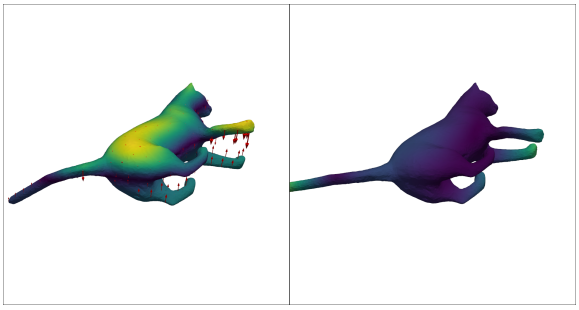
(a) Elasticity delta function along the vertex normal right-hand side initialisation (left) and ground truth solution (right)



(b) Elasticity smooth bumps function right-hand side initialisation (left) and ground truth solution (right)



(c) Elasticity sinusoidal function right-hand side initialisation (left) and ground truth solution (right)



(d) Elasticity hydraulic press right-hand side initialisation (left) and ground truth solution (right)

Figure 4.2: All linear elasticity right-hand side function initialisations and their corresponding ground-truth solutions; vectors are shown to indicate the force direction rather than only the magnitude.

their difference. For an edge from vertex i to vertex j, the edge feature is therefore given by

$$\mathbf{a}_{ij} = \left[\mathbf{x}_j - \mathbf{x}_i,\ \|\mathbf{x}_j - \mathbf{x}_i\|,\ f_i,\ f_j,\ f_j - f_i\right],$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the vertex coordinates, and f_i and f_j are the input field values at the source and target vertices. This gives the network access to both the local mesh geometry and the local variation of the input field.

The message-passing layer computes messages by concatenating the source-node embedding with the edge features and passing this through a learnt linear map. Separate linear maps are used for regular mesh edges and self-loops. The incoming messages at each vertex are aggregated using a channel-wise maximum:

$$\mathbf{h}_i' = \max_{j \in \mathcal{N}(i) \cup \{i\}} \mathbf{m}_{j \to i}.$$

This max aggregation is less similar to local averaging than mean aggregation and is therefore useful for preserving strong local responses, such as those caused by localised source terms or concentrated forces.

After each graph convolution, GroupNorm and a ReLU non-linear activation are applied. The full network first maps the input node features to a hidden embedding space and then applies three residual graph blocks with a hidden size of 64. Each residual block consists of two graph convolution layers, with a skip connection from the input of the block to its output:

$$\mathbf{h}^{(l+1)} = \mathrm{ReLU}\Big(\mathbf{h}^{(l)} + F\big(\mathbf{h}^{(l)}\big)\Big).$$

These residual connections help stabilise training and allow multiple message-passing layers to be stacked without making optimisation unnecessarily difficult.

The final prediction is produced by a multilayer perceptron decoder. Each final vertex embedding is decoded directly into the output field value at that vertex. This makes the model a per-vertex predictor: the network takes an input field defined on the mesh vertices and returns the corresponding predicted solution field on the same vertices.

Although the graph neural network layers themselves operate on graph connectivity and local geometric features, the implementation used in this work is evaluated in the same fixed-mesh setting as the other models. The mesh topology and edge features are constructed from the given mesh, and the batching procedure assumes that all samples in a batch share the same number of vertices and the same connectivity. Therefore, the model is mesh-based and local, but it is not used here as a fully mesh-independent architecture. Its purpose in this comparison is to evaluate how a spatial message-passing model performs against models that make stronger use of fixed global mesh structure, such as spectral neural operators.

4.3.4. Multigrid graph neural network

The Multigrid Graph Neural Network introduces a hierarchy of coarser graph levels to allow information to propagate over longer distances with fewer message-passing steps. The finest level of this hierarchy is the original mesh graph. The implementation is inspired by hierarchical graph neural operator approaches such as the Multipole Graph Neural Operator [20], but is adapted here to the fixed triangular mesh setting. This is useful for PDE solution operators, as the solution at a vertex can depend not only on its immediate neighbourhood but also on information from more distant parts of the surface.

As in the standard graph neural network, the input mesh is represented as a graph whose nodes correspond to mesh vertices. On top of the original mesh graph, three additional coarser levels are constructed, giving a total of four levels. Each coarser level contains a subset of the vertices from the original mesh. In the experiments, a divisor of four is used, meaning that the coarsest level contains approximately $\frac{1}{4^3}$ of the original nodes. These coarser levels allow information to travel over larger parts of the mesh with fewer message-passing steps.

The model stores the vertices from all hierarchy levels in one stacked graph representation. Edges are divided into three types. First, same-level edges connect vertices within the same hierarchy level. These edges allow regular message passing inside each graph resolution. Second, downward edges connect a finer level to the next coarser level, allowing information from the detailed mesh representation to be transferred to a coarser representation. Third, upward edges connect a coarser level back to the finer level, allowing global information to be propagated back to the original mesh resolution.

The same-level edges on the coarser graph levels are constructed using hop distances in the original mesh graph. Level 0 corresponds to the original mesh connectivity. For each higher level l, two selected vertices are connected when their shortest-path distance in the original mesh graph is at most 2^l hops:

$$d_{G_0}(i, j) \leq 2^l$$

where G_0 is the original mesh graph. With four levels, this means that levels 1, 2, and 3 connect selected vertices within 2, 4, and 8 mesh-edge hops, respectively. This gives the coarser levels increasingly larger receptive fields while still measuring distance along the surface connectivity instead of using Euclidean distance in 3D space. If a coarser same-level graph is disconnected, additional bridge edges are added between components using shortest hop distances in the original mesh graph.

Each edge is assigned features based on both geometry and the input field. For an edge from vertex i to vertex j, the edge feature is given by

$$\mathbf{a}_{ij} = \left[\mathbf{x}_j - \mathbf{x}_i,\ \|\mathbf{x}_j - \mathbf{x}_i\|,\ f_i,\ f_j,\ f_j - f_i\right]$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the source and target vertex coordinates, and f_i and f_j are the input field values at these vertices. The input field values are also used as the initial node features. These features give the model access to the local geometry of the mesh, as well as the local variation of the input field along each edge.

The graph update used at each level follows the same max-aggregation message-passing operation as the standard graph neural network. A message is computed from the source-node embedding and the corresponding edge feature. Incoming messages are aggregated at each target node using a channel-wise maximum. The resulting update is passed through GroupNorm and added to the previous hidden representation through a residual connection:

$$\mathbf{h}^{(l+1)} = \mathrm{ReLU}\left(\mathbf{h}^{(l)} + \mathrm{GroupNorm}\left(\mathrm{GraphConv}^{(l)}\left(\mathbf{h}^{(l)}\right)\right)\right).$$

The hidden size is set to 64 in all experiments.

The forward pass follows a multigrid-style cycle. Starting from the finest level, the model first applies same-level message passing and then transfers information downward to the next coarser level. This process is repeated until the coarsest level is reached. After an update on the coarsest level, the model transfers information back upward through the hierarchy. At each finer level, the model first applies an upward update and then applies another same-level update. This forms one complete down-and-up cycle. In the experiments, two such cycles are applied:

$$\text{fine} \rightarrow \text{coarse} \rightarrow \text{fine}$$

This structure allows the network to repeatedly combine local fine-scale information with coarser and more global information.

After the two multigrid cycles, only the embeddings corresponding to the original mesh vertices are kept. The embeddings from the additional coarser levels are used only to support information propagation during message passing. The final prediction is produced by a multilayer perceptron decoder that maps each finest-level vertex embedding to the output field value at that vertex. The model, therefore, takes an input field defined on the original mesh vertices and returns a predicted solution field on the same vertices.

Although the hierarchy introduces coarser graph levels, the implementation is still evaluated in the same fixed-mesh setting as the other models. The hierarchy, edge connections, and edge features are constructed from the given mesh, and all samples for that mesh share the same multigrid structure. The model is therefore not used here as a fully mesh-independent architecture. Its role in this comparison is to test whether explicitly adding coarse graph levels improves the ability of a message-passing model to capture longer-range dependencies on the surface.

Figure 4.3 visualises the selected nodes at the different hierarchical levels. The original mesh forms the finest level, while the higher levels contain increasingly smaller subsets of vertices.

Figure 4.4 shows the edge structure of the multigrid hierarchy. Same-level edges connect vertices within a hierarchy level, while inter-level edges transfer information between finer and coarser levels.

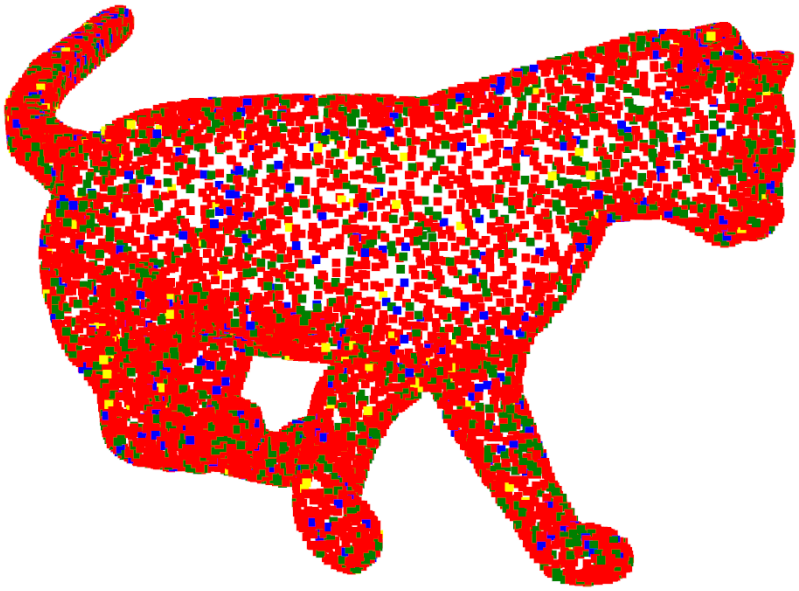


Figure 4.3: Visualisation of the cat mesh with coloured squares indicating the selected nodes at different levels of the multigrid hierarchy. Red denotes the original mesh nodes, while green, blue, and yellow denote increasingly coarser levels, with yellow corresponding to the coarsest level.

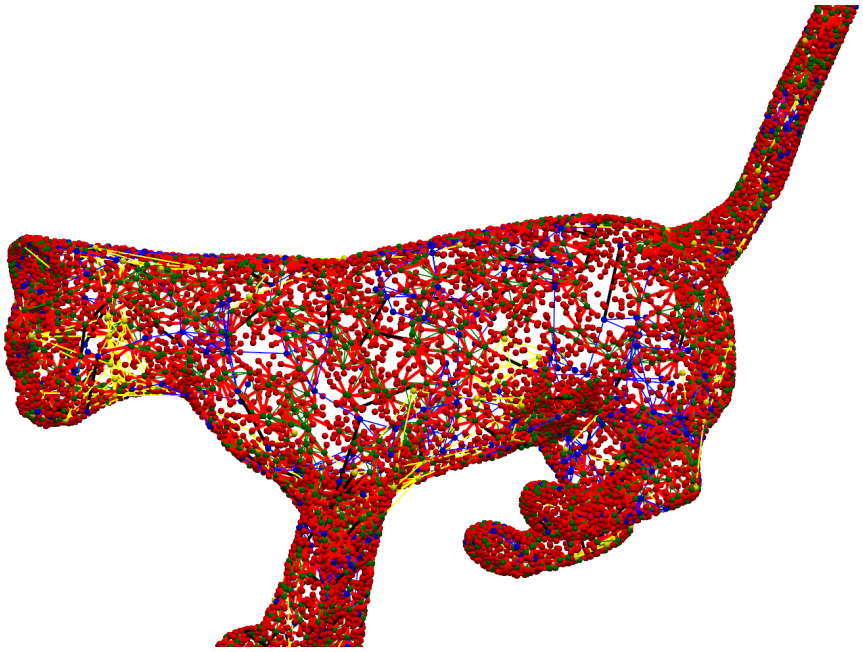


Figure 4.4: Visualisation of the multigrid hierarchy on the cat mesh. Green, blue, and yellow edges show within-level edges for the corresponding coarse graph levels. Red edges indicate inter-level up/down connections rather than original mesh edges. Black edges show additional within-level connections added to keep coarse levels connected and allow information to propagate. Level-0 mesh edges are omitted for readability.

4.3.5. Spectral graph neural network

To better capture global interactions, this work also considers spectral architectures. The spectral graph neural network, in particular, uses global convolutions implemented in the eigenbasis of the mesh Laplacian. Concretely, the signal on the mesh is projected onto the first k eigenvectors of the mesh Laplacian using the mass-weighted projection $V^\top M$, where $V \in \mathbb{R}^{N \times k}$ contains the first k eigenvectors and M is the mass matrix. This projection and its inverse enable transformation between the spatial and spectral domains.

Each spectral layer then operates in this low-dimensional spectral space: it learns an affine spectral filter of the Laplacian eigenvalues and combines the resulting filtered signal with a learnt residual linear map acting in the spatial domain. In practice, this means that for each layer, a learnt function of the eigenvalues is applied to the spectral coefficients (an affine map of the eigenvalues), and a learnt linear transformation of the original features is added as a residual pathway.

The architecture is fixed-mesh: the Laplacian eigenbasis and its associated eigenvalues are pre-computed once for the specific mesh and reused throughout training and inference. This design is mainly inspired by FNO [7] and Sp2GNO [18], which also leverage the first k spectral modes for global convolution; however, FNO applies this idea only to regular grids, while Sp2GNO targets varying geometries and must introduce additional machinery to avoid the $O(N^3)$ cost of repeated eigendecompositions. In contrast, the fixed-mesh setting makes it possible to pre-compute the Laplacian spectrum once and use it directly in every spectral layer.

4.3.6. Hodge Spectral Graph Neural Network

Finally, this work considers Hodge spectral graph neural networks, which extend standard spectral graph neural networks from vertex-only processing to a Hodge/DEC-style treatment of vertices, edges, and faces. Concretely, rather than projecting only onto the eigenfunctions of the vertex Laplacian, the architecture also uses operators associated with the edge and face Laplacians. The resulting three spectral representations (on 0-, 1-, and 2-simplices) are each processed and then combined via learnt weights.

By incorporating information from vertex-, edge-, and face-based operators, this architecture is designed to encode a richer geometric and topological structure than a vertex Laplacian alone, which is particularly important for accurately modelling surface PDEs. Although this approach is motivated by the Hodge-theoretic perspective, it does not claim to implement a full physical Hodge decomposition; instead, it uses Hodge/DEC-style Laplacians as multi-order spectral operators, without explicitly constructing or analysing the full decomposition and associated projection operators.

5

Experiments and Results

5.1. Meshes

Each learnt solution operator was evaluated on two meshes: bunny and cat. Two quite different meshes were chosen to determine if mesh size and mesh geometry influence accuracy, inference time, and speed-up. The meshes were obtained from the SHREC11 dataset and from the Stanford Computer Graphics Laboratory [21, 22]. Each mesh was rescaled and re-centred before model training and evaluation (Table 5.1, Figure 5.1).

Table 5.1: Mesh sizes

Mesh	Vertices
Bunny	2503
Cat	9272

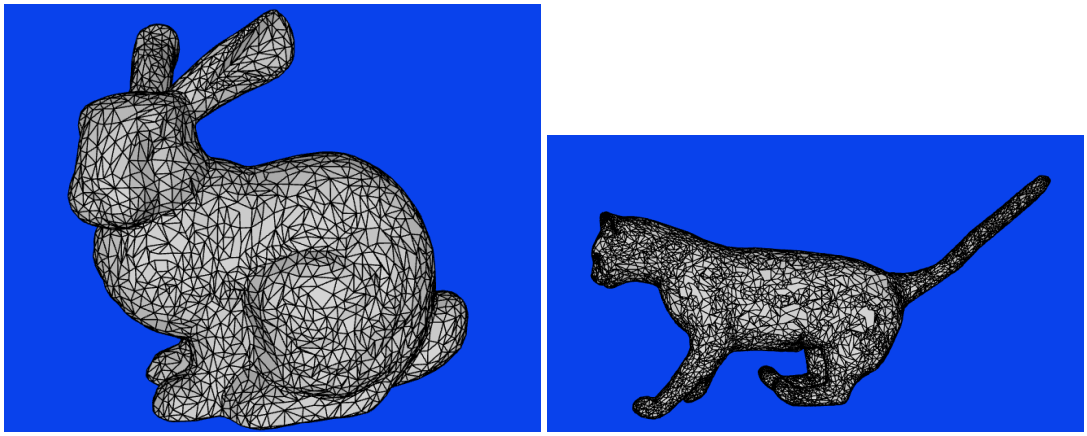


Figure 5.1: The two meshes used for experiments.

5.2. Training and testing

The training data consists of paired samples (x_i, y_i), where x_i denotes the right-hand-side input function and y_i denotes the corresponding ground-truth solution field. Early stopping is used with a patience of 30, a delta of 1×10^{-10}, and a validation set of 2000 instances. The Adam optimiser was used with weight decay set to 1×10^{-6}. Loss is computed based on relative L_2 error (Eq. 4.1) between the ground-truth field and the predicted field. Each model was evaluated on a test set of 5000 instances, generated in the same way as the training data. The models for the Poisson problem were trained with a learning rate of 0.003, while the models for the geodesics and elasticity problems were trained with a learning rate of 0.005.

Training and evaluation have been conducted on an RTX 5070 TI with 16 GB of VRAM. No shared RAM was used. For the MLP, DeepONet, and spectral-based models, a batch size of 256 was used. The GNN and Multigrid GNN require substantially more VRAM due to the message passing layers. These have therefore been trained using a batch size of 8.

5.3. Hyper-parameters

The models used in this work were trained with fixed hyper-parameter settings chosen to provide a consistent comparison between the different architectures. The Standard MLP used one hidden layer with a hidden size of 128. For DeepONet, both the branch network and the trunk network used one hidden layer of size 128, while the final latent representation had size 16.

For the spectral graph neural network, the 64 smallest eigenmodes were used together with three spectral layers of width 16. The Hodge spectral graph neural network used a larger spectral basis for the vertex Laplacian, consisting of the 128 smallest eigenmodes. For the edge and face Laplacians, the 2 smallest eigenmodes were used. This model also used three layers of width 16.

