Learning the Solution Operator of Boundary Value Problems using Graph Neural Networks

Anonymous Authors[1]

Abstract

As an alternative to classical numerical solvers for partial differential equations (PDEs) subject to boundary value constraints, there has been a surge of interest in investigating neural networks that can solve such problems efficiently. In this work, we design a general solution operator for two different time-independent PDEs using graph neural networks (GNNs) and spectral graph convolutions. We train the networks on simulated data from a finite elements solver on a variety of shapes and inhomogeneities. In contrast to previous works, we focus on the ability of the trained operator to generalize to previously unseen scenarios. Specifically, we test generalization to meshes with different shapes and superposition of solutions for a different number of inhomogeneities. We find that training on a diverse dataset with lots of variation in the finite element meshes is a key ingredient for achieving good generalization results in all cases. With this, we believe that GNNs can be used to learn solution operators that generalize over a range of properties and produce solutions much faster than a generic solver. Our dataset, which we make publicly available, can be used and extended to verify the robustness of these models under varying conditions.

1. Introduction

Graph Neural Networks (GNNs) (Scarselli et al., 2009) have recently seen a plethora of new applications ranging from molecule and protein property modeling (Jumper et al., 2021; Henderson et al., 2021; Maziarka et al., 2020), learning complex dynamics from data (Battaglia et al., 2016; Pfaff et al., 2021) to combinatorial optimization (Cappart et al., 2021). Many real-world applications produce data artifacts that are naturally expressed as graphs, such as knowledge databases, social networks or supply chains.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

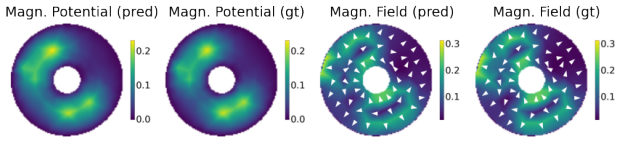


Figure 1. We train a neural network to predict solutions of boundary value problems and additional quantities like in this case the electric potential and electric field of an electrostatics simulation. Here we compare predictions of the model (pred) with ground truth data from an FEM simulation (gt). For the electric field, we visualize the magnitude of the field and overlay it with arrows depicting the field orientation.

In this paper, we focus on a particular application domain that is most naturally defined by graphs: the explicit solution of time-independent (static) boundary value problems. Solutions to such problems are only implicitly defined via partial differential equations (PDEs) subject to boundary value constraints. Solving boundary value problems is required in many physics disciplines, such as thermodynamics or electromagnetics. Finite element methods (FEM), which discretize the PDE on a lattice, are commonly used as generic solvers. The resulting matrix equation is solved using exact or iterative methods of linear algebra. While these methods produce high-quality results, they typically are numerically expensive and any variations on the parameters of the problem require a full rerun to obtain the solutions.

To address this problem, we propose to use GNNs to learn the solution operator of a class of PDEs similar to previous works (Li et al., 2020; 2021). The triangulation of the domain as a first step of an FEM solver is faster than numerically computing the solution of the PDE, which involves an expensive matrix inversion operation. We use only the triangulation grid of an FEM solution as the input to a graph network and employ supervised graph convolution operations to learn the value of the solution function at the triangulation points. We complement previous works that use GNNs to solve time-dependent problems (Pfaff et al., 2021; Sanchez-Gonzalez et al., 2020; Brandstetter et al., 2022). We compare an example prediction of our model to the corresponding FEM solution in figure 1.

To show that the network approximates the solution operator, we investigate if it can learn *superpositions* of different solutions and generalize to different meshes (*shape generalization*) beyond the square mesh that is used frequently in the literature so far (Li et al., 2020; 2021).

GNNs are naturally suited to handle non-square geometries as opposed to CNNs, which are commonly defined on square lattices. CNNs can be seen as a special case of GNNs on regular square domains and a fixed number of neighbors for each node. Hence, CNNs need to work on interpolated solutions if the structure of the data deviates or non-square shapes are used. Moreover, commonly used CNNs need to use masks if parts of the mesh are defined as cut-out regions. We show that our proposed GNN naturally solves all these cases within a unified approach. To our knowledge, this is the first in-depth analysis of GNNs applied to static PDEs that goes beyond learning the solution for a single mesh. Specifically, our contributions are:

- We show that a GNN can efficiently learn the solution to a PDE from simulation data without explicit access to the PDE.
- We demonstrate that diversity with regards to the shape of meshes is a key ingredient for obtaining robust solution operators.
- We demonstrate that our GNN is able to generalize to previously unseen geometries and learn superpositions. As such, we obtain an approximation of the solution operator