Both the GNN and the Multigrid GNN used the same graph update type, a hidden size of 64, and the same per-vertex MLP decoder with dimensions $64 \to 64 \to C_{\text{out}}$. The GNN consisted of three residual graph blocks (six graph-convolution updates in total). The Multigrid GNN instead used four hierarchy levels with a coarsening divisor of 4 between levels and two multigrid cycles before applying the decoder at the finest level.

The scalar tasks use one output channel, while the elasticity task uses three displacement channels, as described in Section 4.1.

5.4. Experiment choices

The PDEs and right-hand-side strategies were chosen to gradually introduce complexity into the problem and determine how much that affects the ability of a model to learn this problem. The Poisson PDE was chosen as a starting point and proof of concept. This PDE was chosen because it is one of the simpler surface PDEs and is also linear. This makes it suitable for determining whether it is even worth looking into more complex and non-linear PDEs. Within the Poisson experiments, three previously described right-hand side strategies were used: polynomial functions, sinusoidal functions, and delta functions. These strategies were chosen because they represent different levels and types of spatial complexity. For instance, the **polynomial function** forms the simplest right-hand sides that often define a smooth and broad variation across the mesh, where one region of the surface has relatively low values and another region has relatively high values. This is the strategy with the least degrees of freedom and should be the easiest for a model to predict. **Sinusoidal functions** increase complexity; rather than one smooth transition over the surface, this right-hand side strategy presents a more complex pattern across the mesh, often featuring multiple alternating regions of high and low values. As a result, the model has to learn more spatially complex patterns and map them to the correct smooth solution function. The **Delta functions** present a different kind of challenge. Rather than a function defined over the whole mesh, this function is highly localised to one vertex. This makes the problem much harder to learn, as each model needs to correctly learn the effect of the function value at a delta position on the solution function. The models cannot rely on predicting an average of neighbours but need to have expressive power in their weights so that only one high value leads to the correct prediction.

Being able to correctly learn to solve these three choices of right-hand sides shows the expressive power of neural operator models to accurately learn the Poisson PDE on a surface. Geodesics with the heat method (Section 3.3) is a more complex PDE problem. It is a method for approximating geodesic distances using two PDEs (Figure 3.1), one of which is the Poisson PDE. Even though the Poisson equation is a linear PDE, the combination of PDEs used for the heat method makes this problem non-linear. The heat method is still relatively easy for numerical solvers, but because of its non-linearity and complexity, it is a suitable next step for the evaluation of the models. For geodesics using the heat method, this work defines 3 right-hand sides: 1 source point, 3 source points, or 5 source points. Since this problem is defined by the number of source points and their locations, there is not much choice for different right-hand sides. The one-source-point case is somewhat similar to the Poisson delta-function right-hand side. However, using multiple source points makes the problem more complex, as rather than the solution always being a smooth transition like for Poisson, multiple source points introduce more complex solution functions, sometimes causing alternating patterns.

Finally, linear elasticity was used as an even more complex PDE. Unlike Poisson and geodesics, linear elasticity is defined as a vector field over the surface rather than a scalar field. This means the models have to learn a displacement vector at each vertex rather than just a scalar value. For this problem, four different right-hand side strategies were considered: delta pulling force along the vertex normal, hydraulic press force, sinusoidal forces, and smooth normal bump forces. Just like the Poisson case, there are two smooth force fields: the sinusoidal forces and smooth normal bumps, and there is one localised delta right-hand side. Furthermore, there is one extra case, which is the hydraulic press right-hand side. This is a more physically interesting case.

5.5. How accurately can neural operator models learn solution operators for PDEs on surfaces of fixed triangular meshes?

To answer the first research question, which asks how well neural operator architectures can predict solution functions for PDEs on fixed triangular mesh surfaces, isolines were visualised to assess qualitative correctness rather than relying only on loss values. Poisson and geodesics are both based on scalar value fields, which can be visualised using colours; however, for geodesics it is also natural to visualise isolines. Isolines are lines that connect points on a surface or field that have the same value, like contour lines on a height map. Isolines were chosen for multiple reasons. First, they can reveal small discrepancies between the ground truth and prediction that may be visually hidden in colour plots due to the chosen colour scale. In PyVista, the colour scale was set using the minimum and maximum values across both the ground truth and prediction so that they share the same scale. Nevertheless, similar global ranges can still make predictions appear visually perfect even when errors are present (Figure 5.2). Second, isolines provide a clear indication of whether predictions are smooth and physically plausible. Smooth but slightly inaccurate solutions often remain visually plausible, whereas noisy solutions can look unrealistic. Because it is not feasible to include isolines for all test samples, this work uses isolines as qualitative examples rather than exhaustive evidence over the full test set. The isolines themselves are not predicted directly: they are extracted from the ground-truth and predicted scalar fields defined at all vertices. For linear elasticity, it is even harder to visualise error due to the vector field per vertex rather than a scalar value. Because of this, in this thesis, each prediction is visualised using four panels. The top two of them show the ground truth displacement and the predicted displacement. The bottom two show the relative L_2 error and the angular error between predicted and ground-truth displacement vectors. The displacement is visualised by showing the actual displaced vertices and the original mesh with a grey outline; furthermore, the magnitude of the displacement is shown using colours to represent scalar magnitudes.

These instances are referred to as representative instances: they are test set samples whose prediction loss is closest to the average loss over the full test set. This is to ensure a fair comparison and to show the accuracy of a representative model prediction for a given problem. For each problem and its right-hand sides, only the best models are included as visualisations to avoid an excessive number of visualisations.

To avoid an excessive number of visualisations in the main text, additional representative predictions are included in Appendix A.1–A.6. Appendices A.7 and A.8 further include representative and worst-case test-set visualisations, which provide extra insight into model robustness.

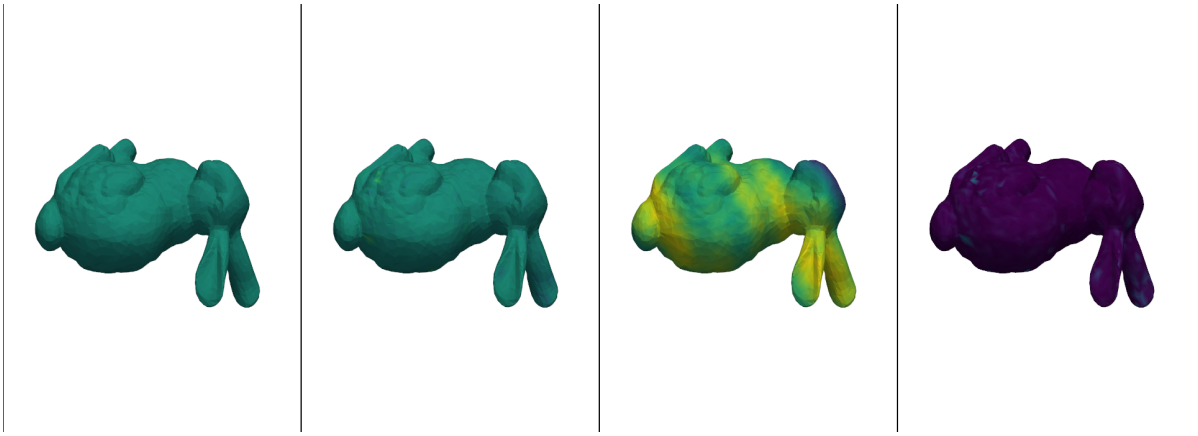


Figure 5.2: Visualisation of scalar function values using direct colour mapping. From left to right, the panels show the ground-truth solution, the model prediction, the right-hand side initialisation, and the magnitude of the prediction error. The prediction and ground truth appear visually similar, making small differences difficult to identify from the colour fields alone. The unnormalised error field is also difficult to interpret visually, while normalising each error field separately could exaggerate small errors and make all samples appear similarly inaccurate.

5.5.1. Poisson

Figure 5.3: Representative test-set predictions for the polynomial right-hand side, shown for the best-performing models. The Spectral GNN (left) performs the best; the isolines are completely overlapping. The Hodge Spectral GNN (middle) also performs exceptionally well. The Multigrid GNN (right) is quite accurate, but is not as closely overlapping as the spectral graph models. (yellow shows the model's prediction; pink shows the ground truth)

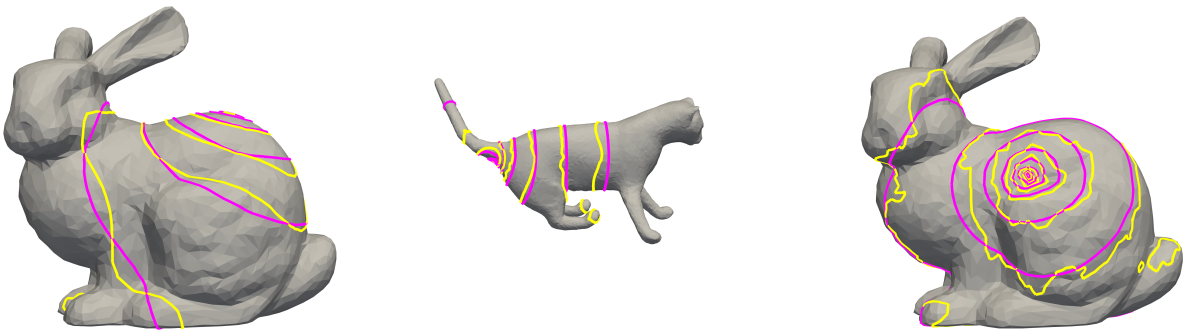


Figure 5.4: Representative test-set predictions for the delta right-hand side, shown for the best-performing models. The delta right-hand side appears to be the most difficult of the Poisson right-hand sides to predict for any model. Even the Spectral GNN (left) and Hodge Spectral GNN (middle) do not have perfectly overlapping isolines. The Multigrid GNN (right) also performs relatively well on the delta right-hand side. (yellow shows the model's prediction; pink shows the ground truth)

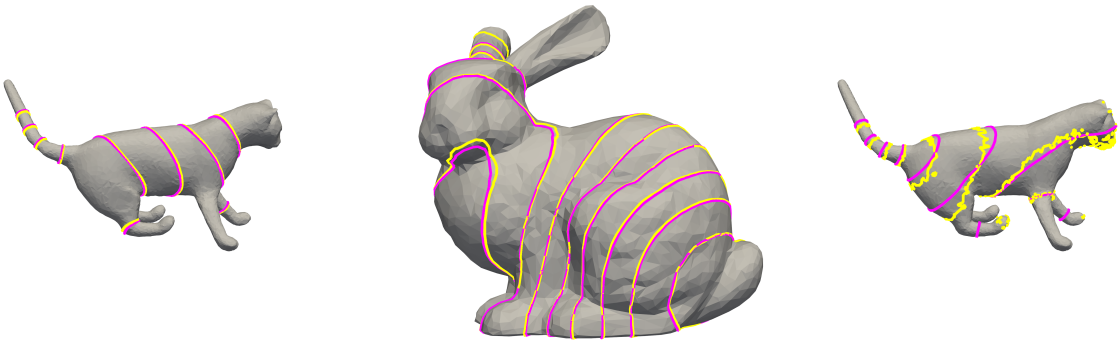


Figure 5.5: Representative test-set predictions for the sinusoidal right-hand side, shown for the best-performing models. Like the polynomial right-hand sides, the spectral models (left and middle) seem to have closely overlapping isolines. The standard MLP (right) seems to predict quite well for the sinusoidal right-hand side. (yellow shows the model's prediction; pink shows the ground truth)

The results show that the spectral models, in particular, perform very well for all right-hand sides of the Poisson problem. In the polynomial and sinusoidal right-hand sides, the isolines of the prediction are seamlessly overlapping, meaning that the predicted function value field is a near perfect prediction. On the polynomial right-hand side, the Multigrid GNN performed quite well. However, with the isoline visualisations, it is still apparent that the prediction is not perfect. Despite the isolines not perfectly overlapping with the isolines from the ground truth, this still seems like a very good prediction. On the sinusoidal right-hand side, the Multigrid GNN performed significantly worse, while the MLP made a relatively good prediction on average. For the delta function, the dominant spectral models had more trouble. Their predictions are still relatively accurate, but the isolines are not as seamless as the polynomial or sinusoidal right-hand side predictions.

5.5.2. Geodesics

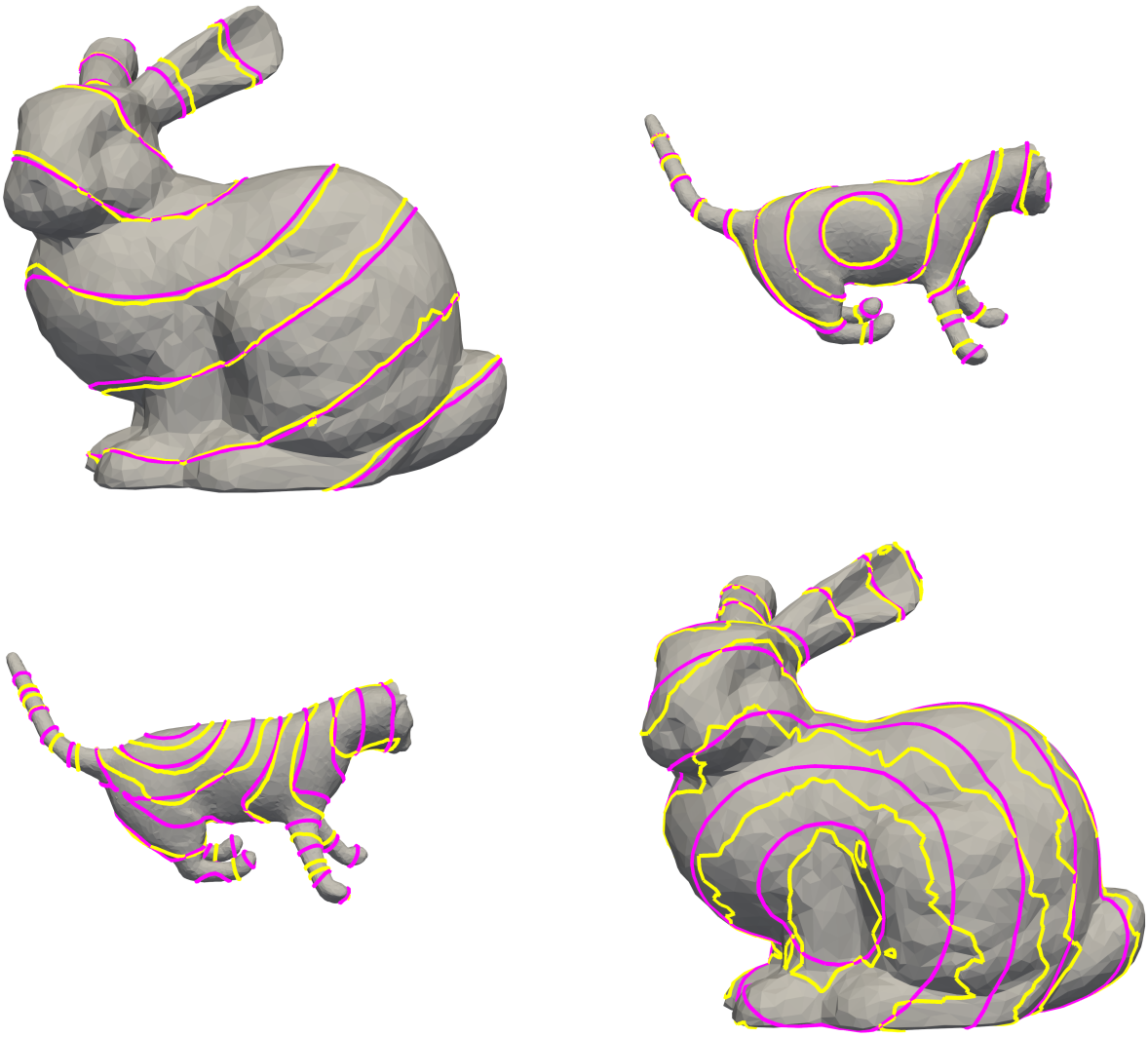


Figure 5.6: Isoline comparison of the geodesics problem with different numbers of source points. Top left and top right show the Hodge Spectral GNN on the one-source-point and three-source-point instances respectively; both predictions are quite good. Bottom left and bottom right show the Spectral GNN on a five-source-point instance and the Multigrid GNN on a one-source-point instance respectively. Even though the predictions are not closely aligned, they are still somewhat accurate predictions since isoline locations can be very strict. (yellow shows the model's prediction; pink shows the ground truth)

Geodesic distance computation using the heat method is a more complex problem, and this is also shown in the results. Despite the Hodge Spectral predictions being quite good on average, other models seem to have quite a few visible errors using the isolines. Except for the Hodge Spectral model, all models had considerable difficulty learning this PDE problem. Even though all options for the number of source points lead to decent results, the models do vary in accuracy based on how many source points are used.

5.5.3. Linear elasticity

The linear elasticity problem is visually different from the Poisson and geodesic experiments, as the solution is a displacement field rather than a scalar value field. Because of this, isolines are less directly applicable. Instead, the prediction quality can be assessed by comparing the ground truth mesh deformation with the deformation from the model's prediction. Furthermore, magnitude error and angular error can be visualised using colours on the mesh to assess smaller differences within deformations.

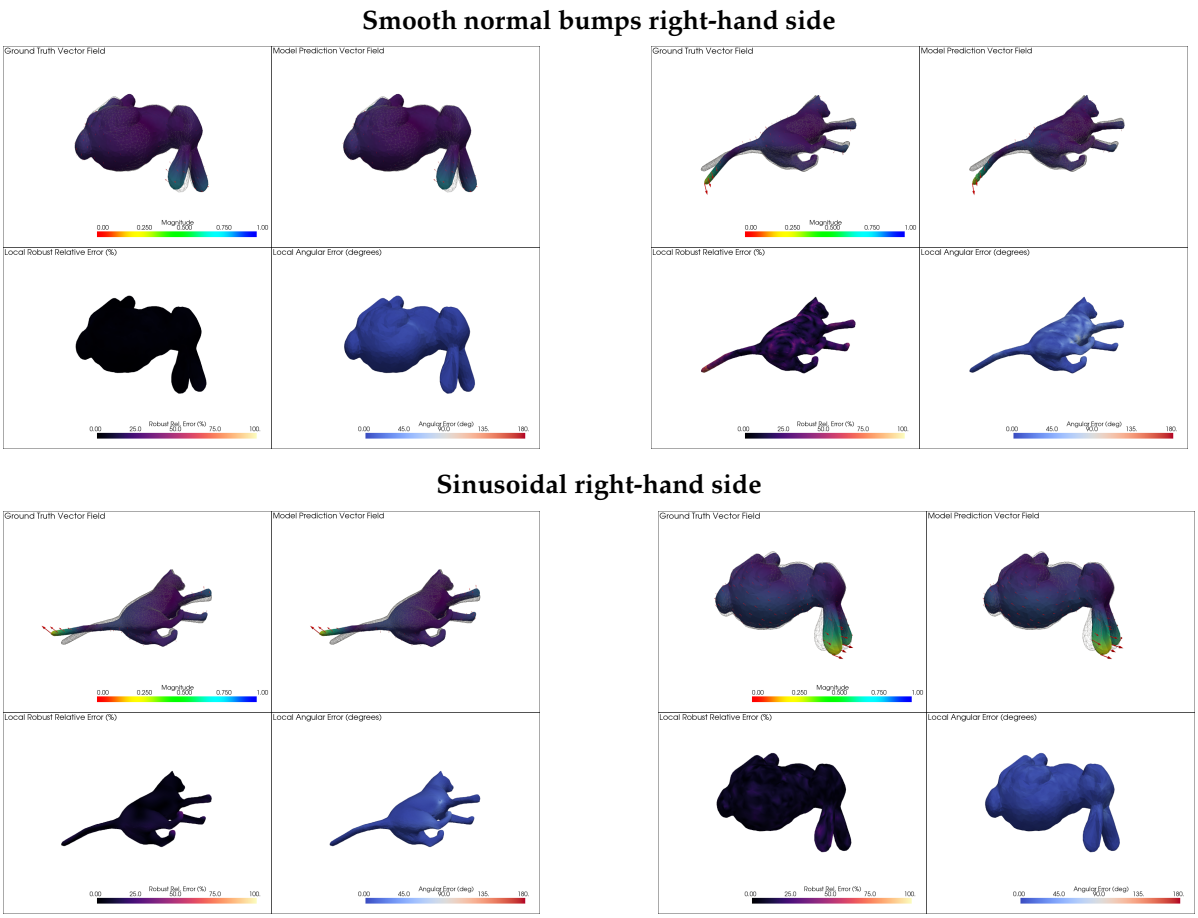


Figure 5.7: Error visualisation of representative test-set predictions on the linear elasticity problem initialised with the smooth normal bumps right-hand side (top two) and the sinusoidal right-hand side (bottom two). The MLP (top left), Hodge Spectral GNN (top right), DeepONet (bottom left) and Multigrid GNN (bottom right) all perform exceptionally well on these right-hand sides. The MLP even shows no visual error with this visualisation method.

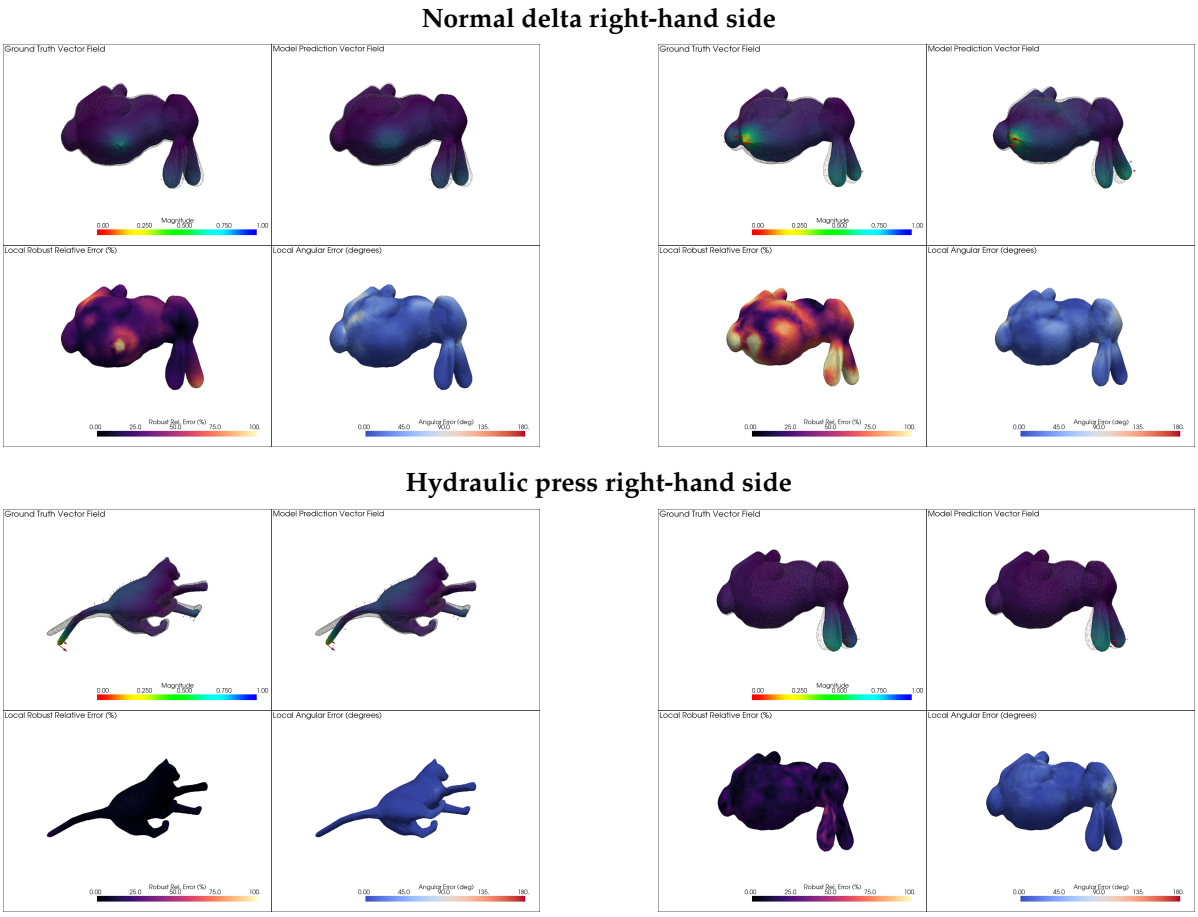


Figure 5.8: Error visualisation of representative test-set predictions on the linear elasticity problem initialised with the normal delta right-hand side (top two) and the hydraulic press right-hand side (bottom two). The MLP (top left) and Hodge Spectral GNN (top right) perform relatively well on the normal delta right-hand side, even though this is the most challenging elasticity setting. Compared to the smoother right-hand sides, these predictions still exhibit noticeable errors. The hydraulic press case is learnt accurately by the MLP (bottom left) and the GNN (bottom right), with low errors in these representative predictions.

The elasticity visualisations do not indicate a single consistently best-performing model. Instead, each model performs well for some right-hand sides and less well for others. The MLP performs strongly on several of the smoother elasticity settings, in particular hydraulic press, sinusoidal, and smooth normal bumps. This suggests that, for these settings, the deformation fields are sufficiently smooth and globally structured for the coordinate-based baseline to approximate effectively. However, the normal delta force remains more difficult to predict, especially on the larger cat mesh. The Graph Neural Network and DeepONet, despite not achieving high accuracy on geodesics and Poisson, show comparatively stronger results on linear elasticity.

5.6. How do different models compare in terms of accuracy?

To determine whether more complex models outperform fast MLP-based baselines, qualitative isoline visualisations are not sufficient on their own. Therefore, to compare models across the full test set, a quantitative loss experiment was performed. The goal of this experiment is to measure, on average, how accurately each model predicts the solution function over all vertices and all test samples.

In addition to the qualitative comparisons, this provides a consistent numerical evaluation across models because all models are tested on the same datasets. Lower loss values generally indicate more accurate models.

The following tables show the loss on the test set for each model.

5.6.1. Poisson

Table 5.2: Poisson test losses on the bunny and cat meshes. Lower is better.

	Polynomial		Delta function		Sinusoidal function	
Model	**Bunny**	**Cat**	**Bunny**	**Cat**	**Bunny**	**Cat**
MLP	4.611×10^{-1}	3.084×10^{-1}	1.493×10^{1}	4.248×10^{2}	5.298×10^{-2}	8.812×10^{-2}
DeepONet	8.687×10^{-2}	1.315×10^{-1}	9.357×10^{-1}	7.33	1.478×10^{-1}	2.158×10^{-1}
Spectral Graph	$\mathbf{8.478 \times 10^{-3}}$	$\mathbf{9.21 \times 10^{-3}}$	2.161×10^{-1}	1.052	$\mathbf{8.12 \times 10^{-3}}$	1.867×10^{-2}
Hodge Spectral	3.106×10^{-2}	1.143×10^{-2}	$\mathbf{1.054 \times 10^{-1}}$	$\mathbf{1.492 \times 10^{-1}}$	2.119×10^{-2}	$\mathbf{1.247 \times 10^{-2}}$
GNN	1.274×10^{-1}	5.571×10^{-2}	6.372×10^{-1}	9.216×10^{-1}	4.574×10^{-1}	5.172×10^{-1}
Multigrid GNN	3.434×10^{-2}	3.632×10^{-2}	2.499×10^{-1}	9.924×10^{-1}	2.315×10^{-1}	3.327×10^{-1}

As can be seen from Table 5.2, the spectral models obtain the lowest losses for all function value initialisations. The Spectral Graph model achieves the best results on both polynomial right-hand sides and on the sinusoidal function right-hand side for the bunny mesh. The Hodge Spectral model achieves the best results on both delta function right-hand sides and on the sinusoidal function right-hand side for the cat mesh. This is likely due to the Poisson problem being modelled using the mesh Laplacian, while both spectral methods use the eigenvalues and eigenvectors of the mesh Laplacian to perform convolution in spectral space.

The other models generally perform worse than the spectral models. On the polynomial right-hand sides, the MLP obtains the highest losses for both meshes. On the delta function right-hand sides, the MLP performs especially poorly, particularly on the cat mesh. The delta function right-hand sides are the hardest to learn overall, as they produce the highest losses for every model on both meshes. This is likely because the delta function is highly localised, making the resulting solution more difficult to approximate accurately. On the sinusoidal function right-hand sides, the GNN obtains the highest losses for both meshes, suggesting that this model struggles more with these smoother, more global input functions. The Multigrid GNN improves over the GNN in several cases, but it still does not reach the accuracy of the spectral models.

5.6.2. Geodesics

Table 5.3: Geodesic test losses on the bunny and cat meshes. Lower is better.

	1 source point		3 source points		5 source points	
Model	**Bunny**	**Cat**	**Bunny**	**Cat**	**Bunny**	**Cat**
MLP	2.899×10^{-1}	3.125×10^{-1}	3.391×10^{-1}	4.103×10^{-1}	3.512×10^{-1}	4.346×10^{-1}
DeepONet	3.105×10^{-1}	3.284×10^{-1}	3.419×10^{-1}	4.132×10^{-1}	3.672×10^{-1}	4.437×10^{-1}
Spectral Graph	5.448×10^{-2}	1.687×10^{-1}	9.109×10^{-2}	1.540×10^{-1}	1.156×10^{-1}	1.789×10^{-1}
Hodge Spectral	$\mathbf{1.357 \times 10^{-2}}$	$\mathbf{7.707 \times 10^{-2}}$	$\mathbf{6.679 \times 10^{-2}}$	$\mathbf{5.055 \times 10^{-2}}$	$\mathbf{7.679 \times 10^{-2}}$	$\mathbf{7.712 \times 10^{-2}}$
GNN	3.306×10^{-1}	3.685×10^{-1}	3.095×10^{-1}	4.430×10^{-1}	2.770×10^{-1}	4.361×10^{-1}
Multigrid GNN	3.125×10^{-1}	4.544×10^{-1}	1.431×10^{-1}	2.331×10^{-1}	1.450×10^{-1}	2.130×10^{-1}

Just as is apparent from the isoline visualisation results (Figure 5.6), the heat method for geodesics is a significantly harder problem to learn than the smoother Poisson settings. Although the spectral models still achieve the lowest losses overall (Table 5.3), their losses are noticeably higher than those on the polynomial and sinusoidal-function Poisson right-hand sides. This suggests that approximating geodesic distance fields is more difficult than approximating the smoother Poisson solution fields.

It is interesting to see that the Hodge Spectral Graph Neural Network obtains the lowest loss for every geodesic setting in Table 5.3. It consistently outperforms the standard Spectral Graph Neural Network on both meshes and for all tested numbers of source points. This suggests that the Hodge-based spectral representation is more suitable for this problem, possibly because the heat method involves

differential quantities on the mesh and is therefore closely related to the geometric structure captured by the Hodge-based formulation.

The non-spectral models perform considerably worse overall. The MLP and DeepONet losses generally increase as the number of source points increases, suggesting that these models struggle more as the input field becomes more complex. The standard GNN behaves differently: on the bunny mesh, its loss decreases as the number of source points increases, while on the cat mesh it remains relatively high for all settings. The Multigrid GNN improves substantially from the 1-source-point setting to the 3- and 5-source-point settings, especially on the bunny mesh, but it still does not reach the accuracy of the spectral models.

Overall, the results show that the Hodge Spectral Graph Neural Network is the most accurate model for the geodesic problem.

5.6.3. Linear elasticity

Table 5.4: Linear elasticity test losses on the bunny and cat meshes for the selected force initialisations. Lower is better.

Force initialisation	Model	Bunny	Cat
Normal delta	MLP	$\mathbf{2.368 \times 10^{-1}}$	8.463×10^{-1}
	DeepONet	6.683×10^{-1}	8.427×10^{-1}
	Spectral Graph	8.289×10^{-1}	8.222×10^{-1}
	Hodge Spectral	5.810×10^{-1}	$\mathbf{4.527 \times 10^{-1}}$
	GNN	8.756×10^{-1}	9.899×10^{-1}
	Multigrid GNN	7.558×10^{-1}	9.960×10^{-1}
Hydraulic press	MLP	$\mathbf{2.503 \times 10^{-2}}$	$\mathbf{4.505 \times 10^{-2}}$
	DeepONet	3.777×10^{-1}	9.667×10^{-2}
	Spectral Graph	3.752×10^{-1}	3.505×10^{-1}
	Hodge Spectral	1.157×10^{-1}	1.960×10^{-1}
	GNN	1.235×10^{-1}	2.421×10^{-1}
	Multigrid GNN	1.028×10^{-1}	9.971×10^{-1}
Sinusoidal	MLP	$\mathbf{1.606 \times 10^{-2}}$	$\mathbf{1.748 \times 10^{-2}}$
	DeepONet	1.403×10^{-1}	7.223×10^{-2}
	Spectral Graph	4.290×10^{-1}	3.294×10^{-1}
	Hodge Spectral	7.561×10^{-2}	9.922×10^{-2}
	GNN	1.008×10^{-1}	1.662×10^{-1}
	Multigrid GNN	8.117×10^{-2}	1.212×10^{-1}
Smooth normal bumps	MLP	$\mathbf{2.383 \times 10^{-2}}$	$\mathbf{4.661 \times 10^{-2}}$
	DeepONet	3.561×10^{-1}	1.031×10^{-1}
	Spectral Graph	6.557×10^{-1}	3.750×10^{-1}
	Hodge Spectral	1.755×10^{-1}	1.759×10^{-1}
	GNN	5.432×10^{-1}	4.072×10^{-1}
	Multigrid GNN	3.489×10^{-1}	3.294×10^{-1}

The linear elasticity results show a different trend from the Poisson and geodesics experiments. For the smoother force initialisations, such as hydraulic press, sinusoidal, and smooth normal bumps, the MLP achieves the lowest loss on both meshes. This suggests that these elasticity solutions are smooth enough for a coordinate-based baseline to approximate them effectively. However, this does not necessarily mean that the MLP is the best general model, since the normal delta force shows a different behaviour. On the normal delta force, the MLP performs best on the bunny mesh, while the Hodge Spectral GNN performs best on the larger cat mesh. This suggests that localised elastic loads are harder to learn and that spectral information can become more useful when the mesh becomes larger or the deformation pattern becomes more localised.

Overall, the elasticity results show that model performance depends strongly on the type of force initialisation. While spectral models dominate the Poisson and geodesic experiments, the MLP is very

competitive for several elasticity settings. This indicates that the best architecture is not only determined by the PDE but also by the structure of the input forces and the smoothness of the resulting solution field.

5.7. Which models provide the best trade-off between accuracy, inference speed and model size for real-time applications?

To determine which models are most suitable for real-time applications, this section compares inference speed, solver speed, model size, and the number of learnable parameters. The goal is to evaluate whether the trained neural operators can provide useful speed-ups over the classical solvers used to generate the data while keeping the model size small enough for practical use in applications such as games and interactive simulations.

5.7.1. Inference speed, speed-up, and model size

This work conducts non-batched inference on 5000 instances and calculates the average inference speed per instance for each model on each mesh. The same is done for the traditional solvers, which in this experiment form a baseline. Since neural operator architectures are designed to directly predict the solution function, they are not limited by solving linear systems of equations that scale with the mesh size and the complexity of the PDE. In this experiment, the exact speed-up achieved by the proposed neural operator architectures over the traditional solvers used for generating the training data is measured. Furthermore, the number of learnable parameters is analysed to determine whether the models are suitable as surrogate models in real-time applications such as games.

Table 5.5: Classical solver generation speeds for Poisson, geodesics, and linear elasticity. Speeds are reported in samples per second.

Problem	Right-hand side	Bunny	Cat
Poisson	Polynomial	1501.02	231.64
	Delta function	2434.31	396.18
	Sinusoidal function	885.90	63.81
	Average	**1607.08**	**230.54**
Geodesics	1 source point	191.06	90.90
	3 source points	204.08	83.34
	5 source points	190.75	77.27
	Average	**195.30**	**83.84**
Elasticity	Normal delta	32.08	9.19
	Hydraulic press	11.61	6.81
	Sinusoidal	32.03	9.15
	Smooth normal bumps	27.52	8.79
	Average	**25.81**	**8.49**

For Poisson, the settings correspond to polynomial, delta, and sinusoidal right-hand sides. For geodesics, they correspond to 1, 3, and 5 source points. For elasticity, they correspond to the four selected force initialisations used in the main elasticity loss table.

The classical elasticity solver is especially slow on the cat mesh, reaching only around 8–9 samples per second for the tested force initialisations. This is below the approximately 30 solves per second that would be needed for one solve per frame in a 30 FPS interactive application, which supports the motivation for learnt surrogate models in real-time settings.

Table 5.6: Average model inference speeds for Poisson, geodesics, and linear elasticity. Speeds are reported in samples per second.

Model	Poisson		Geodesics		Elasticity	
	Bunny	Cat	Bunny	Cat	Bunny	Cat
MLP	2057.67	1957.38	2036.98	1869.10	1846.66	1442.47
DeepONet	1318.61	1293.00	1255.46	1170.57	1313.47	1154.16
Spectral GNN	641.40	650.47	614.25	576.79	641.70	609.61
Hodge Spectral GNN	244.32	244.75	228.71	226.40	248.97	240.56
Graph Neural Network	145.13	104.32	173.87	122.40	135.77	86.21
Multigrid GNN	61.14	57.07	65.57	65.97	63.14	58.13

Table 5.7: Average speed-up relative to the classical solver, together with the number of learnable parameters (reported for the scalar, 1-channel configurations used in the Poisson and geodesics experiments; elasticity uses 3-channel inputs/outputs and therefore has different parameter counts for MLP and DeepONet).

Model	Params Bunny	Params Cat	Poisson speed-up		Geodesics speed-up		Elasticity speed-up	
			Bunny	Cat	Bunny	Cat	Bunny	Cat
Classical solver	–	–	1.00×	1.00×	1.00×	1.00×	1.00×	1.00×
MLP	643,399	2,383,032	**1.28×**	**8.49×**	**10.43×**	**22.29×**	**71.55×**	**170.00×**
DeepONet	325,152	1,191,584	0.82×	**5.61×**	**6.43×**	**13.96×**	**50.89×**	**136.02×**
Spectral GNN	1,249	1,249	0.40×	**2.82×**	**3.15×**	**6.88×**	**24.86×**	**71.85×**
Hodge Spectral GNN	14,377	14,377	0.15×	**1.06×**	**1.17×**	**2.70×**	**9.65×**	**28.35×**
Graph Neural Network	112,033	112,033	0.09×	0.45×	0.68×	**1.12×**	**5.26×**	**10.16×**
Multigrid GNN	579,265	579,265	0.04×	0.25×	0.30×	0.72×	**2.45×**	**6.85×**

These results show that the learnt models scale more favourably with mesh size than the classical solvers. For the Poisson problem, the classical solver is already very fast, so only the MLP provides a speed-up on the bunny mesh. On the larger cat mesh, however, several learnt models become faster than the classical solver. This effect becomes stronger for geodesics, where the classical heat method solver is slower. In this case, the MLP, DeepONet, Spectral GNN, and Hodge Spectral GNN all provide speed-ups on both meshes.

However, speed alone is not sufficient for selecting a model for real-time applications. The MLP and DeepONet are the fastest models, but they also have substantially more parameters and performed worse in the accuracy experiments for the Poisson and geodesics problems. The spectral models are slower, but they require far fewer learnable parameters. The Hodge Spectral GNN gives the strongest geodesic accuracy, while the Spectral GNN provides a good compromise between accuracy, parameter count, and speed. The GNN and Multigrid GNN are generally less favourable in this setting, as they are slower and often do not compensate for this with better accuracy.

6

Discussion and Limitations

6.1. Discussion

The experiments show that neural operator architectures can learn useful approximations of PDE solution operators on fixed triangular surface meshes (Sections 5.5–5.7). However, the results also show that not every model performs well on every problem and every initialisation. The main observation is that the success of a model depends strongly on the structure of the operator being learnt. The Poisson problem, geodesics using the heat method, and linear elasticity each favour different architectural biases. Instead, each model performs well on some right-hand-side distributions and poorly on others. This reinforces that the choice of architecture should depend on the PDE, the mesh setting, and the expected input distribution rather than on a single overall ranking.

Among the tested architectures, the Hodge Spectral Graph Neural Network appears to be the most consistent model across the full set of experiments (e.g., Tables 5.2 and 5.3). It does not achieve the lowest loss in every setting, especially not for the smoother linear elasticity force fields where the MLP performs best, but it remains competitive across all three PDE problems. Unlike several other models, it does not show a clear failure case in the reported accuracy tables. This suggests that the Hodge-based spectral representation provides a robust inductive bias for learning surface-based PDE solution operators on fixed triangular meshes.

For specific problems, the Hodge Spectral Graph Neural Network achieves the highest accuracy on all geodesic initialisations. However, the geodesic initialisations are somewhat similar, since the only varied factor is the number of source points. For the Poisson right-hand sides, it either performs best or second best in all tested settings. This makes it a strong general-purpose model in this thesis, even though it is not always the best specialised choice.

The Poisson results show that spectral representations are especially effective when the PDE operator is closely related to the Laplacian. This is expected, because the Spectral Graph Neural Network and Hodge Spectral Graph Neural Network both use spectral information derived from mesh Laplacians. The Poisson problem itself is also defined through the Laplacian, so the spectral basis provides a strong inductive bias for this task. In particular, polynomial and sinusoidal right-hand sides are learnt accurately because the corresponding Poisson solutions are smooth and dominated by lower-frequency structure. The Poisson solve has a smoothing effect, so models that can represent smooth global functions are strongly favoured.

The delta-function right-hand sides are much harder. This is visible in both the Poisson and linear elasticity experiments. A delta function provides information at only one vertex, but the model must still predict a full solution field over the entire mesh. This requires the model to learn how every possible source location influences every other vertex. In the fixed-mesh setting, this is similar to learning many columns of an inverse operator. When the mesh becomes larger, there are more possible source locations, but the amount of training data does not necessarily cover all of them sufficiently. This likely explains why localised inputs generalise poorly compared to smoother input fields.

The geodesic task is harder to learn than the smoother Poisson settings, but it does not show the same kind of extreme failure behaviour as the Poisson delta-function cases. This suggests that the difficulty is more consistent across models rather than being dominated by a small number of outliers.

Although the heat method uses a heat solve and a Poisson solve, the full mapping from source points to geodesic distance fields is more complex than the Poisson problem. The heat method includes gradient normalisation and the resulting distance fields contain sharper geometric transitions, especially when multiple source points are present. The strong performance of the Hodge Spectral Graph Neural Network suggests that the Hodge-based representation is useful for problems involving differential structure on the surface.

The model comparison also shows a clear difference between the MLP-based baselines and the more geometry-aware models. For Poisson and geodesics, the more complex models generally outperform the MLP and DeepONet. This suggests that explicitly using mesh structure or spectral information is important for scalar surface PDEs where the solution depends strongly on global surface geometry. However, this trend does not hold for linear elasticity. In the elasticity experiments, the MLP-based models, especially the standard MLP, are surprisingly dominant for the smoother force initialisations. This suggests that, for the tested linear elasticity setting, explicit mesh geometry is less decisive than expected. Since the mesh, material parameters, and force distributions are fixed, the MLP can specialise directly to the finite-dimensional force-to-displacement mapping.

This does not mean that vector-valued fields are necessarily harder to learn than scalar fields. Linear elasticity uses a three-dimensional force vector as input and a three-dimensional displacement vector as output, while Poisson and geodesics use scalar fields. Despite this, many elasticity predictions are visually accurate, especially for smooth force fields. Therefore, the difficulty is not only determined by the number of channels. The smoothness, locality, and structure of the input-output mapping appear to matter more than whether the field is scalar-valued or vector-valued.

The losses should be interpreted together with the qualitative visualisations. In particular, scalar-field errors and displacement-field errors are not always visually comparable: a moderate relative loss can still preserve the main deformation pattern, while small spatial shifts in scalar isolines can be visually noticeable.

DeepONet shows another important qualitative behaviour. Its predictions tend to be smooth, even when it does not obtain the lowest loss. This can make DeepONet visually plausible in some settings, especially for smooth elasticity fields. However, this smoothness is not always enough for high accuracy. DeepONet uses embedded three-dimensional coordinates as query locations, but these coordinates describe the object in ambient space rather than along the mesh surface. For surface PDEs, the intrinsic geometry is determined by distances and connectivity along the surface. Two vertices can be close in three-dimensional space while being far apart over the mesh. Therefore, the coordinate embedding does not fully encode the surface geometry needed for Poisson or geodesic problems.

The graph-based models show a different kind of behaviour. The GNN and Multigrid GNN sometimes produce visually reasonable predictions, especially in cases where the global structure of the solution is captured. However, their isolines often appear less smooth and more oscillatory than those of the spectral models. This suggests that they can approximate the broad solution pattern but may introduce local noise or high-frequency artefacts. This is important for real-time applications because a slightly inaccurate but smooth prediction may be more visually acceptable than a prediction with noisy local artefacts.

The Spectral Graph Neural Network provides one of the clearest examples of a useful but specialised inductive bias. It performs very well on Poisson, where the scalar Laplacian basis is directly aligned with the PDE. However, it performs much worse on linear elasticity compared to the MLP-based models. This suggests that a scalar Laplacian spectral basis is not automatically optimal for every surface PDE. Linear elasticity involves vector-valued displacement, material behaviour, and shell deformation effects, so the scalar Laplacian basis may not capture the most relevant structure for this operator.

Finally, the speed results show that neural operators can be useful as surrogate approximators, especially when the classical solver is expensive or the mesh is larger. For Poisson, the classical solver is already fast, so the speed-up advantage is limited. For geodesics and especially linear elasticity, the speed-ups are more substantial. This supports the third research question: neural operators can provide a useful trade-off between accuracy and inference speed when many repeated solves are needed on the same mesh. However, the best practical model depends on the application. If accuracy is the priority, the spectral and Hodge spectral models are more reliable for Poisson and geodesics. If speed is the priority and the expected inputs are smooth elasticity force fields, the MLP can be a strong surrogate despite its simpler architecture.

6.2. Limitations

6.2.1. Fixed mesh setting

While the fixed-mesh setting is mainly framed as a solution that enables the MLP, DeepONet, and spectral-based models to learn a solution operator specialised for a mesh, it implies that in real-world applications, each mesh could require a model. This could increase the application size because a separate trained parameter set may need to be stored for each mesh. The practical storage cost depends on the chosen architecture, parameter precision, and compression.

6.2.2. PDE choices

This work only evaluates a few PDEs. Since the difference in results is quite large, it is infeasible to draw conclusions about other PDEs based on these results. New PDEs have to be tested first in controlled environments before being applied in real-time applications.

6.2.3. DeepONet

DeepONet is implemented using 3D coordinates for the query locations. Although this is sensible for volume-based PDE solving, it is less appropriate for surface-based PDE solving. Due to the encoding of the query locations not representing either connectivity or geodesic distances but only raw distances, it could be that the model learns that, in many parts of a mesh, the connectivity and surface are implied by the distance between the vertices. This likely causes the model to pass information through the volume rather than along the surface. This may be the reason that many results from DeepONet appear to be correct but are shifted by some bias.

6.2.4. Datasets

Since this research generates datasets automatically rather than using a fixed public benchmark dataset, the exact results may differ between dataset generations. This is especially relevant because no fixed random seeds were used during dataset generation. Regenerating the datasets would therefore not necessarily produce exactly the same training and test samples, which could affect training stability, test losses, and grid-search hyper-parameter choices. A larger test set is used to make the reported evaluation metrics less dependent on individual test samples, but it does not fully remove variation caused by generating a different dataset.

6.2.5. Right-hand sides

The choice of right-hand sides is somewhat arbitrary and not based on real-life data. This matters because the models may perform well partly because the right-hand sides follow the synthetic distributions used during training and testing. In practical applications, inputs may be driven by contact forces, user interaction, time-varying loads, collisions, or other structured physical signals that differ substantially from the synthetic initialisations used here. Therefore, the reported results should be interpreted as controlled proof-of-concept behaviour rather than guaranteed performance on arbitrary real application data.

6.2.6. Grid-search

In this thesis, grid-search was only performed for the Poisson problem. The resulting model hyper-parameters obtained from grid-search have been reused for geodesics and linear elasticity as well. Due to time and computation constraints, grid-search could not be performed for geodesics and linear elasticity. A dedicated grid-search for these tasks could improve the reported results.

6.2.7. Elasticity shell parameters

The linear elasticity experiments in this thesis vary the applied force fields but keep the Reissner–Mindlin shell configuration fixed. In particular, the shell thickness, Young's modulus, and Poisson ratio are kept constant across all elasticity datasets. As a result, the reported elasticity results only show how the models perform for one fixed material and shell configuration. Different material parameters could change the scale, smoothness, and spatial structure of the resulting displacement fields, which may affect the relative performance of the evaluated architectures.

6.3. Future work

A key direction for future work is to study generalisation beyond the fixed data distributions used in this thesis. The experiments in this work train and evaluate models on fixed meshes and on right-hand side distributions generated in a controlled manner. Future work could test whether these models generalise to unseen right-hand side patterns, unseen input distributions, and new mesh geometries. Extending the approach to new meshes would be especially important for broader practical use, but would also require addressing the fact that several of the models in this thesis depend on a fixed number of vertices and a fixed vertex ordering. Future work could also test discretisation invariance more directly by training and evaluating models across multiple resolutions of the same geometry, for example by using subdivided versions of the same mesh. This would show whether graph-based and spectral models can transfer across discretisations rather than only specialise in one fixed vertex set.

Another important direction is to extend the set of PDE problems. This thesis considers three surface-based problems: the Poisson problem, geodesic distance approximation using the heat method, and linear elasticity. These problems provide a useful comparison between scalar fields, distance fields, and vector-valued displacement fields, but they do not cover the full range of PDEs used in simulation and geometry processing. Future work could include time-dependent PDEs, non-linear PDEs, coupled physical systems, or PDEs where several interacting fields must be predicted at the same time.

For the linear elasticity experiments, future work could also investigate a broader range of physical settings. This thesis keeps the Reissner–Mindlin shell configuration fixed and varies only the force field. Different Young's moduli, Poisson ratios, shell thicknesses, or constraint choices could lead to different solution behaviour and may affect which model architectures perform best. Evaluating these variations would help determine whether the conclusions for elasticity remain valid beyond the fixed shell setting used in this work.

Finally, future work should evaluate more realistic and application-driven right-hand sides. The current datasets are generated using synthetic initialisation strategies, which are useful for controlled experiments but may not fully represent the force fields or source distributions encountered in real applications. For example, practical simulation settings may involve contact forces, collision events, user interactions, external constraints, or force fields derived from recorded simulation traces. Training and evaluating on such data would clarify how well the models transfer from proof-of-concept experiments to practical geometry processing or real-time simulation pipelines.

7

The Broader Field of Computer Science

7.1. Reflection on Research in the Broader CS Field

Traditional numerical solvers for PDEs are designed to produce accurate solutions for individual problem instances. However, in applications where many similar problems must be solved repeatedly on the same domain, the cost of repeatedly running a solver can become a bottleneck. Neural operators approach this problem from an amortised computation perspective. Neural operators are similar to approximation algorithms. They do not guarantee a perfect replacement for traditional solvers; instead, they provide a learnt approximation of a traditional solver. This causes neural operator research to relate significantly to the trade-off between accuracy and runtime cost, which is widely studied in the field of computer science. Rather than doing heavy computation when it is needed, neural operators train offline, and their datasets can also be generated offline. This results in a faster surrogate model at runtime.

In this way, this research is also related to the field of computer graphics. Many graphics applications, such as games and simulations, need real-time approximations of physical and geometrical behaviour. In these settings, perfect accuracy is not always the priority. These applications focus more on speed, stability, and visually plausible results. Within computer graphics and geometric data processing, there are already many approximation algorithms used; for example, the heat method for geodesics (Section 3.3). A game developer would much rather use the heat method than exact geodesics, as exact geodesics are significantly harder to compute [5] and may not justify the additional cost. For the same reasons, a developer may also opt to use a neural operator architecture to directly predict these geodesics, which can be even faster than the already optimised heat method, as shown by the results of this research.

The same can be said for geometric data processing. Many algorithms in geometric data processing rely on PDEs, especially surface PDEs, which this work focuses on as well. Mesh smoothing, deformations, and many other geometric data processing applications could use neural operator architectures to provide approximate results.

From the machine learning perspective, neural operators extend deep learning research, such as neural networks, non-linear activation functions, transformers, and graph neural networks, to the field of operator learning. With the help of suitable methods that allow models to learn mappings between function spaces rather than finite-dimensional vectors, many well-researched deep learning methods can and have already been applied to this field of operator learning. Yet, there are many more new architectural designs and optimisation methods to be discovered. Feed-forward neural networks, graph neural networks, recurrent neural networks, transformers, and even diffusion models have already been applied to neural operator research.

7.2. Reflection on Implications and Applicability

7.2.1. Neural operators in general

The main benefit of neural operators is the possibility of amortised computation. Generating training data and training the model can be expensive, but this cost is paid offline. Once trained, the model can evaluate new inputs much faster than a traditional solver. This is especially useful when many related PDE problems must be solved repeatedly. In such settings, the initial cost of data generation and training may be justified by the speed-up during later use.

This has potential value in many application areas. In climate and weather modelling, fast surrogate models such as FourCastNet have shown that data-driven neural operators can support high-resolution forecasting and rapid large-ensemble prediction, which is useful for uncertainty estimation and scenario exploration [23]. In healthcare and biological modelling, surrogate solvers could make certain simulations more practical in time-sensitive workflows [24, 25, 26, 27]. In engineering, structural analysis, and design, they could allow designers to test many structural or physical variations more quickly [13, 28, 29]. In fusion research, faster models of plasma behaviour could support design exploration or real-time monitoring [30, 31]. These examples show that neural operators are not only a machine learning technique but also a possible tool for accelerating scientific and engineering workflows.

However, these benefits come with risks. A neural operator learns from examples, and therefore inherits the assumptions and limitations of the data used to train it. If the training data do not cover certain physical regimes, boundary conditions, material parameters, or input patterns, the model may perform poorly in those cases. This is especially dangerous when the model is used in settings where incorrect predictions have real-world consequences. For this reason, neural operators should be seen as complements to classical solvers rather than unconditional replacements. Classical solvers remain essential for generating training data, validating predictions, and handling cases where certified accuracy is required.

7.2.2. Applicability of this thesis

The specific contribution of this thesis is narrower than neural operator research in general. This work studies PDE solution operators on fixed triangular surface meshes. The fixed-mesh setting limits generality, but it also makes the approach practically relevant for applications where the same geometry is reused many times. This is common in computer graphics, games, interactive simulations, and geometry processing, where a fixed object may be evaluated repeatedly under different input conditions.

In such applications, the main benefit is runtime speed. If a model can approximate a Poisson solve, geodesic distance computation, or linear elasticity solve quickly, it may allow PDE-based effects to be used in interactive settings where classical solvers are too expensive. For example, a game or design tool could use a trained model to provide fast approximations of surface deformation, distance fields, or other geometry-dependent quantities. In these contexts, exact physical correctness is not always required; speed, stability, and visual plausibility may be more important.

The results of this thesis support this possible use case, but only under specific conditions. The models are trained and evaluated on fixed meshes and controlled right-hand side distributions. Therefore, the results do not imply that the models can automatically generalise to arbitrary new meshes, materials, or force fields. This is an important limitation for practical deployment. A model that performs well on one mesh may not work on another mesh, and a model trained on synthetic inputs may not perform equally well on real application data.

The main risks of this specific work are therefore related to overgeneralisation and overtrust. In a game or visual application, a wrong prediction may only lead to visual artefacts or physically implausible behaviour. In a design or engineering tool, however, a visually plausible but inaccurate prediction could mislead users if it is interpreted as a reliable physical simulation. This is especially relevant for localised inputs, unusual force patterns, or cases outside the training distribution, since the experiments in this thesis show that performance can vary strongly between different PDEs and right-hand side types.

Another practical concern is that fixed-mesh models may require separate training for each object or geometry. This could make the approach less useful when many different meshes are needed, unless future work improves generalisation across meshes. The fixed-mesh setting should therefore be understood as a specialised surrogate-solver setting: it is most useful when the same mesh is reused many times and when the cost of training can be justified by the number of later evaluations.

7.2.3. Stakeholders

Different stakeholders are affected by this research in different ways. For researchers in neural operators, scientific machine learning, and geometric deep learning, this thesis provides a controlled comparison of multiple model families on surface-based PDE problems. The results may help guide future work on which architectural biases are useful for different PDEs, input distributions, and output fields.

For developers in computer graphics, games, and interactive simulation, the main benefit is the possibility of faster runtime approximations. These stakeholders may be interested in neural operators if they allow more advanced PDE-based effects to be used interactively. However, they would also need to consider the cost of data generation, model training, model storage, and validation.

For engineers and designers, the benefits and risks are more balanced. Fast surrogate models could make design iteration more efficient, but only if users understand that the predictions are approximate. In this context, neural operators should be used for exploration, previews, or repeated low-risk evaluations unless their accuracy has been carefully validated for the specific application.

For end users, the effect is indirect. They may benefit from faster simulations, more responsive tools, or richer interactive experiences. However, they may also be affected by errors if a surrogate model is used in a context where incorrect physical predictions influence decisions. This reinforces the need to communicate the limitations of learnt surrogate solvers clearly.

Overall, this thesis contributes to the broader computer science field by studying how learnt surrogate models can approximate PDE solution operators on triangular surface meshes. The work supports the idea that neural operators can be useful for repeated-query settings, especially in real-time graphics and interactive simulation. At the same time, it shows that accuracy depends strongly on the PDE, the model architecture, and the input distribution. Responsible use therefore requires treating neural operators not as universal replacements for classical solvers, but as fast approximate tools whose reliability must be evaluated in the context where they are used.

8

Conclusion

This thesis investigated whether neural operator architectures can learn solution operators for partial differential equations on fixed triangular surface meshes. The main goal was to determine whether trained neural operator models can approximate the mapping from an input field on a mesh to the corresponding PDE solution field, without running a classical numerical solver for every new input. To study this, several model families were compared: a standard multilayer perceptron, DeepONet, a graph neural network, a multigrid graph neural network, a spectral graph neural network, and a Hodge spectral graph neural network. These models were evaluated on three surface-based PDE tasks: the Poisson problem, geodesic distance approximation using the heat method, and surface linear elasticity using a Reissner–Mindlin shell formulation.

The results show that neural operator models can learn solution operators on fixed triangular meshes, but the quality of the approximation depends strongly on the PDE, the input field, and the model architecture. For the Poisson problem, the spectral models achieved the strongest results. This is consistent with the structure of the problem, since the Poisson equation is closely related to the mesh Laplacian and the spectral models explicitly use a Laplacian-based representation. The polynomial and sinusoidal right-hand sides were learnt most accurately, while the delta right-hand side was more difficult because it requires the model to map a highly localised input to a global solution field. This indicates that smooth input functions are easier to learn, while localised forcing terms require more expressive models.

The geodesic experiments showed that learning geodesic distance fields using the heat method is more difficult than learning the Poisson solution operator. Although the heat method itself contains Poisson solves, the full mapping from source points to geodesic distance fields is more complex and non-linear. The spectral models still performed best overall, but their errors were higher than in the Poisson experiments. In particular, the Hodge spectral graph neural network performed better than the standard spectral graph neural network for the geodesic task, suggesting that the Hodge-based representation can be more robust for more complex surface-based operators. However, the visualisations also showed that the learnt geodesic fields can contain shifts, noise, or inaccurate isolines, meaning that this problem remains challenging for the tested architectures.

The linear elasticity experiments extend the evaluation from scalar-valued PDE outputs to vector-valued displacement fields. In this task, the input is a three-dimensional force vector at every vertex, and the output is a three-dimensional displacement vector at every vertex. This makes the problem different from the Poisson and geodesic experiments, both in terms of physical interpretation and output structure. The elasticity results show a different trend from the scalar PDE tasks. For smoother force initialisations, such as hydraulic press forces, sinusoidal forces, and smooth normal bumps, the standard MLP achieved the lowest loss on both meshes. This suggests that these displacement fields are smooth and structured enough for a coordinate-based fixed-size model to learn effectively. In these cases, the simplicity of the MLP was not a disadvantage, and it even outperformed the more geometry-aware models.

The normal delta force initialisation was the hardest elasticity setting. This is similar to the Poisson delta right-hand side, since a highly localised input force can cause a global response over the surface. In this case, the MLP performed best on the bunny mesh, while the Hodge spectral graph neural

network performed best on the larger cat mesh. This suggests that localised elastic loads are harder to approximate and that spectral information can become more useful when the mesh is larger or when the deformation pattern depends more strongly on global surface structure. Overall, the elasticity results show that the best model is not determined only by the PDE type. It also depends on the smoothness, locality, and spatial structure of the input field and the resulting solution field.

In terms of model comparison, there is no single architecture that performs best in all settings. Spectral models are the strongest choice for the Poisson problem and remain competitive for geodesics. The Hodge spectral graph neural network is especially useful for the more difficult geodesic task and for the localised linear elasticity case on the larger mesh. The MLP is much weaker for several Poisson and geodesic settings, but it is surprisingly strong for many elasticity settings. DeepONet is generally fast and stable, but it does not consistently achieve the best accuracy. The graph neural network and multigrid graph neural network did not provide the best overall trade-off in these experiments. Although their mesh-based structure is theoretically attractive, their accuracy and speed did not consistently outperform the simpler or spectral alternatives in the fixed-mesh setting.

The speed experiments show that neural operators can provide substantial inference speed-ups over classical solvers, especially when the classical solver is relatively expensive. For the Poisson problem, the solver is already fast, so the speed-up advantage is limited, especially on the smaller bunny mesh. For geodesics and linear elasticity, the advantage becomes more pronounced. The largest speed-ups occur in the linear elasticity experiments, where all learnt models were faster than the classical solver on both meshes. The MLP and DeepONet achieved the highest speed-ups, while the spectral models provided a more balanced trade-off between accuracy, model size, and speed. This supports the use of neural operators as surrogate solvers in settings where many repeated solves are required after an offline training phase.

For real-time applications such as games and interactive simulations, these results are promising but also highlight important trade-offs. If speed is the main priority and the expected input fields are smooth, a simple MLP can be a strong option, especially for linear elasticity. If accuracy is more important, spectral models are generally more reliable, particularly for Poisson and geodesic tasks. The spectral graph neural network provides a strong compromise between accuracy, parameter count, and inference speed, while the Hodge spectral graph neural network can be preferable for more difficult or more geometry-dependent tasks. Therefore, the practical choice of model should depend on the target PDE, the expected input distribution, and the acceptable balance between speed and accuracy.

In conclusion, this work shows that neural operator architectures can learn useful approximations of PDE solution operators on fixed triangular surface meshes. The experiments demonstrate accurate results for several Poisson and linear elasticity settings, more limited but still meaningful results for geodesic distance approximation, and significant inference speed-ups over classical solvers in the more expensive tasks. The main conclusion is that neural operators are a promising approach for amortising repeated PDE solves on fixed mesh surfaces, but the best architecture depends strongly on the PDE and the structure of the input and output fields.

9 Use of AI

9.1. Code

9.1.1. Models

I implemented the evaluated model families, with DeepONet, graph neural networks, and the multigrid hierarchy developed with reference to [32]. The spectral graph neural network was implemented by me based on an explanation created with AI assistance. The Hodge Spectral Graph Neural Network implementation was developed with substantial AI assistance and then reviewed, tested, and integrated by me.

AI assistance was also used during development for performance profiling, refactoring, and debugging suggestions, which helped improve the efficiency of the graph-based models and identify implementation issues.

9.1.2. Additional code

The training pipeline, grid-search, model utilities, training loops, mesh and data loading utilities, function value initialisations, data generation scripts, and the Poisson, heat-method geodesics, and linear elasticity solvers were implemented by me. AI assistance was used selectively for code review and for improving clarity and maintainability.

Plotting and data-visualisation scripts (e.g., PyVista, Matplotlib) were drafted with AI assistance and then adapted and integrated by me.

Elasticity data generation uses NGSolve, following the tutorial in [33], which makes use of the TDNNS method [34] for the Reissner–Mindlin shell. AI assistance was used to refactor this implementation into a reusable class and to improve runtime performance.

9.2. Paper

Certain sections were drafted or revised with AI assistance and then reviewed and edited, including tables, mathematical formulas, and reference entries, as well as revisions of sections originally written by me. AI also assisted with structuring parts into subsections, identifying inaccuracies in the written text, evaluating the sections critically, explaining what should be included in each section, and providing general ideation on how to write a section. Apart from that, Overleaf's AI suggestions for formulation errors were utilised.

References

[1] Jean Donea and Antonio Huerta. *Finite Element Methods for Flow Problems*. Wiley, 2003. ISBN: 9780471496663. DOI: 10.1002/0470013826.

[2] Olek C. Zienkiewicz, Robert L. Taylor, and David A. Fox. *The Finite Element Method for Solid and Structural Mechanics*. 7th ed. Oxford: Elsevier, 2013. ISBN: 9780080951362.

[3] Allen Taflove and Susan C. Hagness. *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. 3rd ed. Norwood, MA: Artech House, 2005. ISBN: 9781580538329.

[4] Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Society for Industrial and Applied Mathematics, 2007. ISBN: 9780898716290. DOI: 10.1137/1.9780898717839.

[5] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. "Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow". In: *ACM Transactions on Graphics* 32.5 (2013), 152:1–152:11. DOI: 10.1145/2516971.2516977.

[6] Thorsten Kurth et al. "FourCastNet: Accelerating Global High-Resolution Weather Forecasting using Adaptive Fourier Neural Operators". In: *arXiv preprint arXiv:2208.05419* (2022). DOI: 10.48550/arXiv.2208.05419. arXiv: 2208.05419.

[7] Zongyi Li et al. "Fourier Neural Operator for Parametric Partial Differential Equations". In: *International Conference on Learning Representations (ICLR)* (2021). arXiv: 2010.08895.

[8] Lu Lu, Pengzhan Jin, and George Em Karniadakis. "DeepONet: Learning Nonlinear Operators for Identifying Differential Equations Based on the Universal Approximation Theorem of Operators". In: *Nature Machine Intelligence* 3 (2021), pp. 218–229. arXiv: 1910.03193.

[9] Tianping Chen and Hong Chen. "Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems". In: *IEEE Transactions on Neural Networks* 6.4 (1995), pp. 911–917. DOI: 10.1109/72.392253.

[10] Tapas Tripura and Souvik Chakraborty. "Wavelet Neural Operator: A Neural Operator for Parametric Partial Differential Equations". In: *arXiv preprint arXiv:2205.02191* (2022).

[11] Miguel Liu-Schiaffini et al. "Neural Operators with Localized Integral and Differential Kernels". In: *Proceedings of the 41st International Conference on Machine Learning (ICML)*. Vol. 235. Proceedings of Machine Learning Research. Vienna, Austria, 2024.

[12] Boris Bonev et al. "Spherical Fourier Neural Operators: Learning Stable Dynamics on the Sphere". In: *Proceedings of the 40th International Conference on Machine Learning (ICML)* (2023). arXiv: 2306.03838.

[13] Zongyi Li et al. "Fourier Neural Operator with Learned Deformations for PDEs on General Geometries". In: *Journal of Machine Learning Research* 24.388 (2023), pp. 1–26. arXiv: 2207.05209.

[14] Zongyi Li et al. "Geometry-Informed Neural Operator for Large-Scale 3D PDEs". In: *arXiv preprint arXiv:2309.00583* (2023).

[15] Zongyi Li et al. "Neural Operator: Graph Kernel Network for Partial Differential Equations". In: *arXiv preprint arXiv:2003.03485* (2020).

[16] Bo Pang et al. "Learning the Geodesic Embedding with Graph Neural Networks". In: *ACM Transactions on Graphics* 42.6 (2023), pp. 1–12. DOI: 10.1145/3618317.

[17] Guangya Zhang et al. "A Graph-Structured, Physics-Informed DeepONet Neural Network for Complex Structural Analysis". In: *Machine Learning and Knowledge Extraction* 7.4 (2025), p. 137.

[18] Subhankar Sarkar and Souvik Chakraborty. "Spatio-Spectral Graph Neural Operator for Solving Computational Mechanics Problems on Irregular Domain and Unstructured Grid". In: *arXiv preprint arXiv:2409.00604* (2024).

[19] Math Insight. *Introducing a Rabbit-Predator Model*. URL: https://mathinsight.org/introducing_rabbit_predators (visited on 05/02/2026).

[20] Zongyi Li et al. "Multipole Graph Neural Operator for Parametric Partial Differential Equations". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6755–6766. arXiv: 2006.09535.

[21] Zhouhui Lian and Afzal A. Godil. "SHREC'11 Track: Shape Retrieval on Non-rigid 3D Watertight Meshes". In: *Eurographics Workshop on 3D Object Retrieval*. Accessed: 2026-05-20. Llandudno, United Kingdom, 2011. URL: https://catalog.data.gov/dataset/shrec11-track-shape-retrieval-on-non-rigid-3d-watertight-meshes.

[22] Stanford Computer Graphics Laboratory. *Stanford Bunny Mesh*. https://graphics.stanford.edu/~mdfisher/Data/Meshes/bunny.obj. Accessed: 2026-05-20.

[23] Jaideep Pathak et al. "FourCastNet: A Global Data-Driven High-Resolution Weather Model Using Adaptive Fourier Neural Operators". In: *arXiv preprint arXiv:2202.11214* (2022).

[24] Minglang Yin et al. "A scalable framework for learning the geometry-dependent solution operators of partial differential equations". In: *Nature Computational Science* 4 (2024), pp. 928–940. DOI: 10.1038/s43588-024-00732-2.

[25] Ehsan Naghavi et al. "Rapid prediction of cardiac activation in the left ventricle with geometric deep learning: a step towards cardiac resynchronization therapy planning". In: *npj Digital Medicine* 9 (2026), p. 225. DOI: 10.1038/s41746-026-02399-7.

[26] Huaiqian You et al. "A Physics-Guided Neural Operator Learning Approach to Model Biological Tissues From Digital Image Correlation Measurements". In: *Journal of Biomechanical Engineering* 144.12 (2022), p. 121012. DOI: 10.1115/1.4055918.

[27] Zhijun Zeng et al. *Neural Born Series Operator for Biomedical Ultrasound Computed Tomography*. 2023. DOI: 10.48550/arXiv.2312.15575. arXiv: 2312.15575 [eess.IV].

[28] David Erzmann and Sören Dittmer. "Equivariant neural operators for gradient-consistent topology optimization". In: *Journal of Computational Design and Engineering* 11.3 (2024), pp. 91–100. DOI: 10.1093/jcde/qwae039.

[29] Jangseop Park and Namwoo Kang. "Point-DeepONet: Predicting Nonlinear Fields on Non-Parametric Geometries under Variable Load Conditions". In: *Neural Networks* 198 (2026), p. 108560. DOI: 10.1016/j.neunet.2026.108560.

[30] Vignesh Gopakumar et al. "Plasma Surrogate Modelling using Fourier Neural Operators". In: *Nuclear Fusion* 64.5 (2024), p. 056025. DOI: 10.1088/1741-4326/ad313a.

[31] N. Carey et al. *Neural operator surrogate models of plasma edge simulations: feasibility and data efficiency*. 2025. DOI: 10.48550/arXiv.2502.17386. arXiv: 2502.17386 [physics.plasm-ph].

[32] Jean Kossaifi et al. "A Library for Learning Neural Operators". In: *arXiv preprint arXiv:2412.10354* (2025). DOI: 10.48550/arXiv.2412.10354. URL: https://arxiv.org/abs/2412.10354.

[33] NGSolve. *Linear Kirchhoff-Love and Reissner-Mindlin shells*. https://docu.ngsolve.org/ngs24/SaS/linear_KL_RM_shell_HHJ_TDNNS.html. Accessed: 2026-05-04. 2024.

[34] Michael Neunteufel and Joachim Schöberl. "The Hellan–Herrmann–Johnson and TDNNS methods for linear and nonlinear shells". In: *Computers & Structures* 305 (2024), p. 107543. DOI: 10.1016/j.compstruc.2024.107543.

A

Qualitative Visualisations

This appendix shows representative (typical test-set) qualitative visualisations for all evaluated models, grouped by model and then by problem.

A.1. MLP

A.1.1. Poisson

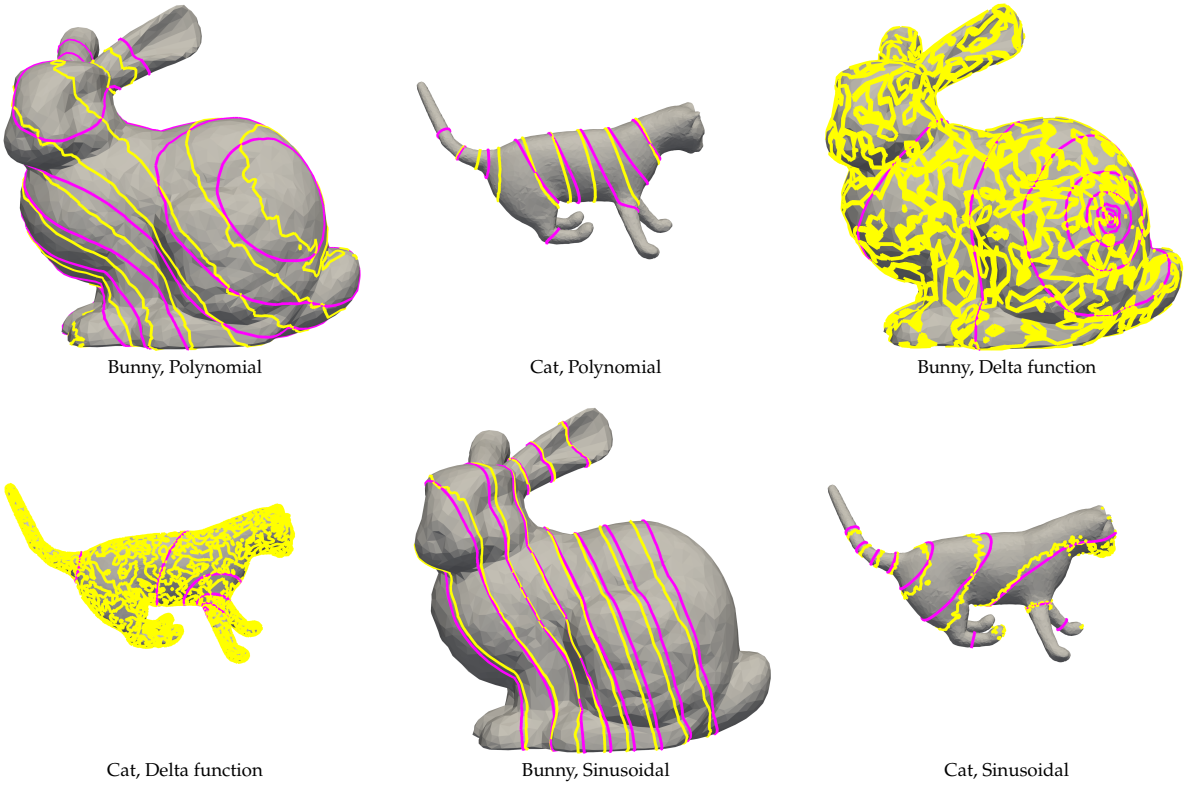
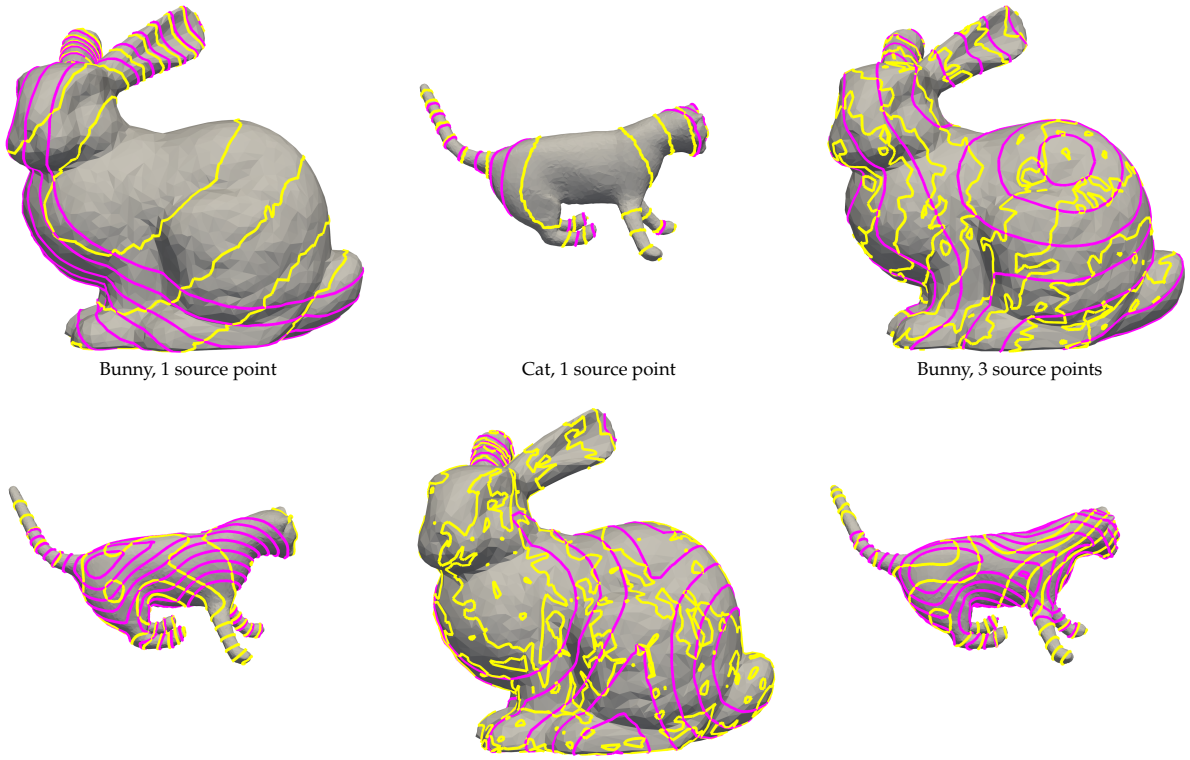


Figure A.1: Representative Poisson visualisations for MLP.

A.1.2. Geodesics

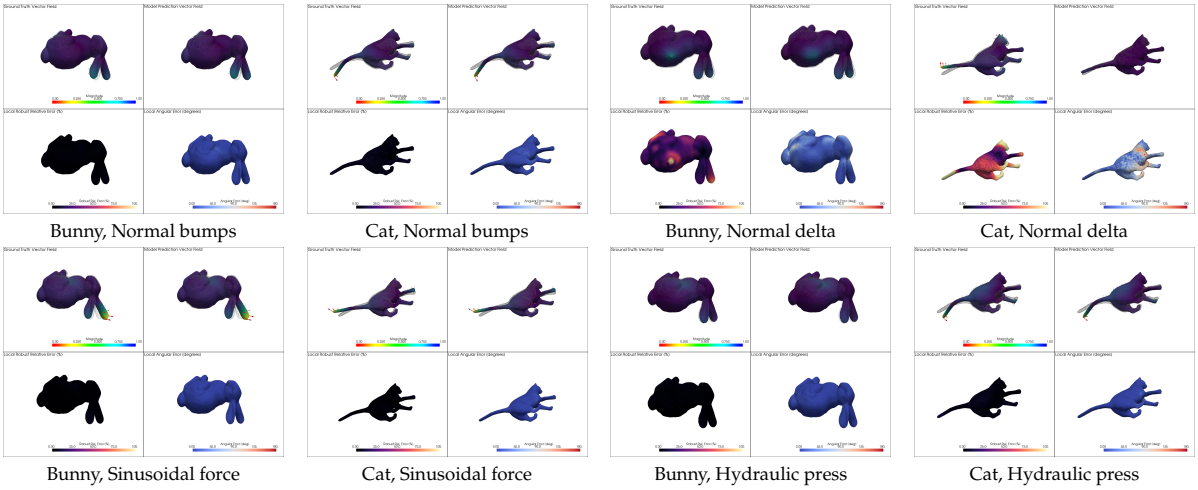


Bunny, 1 source point | Cat, 1 source point | Bunny, 3 source points

Cat, 3 source points | Bunny, 5 source points | Cat, 5 source points

Figure A.2: Representative geodesics visualisations for MLP.

A.1.3. Linear elasticity



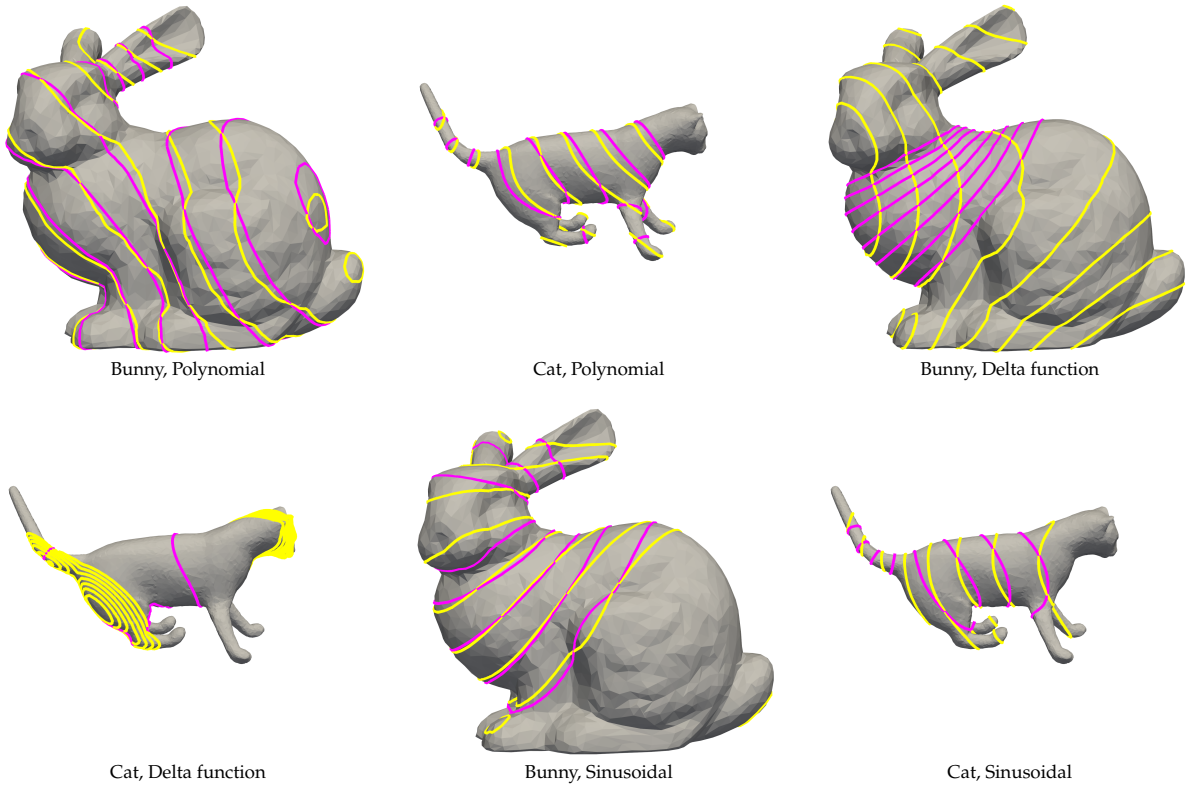
Bunny, Normal bumps | Cat, Normal bumps | Bunny, Normal delta | Cat, Normal delta

Bunny, Sinusoidal force | Cat, Sinusoidal force | Bunny, Hydraulic press | Cat, Hydraulic press

Figure A.3: Representative linear elasticity visualisations for MLP.

A.2. DeepONet

A.2.1. Poisson



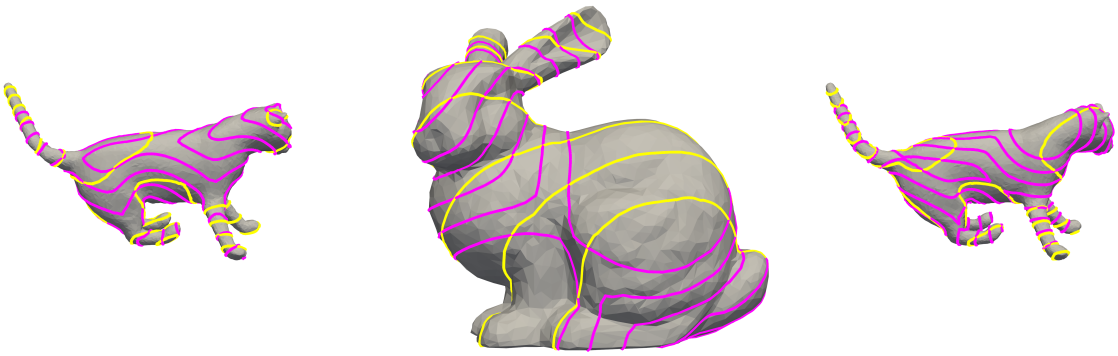
Bunny, Polynomial | Cat, Polynomial | Bunny, Delta function

Cat, Delta function | Bunny, Sinusoidal | Cat, Sinusoidal

Figure A.4: Representative Poisson visualisations for DeepONet.

A.2.2. Geodesics

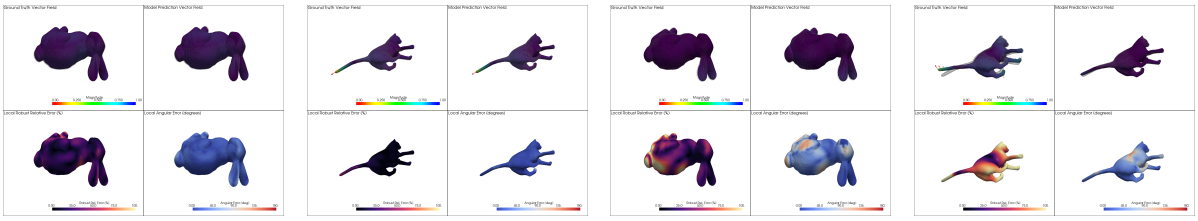
Bunny, 1 source point | Cat, 1 source point | Bunny, 3 source points



Cat, 3 source points | Bunny, 5 source points | Cat, 5 source points

Figure A.5: Representative geodesics visualisations for DeepONet.

A.2.3. Linear elasticity



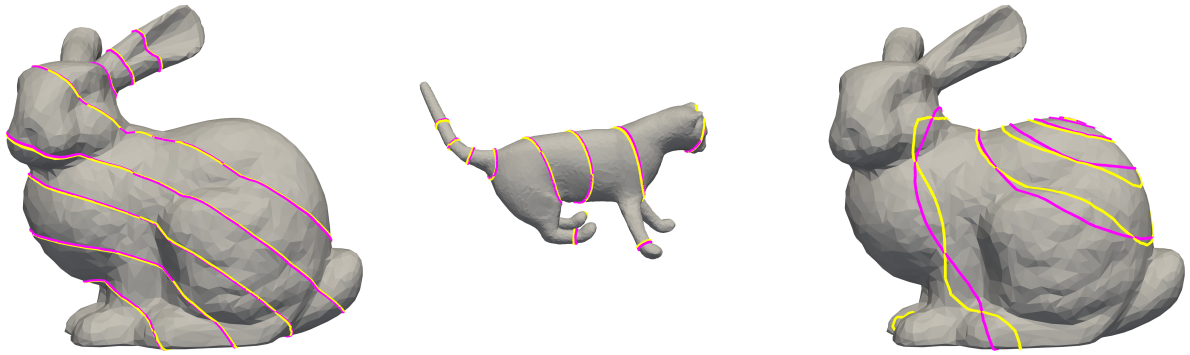
Bunny, Normal bumps | Cat, Normal bumps | Bunny, Normal delta | Cat, Normal delta

Bunny, Sinusoidal force | Cat, Sinusoidal force | Bunny, Hydraulic press | Cat, Hydraulic press

Figure A.6: Representative linear elasticity visualisations for DeepONet.

A.3. Spectral GNN

A.3.1. Poisson

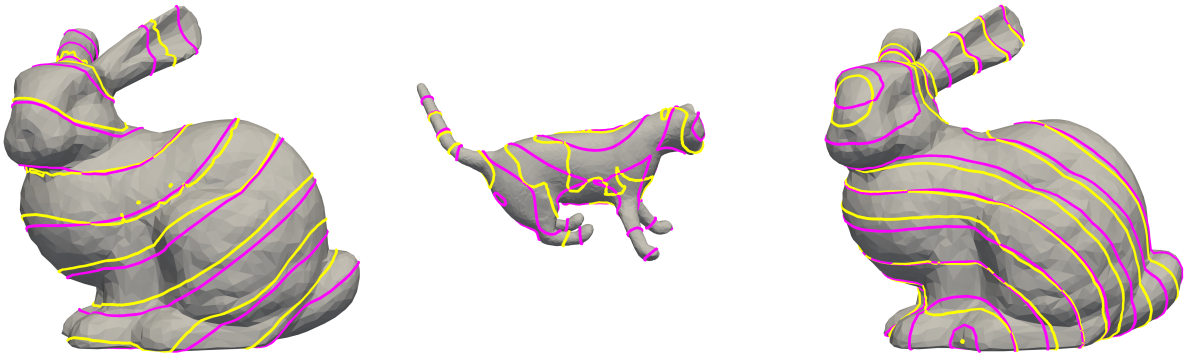


Bunny, Polynomial | Cat, Polynomial | Bunny, Delta function

Cat, Delta function | Bunny, Sinusoidal | Cat, Sinusoidal

Figure A.7: Representative Poisson visualisations for Spectral GNN.

A.3.2. Geodesics

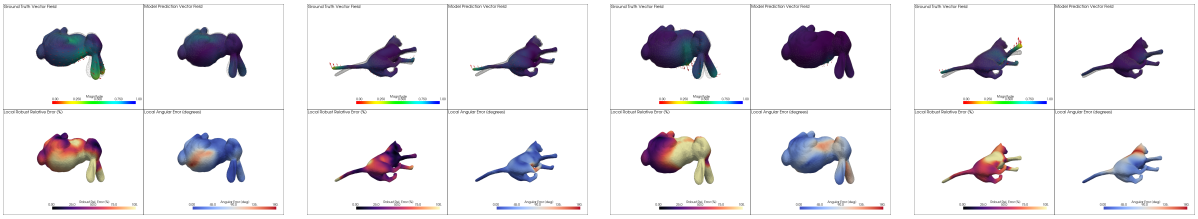


Bunny, 1 source point | Cat, 1 source point | Bunny, 3 source points

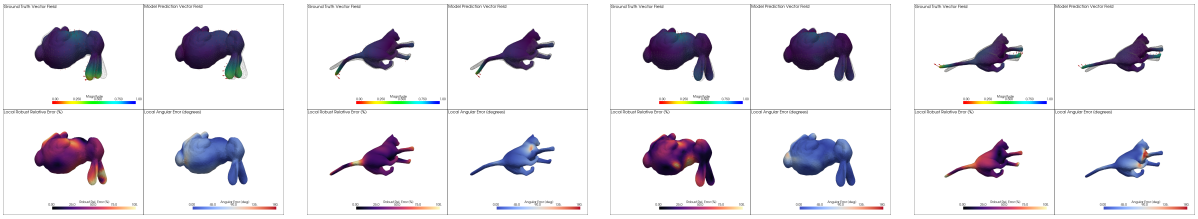
Cat, 3 source points | Bunny, 5 source points | Cat, 5 source points

Figure A.8: Representative geodesics visualisations for Spectral GNN.

A.3.3. Linear elasticity



Bunny, Normal bumps | Cat, Normal bumps | Bunny, Normal delta | Cat, Normal delta

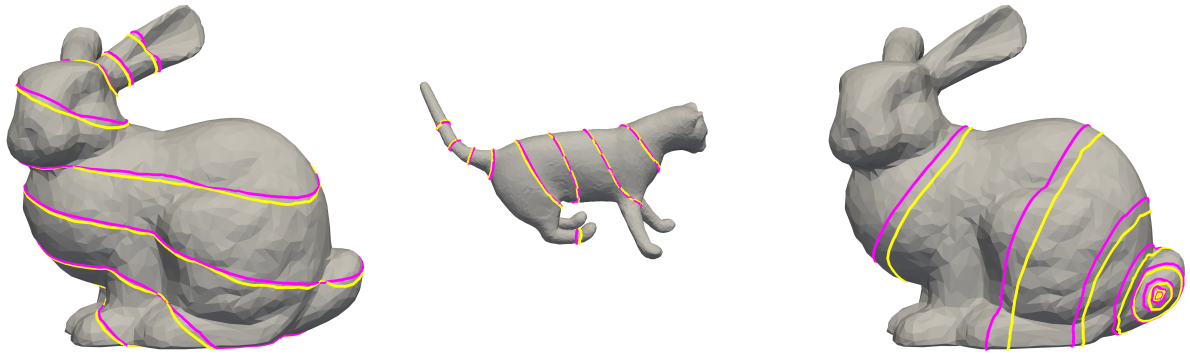


Bunny, Sinusoidal force | Cat, Sinusoidal force | Bunny, Hydraulic press | Cat, Hydraulic press

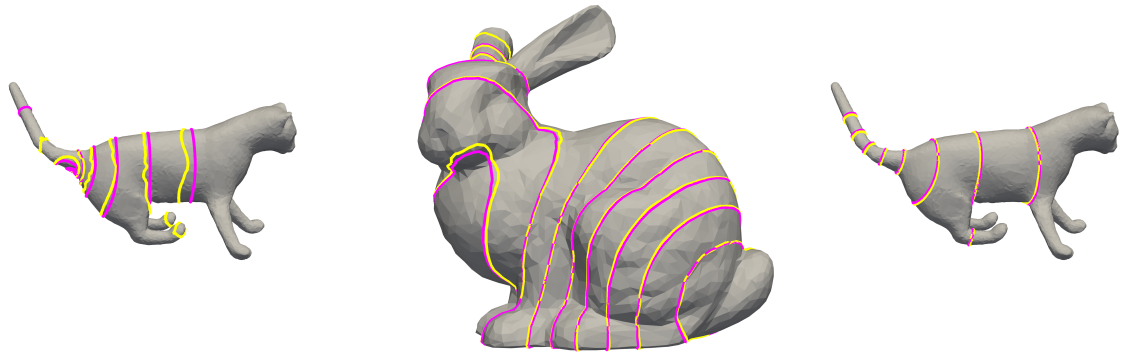
Figure A.9: Representative linear elasticity visualisations for Spectral GNN.

A.4. Hodge Spectral GNN

A.4.1. Poisson



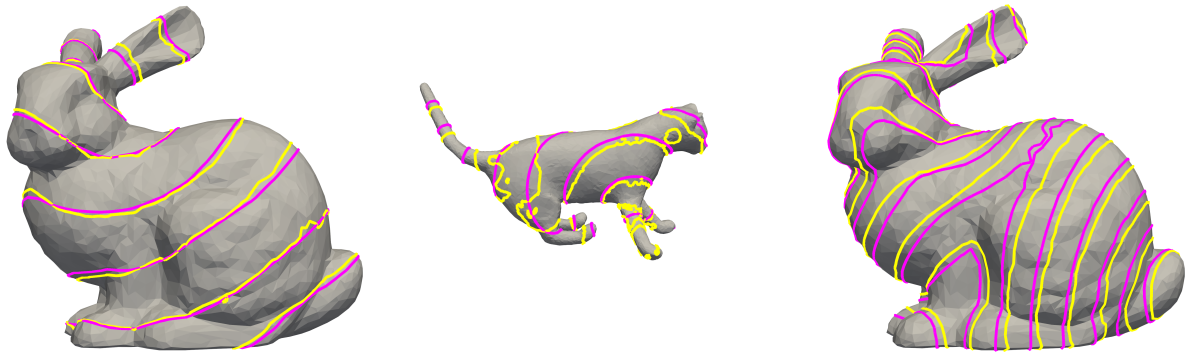
Bunny, Polynomial | Cat, Polynomial | Bunny, Delta function



Cat, Delta function | Bunny, Sinusoidal | Cat, Sinusoidal

Figure A.10: Representative Poisson visualisations for Hodge Spectral GNN.

A.4.2. Geodesics

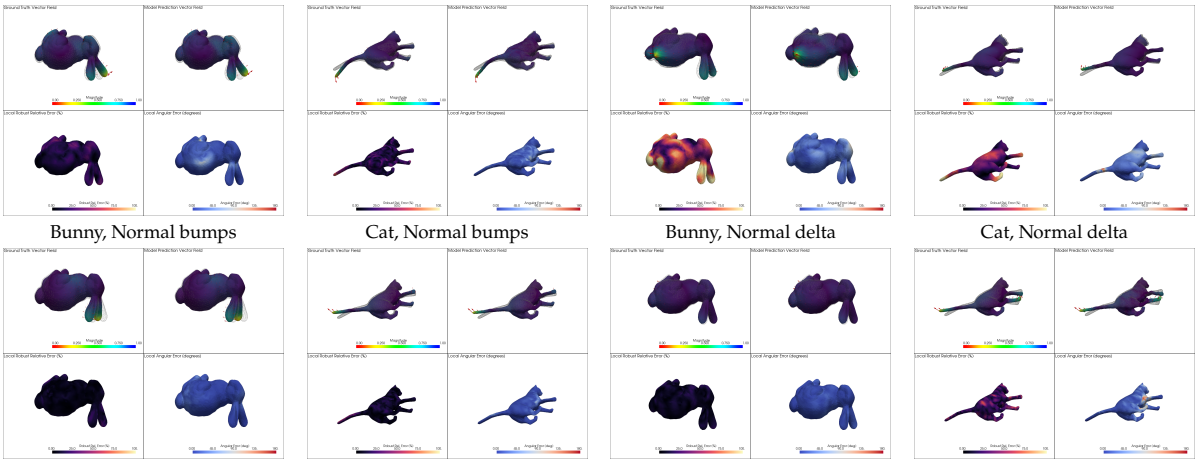


Bunny, 1 source point | Cat, 1 source point | Bunny, 3 source points

Cat, 3 source points | Bunny, 5 source points | Cat, 5 source points

Figure A.11: Representative geodesics visualisations for Hodge Spectral GNN.

A.4.3. Linear elasticity



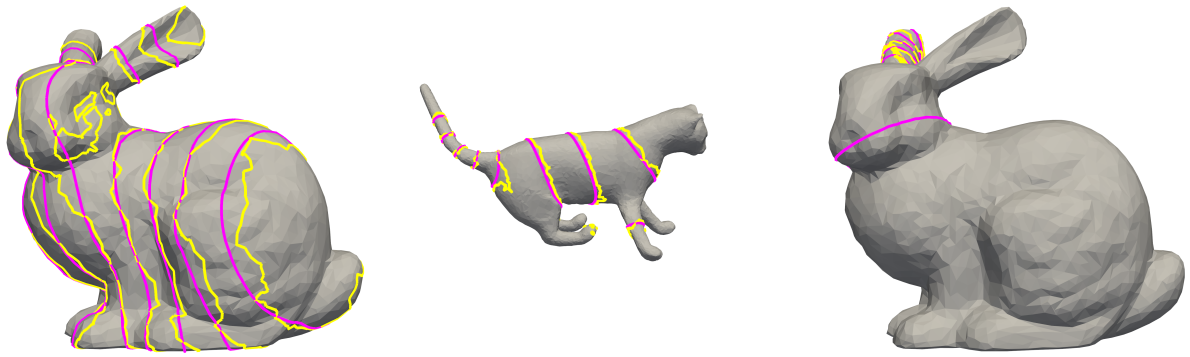
Bunny, Normal bumps | Cat, Normal bumps | Bunny, Normal delta | Cat, Normal delta

Bunny, Sinusoidal force | Cat, Sinusoidal force | Bunny, Hydraulic press | Cat, Hydraulic press

Figure A.12: Representative linear elasticity visualisations for Hodge Spectral GNN.

A.5. GNN

A.5.1. Poisson



Bunny, Polynomial | Cat, Polynomial | Bunny, Delta function

Cat, Delta function | Bunny, Sinusoidal | Cat, Sinusoidal

Figure A.13: Representative Poisson visualisations for GNN.

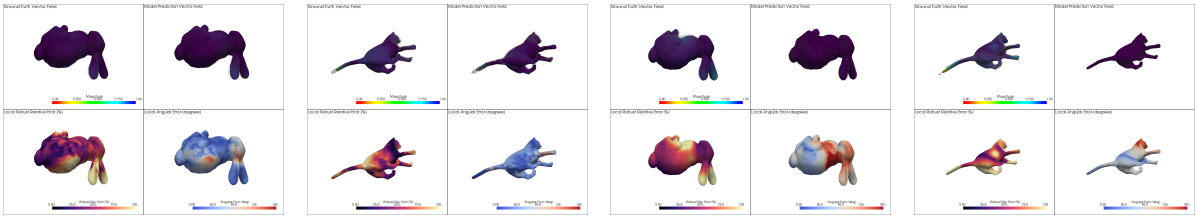
A.5.2. Geodesics

Bunny, 1 source point | Cat, 1 source point | Bunny, 3 source points

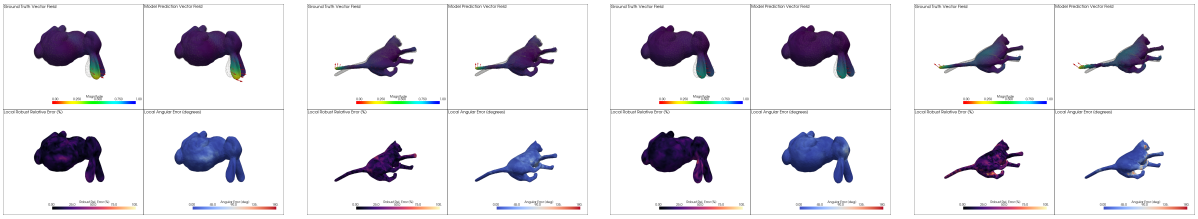
Cat, 3 source points | Bunny, 5 source points | Cat, 5 source points

Figure A.14: Representative geodesics visualisations for GNN.

A.5.3. Linear elasticity



Bunny, Normal bumps | Cat, Normal bumps | Bunny, Normal delta | Cat, Normal delta

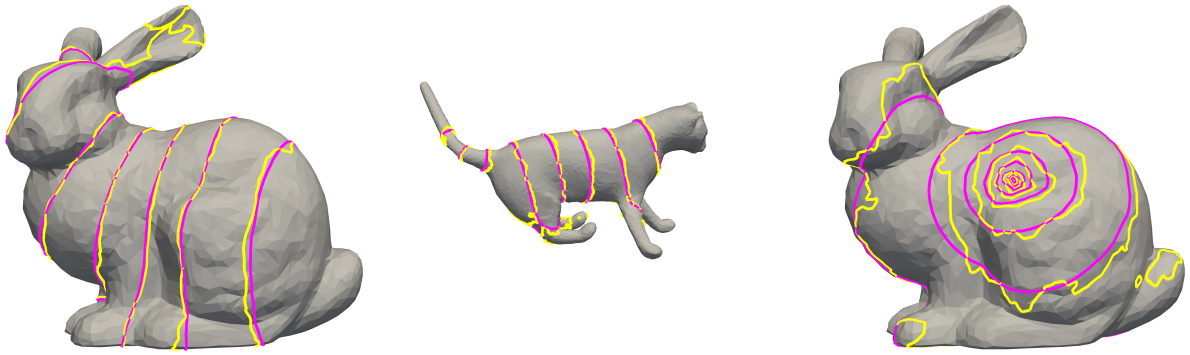


Bunny, Sinusoidal force | Cat, Sinusoidal force | Bunny, Hydraulic press | Cat, Hydraulic press

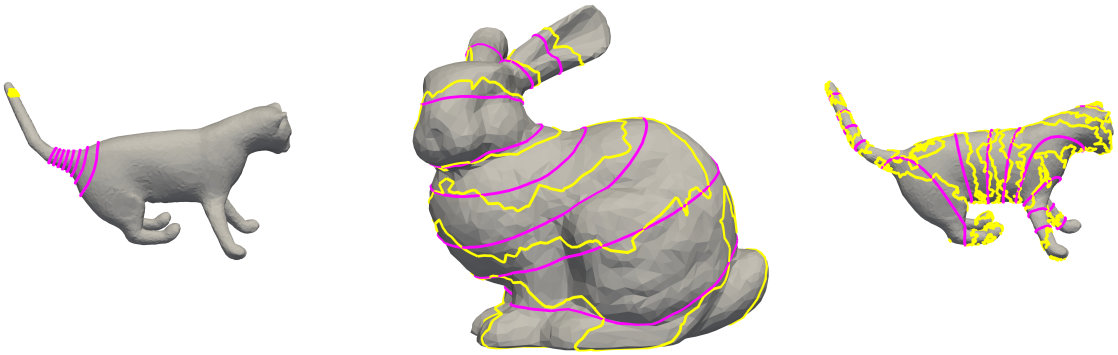
Figure A.15: Representative linear elasticity visualisations for GNN.

A.6. Multigrid GNN

A.6.1. Poisson



Bunny, Polynomial | Cat, Polynomial | Bunny, Delta function

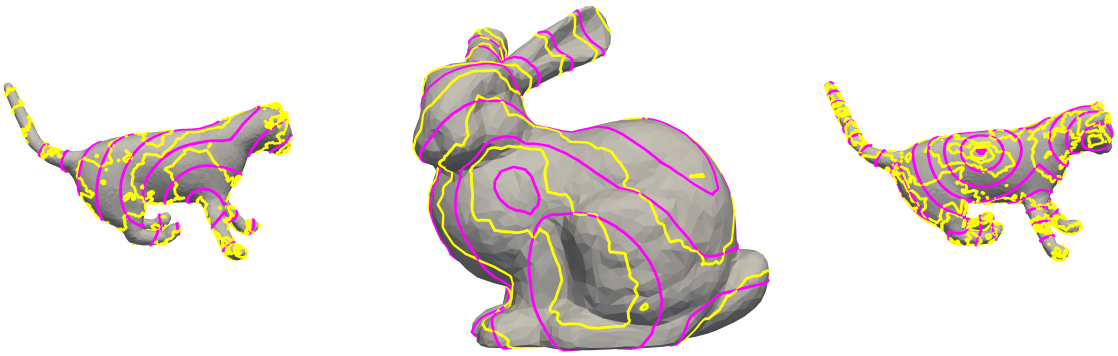


Cat, Delta function | Bunny, Sinusoidal | Cat, Sinusoidal

Figure A.16: Representative Poisson visualisations for Multigrid GNN.

A.6.2. Geodesics

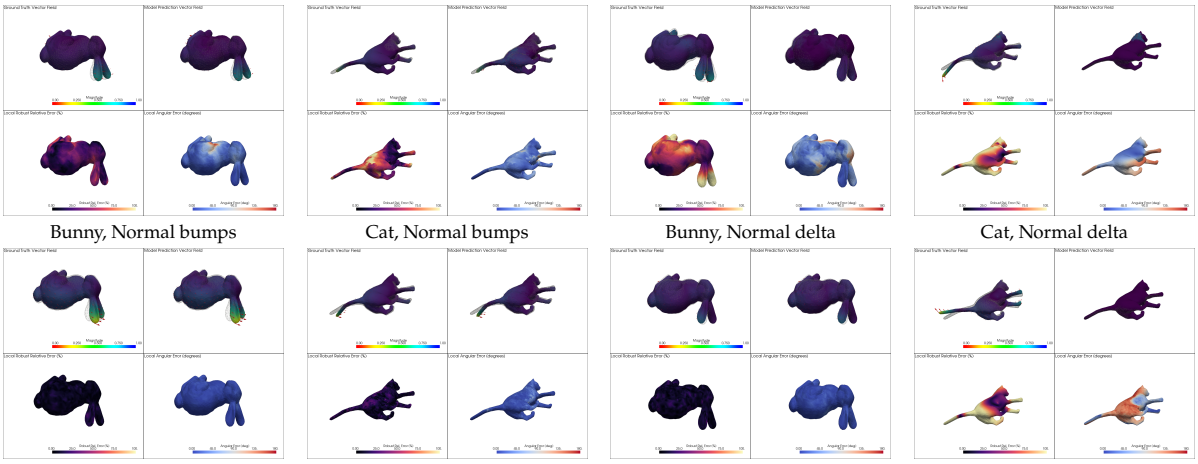
Bunny, 1 source point | Cat, 1 source point | Bunny, 3 source points



Cat, 3 source points | Bunny, 5 source points | Cat, 5 source points

Figure A.17: Representative geodesics visualisations for Multigrid GNN.

A.6.3. Linear elasticity



Bunny, Normal bumps | Cat, Normal bumps | Bunny, Normal delta | Cat, Normal delta

Bunny, Sinusoidal force | Cat, Sinusoidal force | Bunny, Hydraulic press | Cat, Hydraulic press

Figure A.18: Representative linear elasticity visualisations for Multigrid GNN.

A.7. Representative and worst-case test losses

The qualitative visualisations in this appendix focus on representative test-set predictions. These representative examples are useful for understanding typical model behaviour and for visually comparing the predicted and ground-truth solution fields. However, representative examples do not show how large the error can become on difficult test instances. To complement the representative visualisations, Figures A.19, A.20, and A.21 compare representative and worst-case test losses.

It should be emphasised that this thesis is not primarily focused on worst-case guarantees, uncertainty estimation, or explicit error-mitigation strategies. The main goal of the experiments is to compare neural operator architectures in terms of accuracy, qualitative prediction behaviour, inference speed, and model size. Nevertheless, the recorded test losses also make it possible to inspect worst-case behaviour. This provides additional insight into model robustness, which is especially relevant when considering whether such models could be used in real-time applications.

In the figures below, each model is shown with two bars per setting: a darker front bar for the representative loss and a lighter background bar for the worst-case loss. Both values are averaged over the bunny and cat meshes. This makes it possible to compare not only how large the worst-case errors can become, but also how those errors relate to the corresponding representative performance.

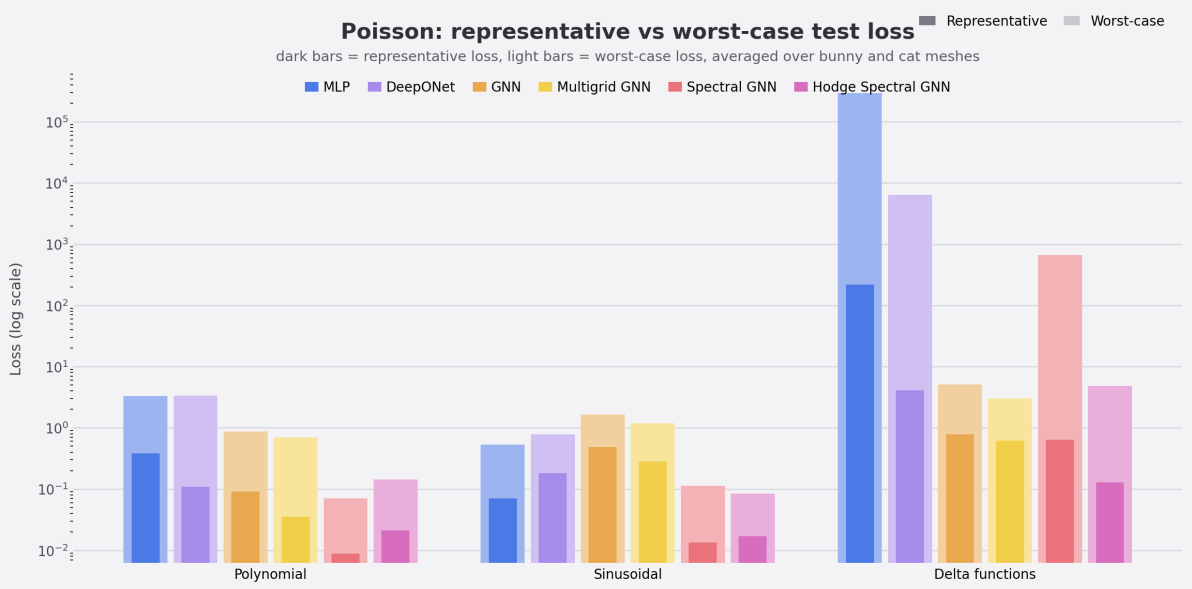


Figure A.19: Representative and worst-case test losses for the Poisson problem, averaged over the bunny and cat meshes. For each model and right-hand side, the darker bar shows the representative loss and the lighter bar shows the worst-case loss. The vertical axis is shown on a logarithmic scale, because the delta-function right-hand side leads to extremely large worst-case losses for several models. Without logarithmic scaling, these outliers would dominate the plot and make the remaining bars difficult to interpret.

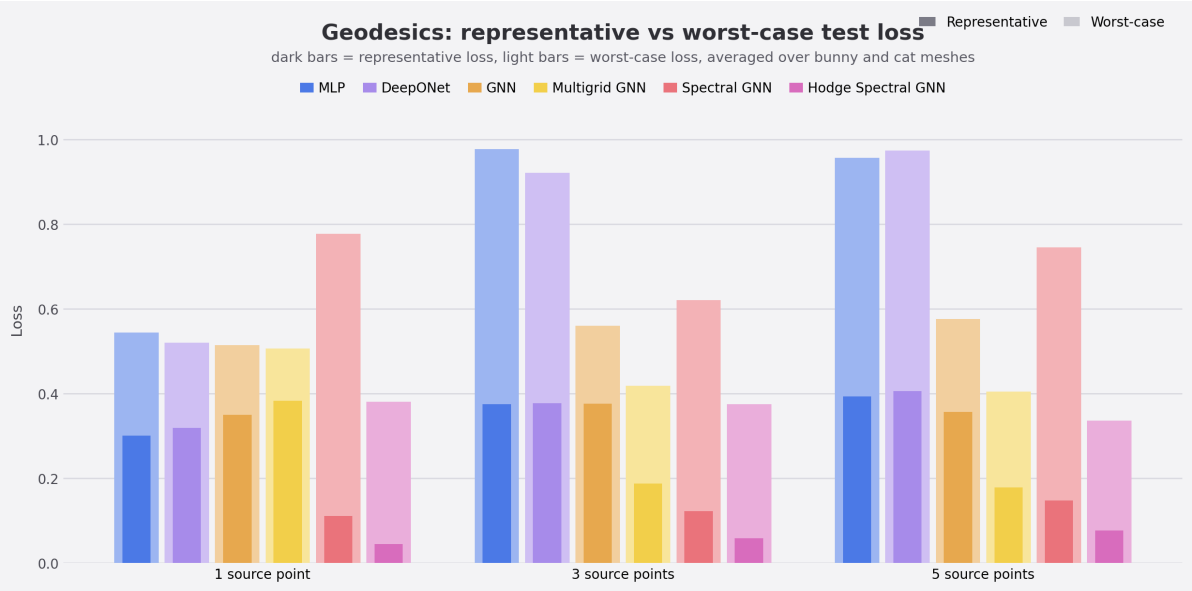


Figure A.20: Representative and worst-case test losses for the geodesics problem, averaged over the bunny and cat meshes. For each model and number of source points, the darker bar shows the representative loss and the lighter bar shows the worst-case loss. In contrast to the Poisson delta-function setting, the losses here remain within a more moderate range, making a standard linear scale sufficient.

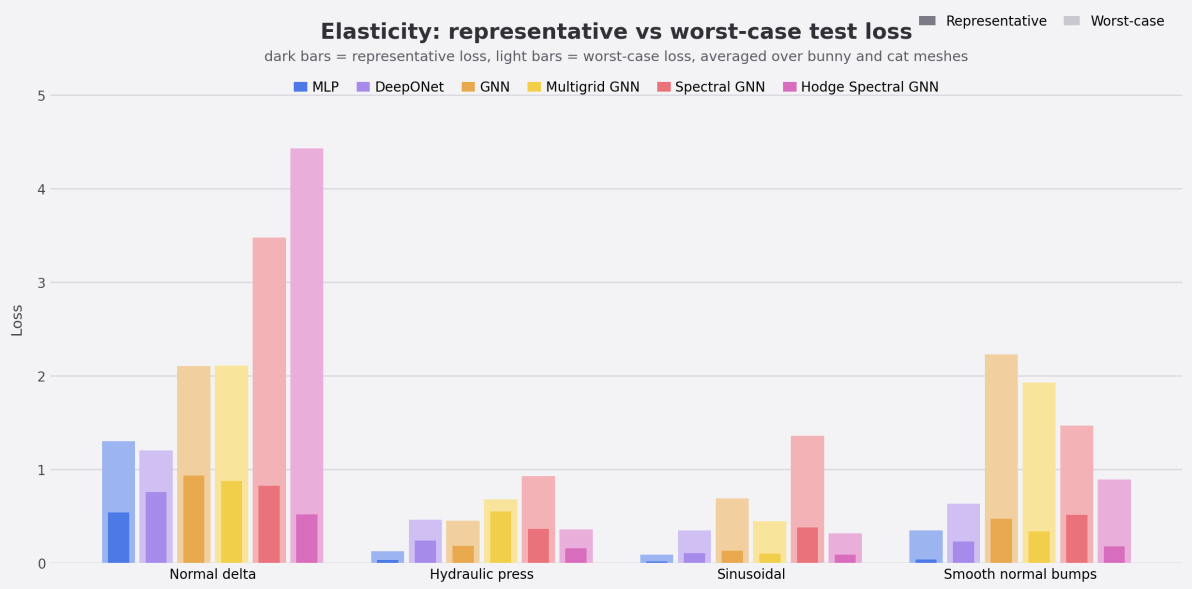


Figure A.21: Representative and worst-case test losses for the linear elasticity problem, averaged over the bunny and cat meshes. For each model and force initialisation, the darker bar shows the representative loss and the lighter bar shows the worst-case loss. These visualisations help indicate whether a model remains stable across the test set or whether some difficult inputs can lead to noticeably larger errors than the representative examples suggest.

Interpretation

Several useful observations can be made from these visualisations. First, the Poisson results show that highly localised delta-function right-hand sides are clearly the most problematic in terms of worst-case behaviour. For several models, the worst-case loss becomes much larger than the representative loss, indicating that occasional large failures can occur even when the representative predictions appear acceptable. This suggests that these models are not reliable choices for real-time applications that require reasonably accurate predictions for strongly localised delta-like inputs.

Second, the visualisations also show that a large worst-case amplification does not automatically imply poor absolute prediction quality compared to every other model. In some settings, the worst-case loss of one model can still be comparable to, or even lower than, the representative loss of another model. This is important when interpreting the results: robustness and absolute accuracy are related, but they are not the same. A model may be less robust in the sense that its error varies more across the test set while still achieving strong absolute performance in typical cases.

Third, for smoother or less localised initialisations, the difference between representative and worst-case loss is often much smaller. This can be seen in several polynomial and sinusoidal Poisson cases, in many geodesics cases, and in multiple elasticity settings such as hydraulic press, sinusoidal forces, and smooth normal bumps. In these cases, the worst-case loss often remains in the same general range as the representative loss, which indicates that model behaviour is more stable across the test set. This suggests that, for such inputs, some of the models are relatively robust and are unlikely to produce very inaccurate predictions on isolated test instances.

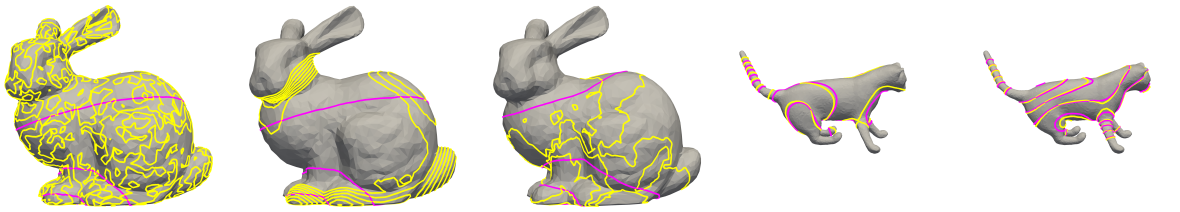
Finally, the figures highlight that robustness differs between model families. Some models show only a modest increase from representative to worst-case loss, whereas others show a much larger gap. This means that two models with similar representative accuracy can still differ substantially in how safely they generalise across unseen test samples. These visualisations therefore provide an additional perspective on model quality: not only how accurate a model is on a typical prediction, but also how much that accuracy can deteriorate on difficult cases.

A.8. Selected worst-case qualitative visualisations

The previous appendix figures focus mainly on representative test-set examples and on aggregated representative versus worst-case losses. This section adds a small set of selected worst-case visualisations. These examples are not meant as a complete worst-case analysis or as an error-mitigation study. Instead, they show what some of the numerical worst cases look like visually. This is useful because a single loss

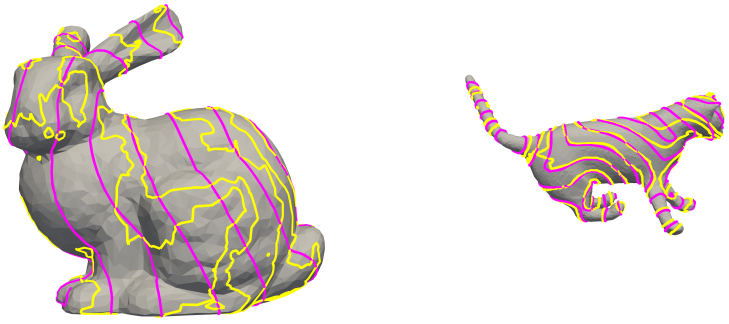
value does not always make it clear whether the resulting prediction is still usable in practice.

The examples are selected to show both failure cases and robust cases. For Poisson, several delta-function examples are included because the delta right-hand side is the most localised setting and leads to the largest worst-case errors. In contrast, the polynomial and sinusoidal Poisson examples show that some spectral models can remain accurate even in their worst observed test case. The geodesics and elasticity examples show the same contrast: some worst cases remain visually reasonable, while others indicate that the model can fail on particular inputs.



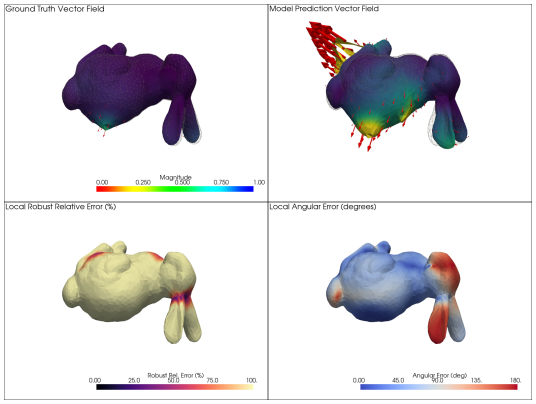
MLP, delta | DeepONet, delta | Multigrid GNN, delta | Spectral GNN, polynomial | Hodge Spectral GNN, sinusoidal

Figure A.22: Selected worst-case Poisson visualisations. The first three panels show worst cases for the delta-function right-hand side. The MLP and DeepONet examples are included to show how the largest delta-function failures can look visually, while the Multigrid GNN example shows that even the model with the lowest worst-case delta loss can still produce a prediction that would be unsuitable when high accuracy is required. The final two panels show the opposite behaviour: the Spectral GNN on the polynomial right-hand side and the Hodge Spectral GNN on the sinusoidal right-hand side remain visually accurate even in their worst observed test case, indicating much more robust behaviour for these smoother Poisson settings.

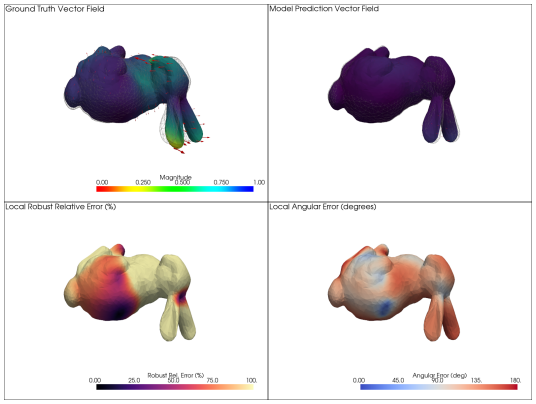


Multigrid GNN, 3 sources | Hodge Spectral GNN, 5 sources

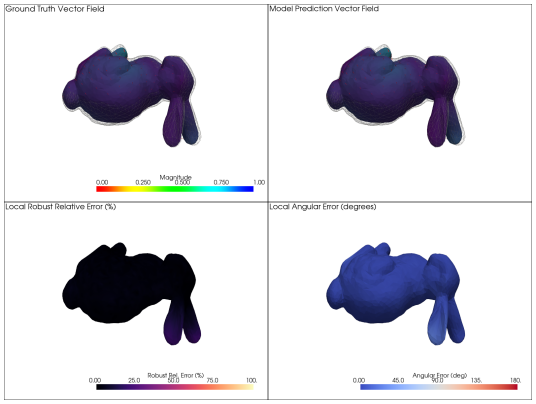
Figure A.23: Selected worst-case geodesics visualisations. The Multigrid GNN example is included because it has the highest selected worst-case loss for the three-source-point geodesics setting, showing a case where the predicted isolines deviate noticeably from the ground truth. The Hodge Spectral GNN example is included as a contrasting robust case: even for five source points, its worst-case prediction remains comparatively accurate, which supports the quantitative observation that the Hodge Spectral GNN is the most reliable model for the geodesics task.



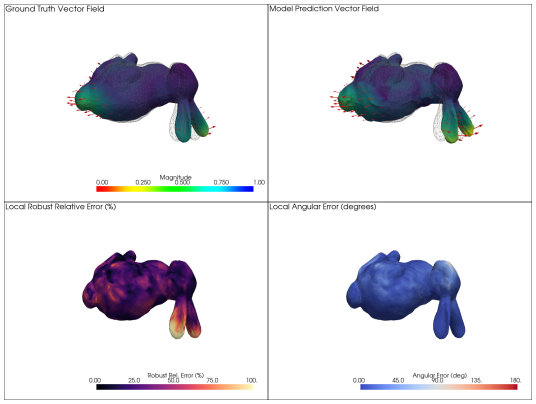
Hodge Spectral GNN, normal delta



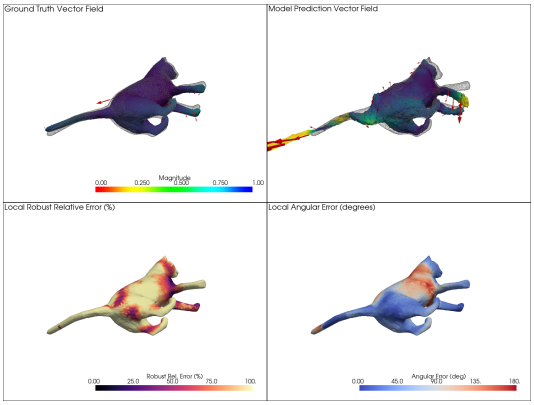
DeepONet, normal delta



MLP, sinusoidal force



Multigrid GNN, hydraulic press



GNN, sinusoidal force

Figure A.24: Selected worst-case linear elasticity visualisations. The Hodge Spectral GNN and DeepONet normal-delta cases are included to compare a stronger and weaker model on the most localised elasticity force initialisation. The MLP sinusoidal-force and Multigrid GNN hydraulic-press examples show that some worst-case elasticity predictions can still remain accurate and visually usable. The GNN sinusoidal-force case is included as a counterexample: it shows that inaccurate worst-case behaviour is not limited to delta-like inputs, since a smoother sinusoidal force can also lead to a visibly poor prediction for some models.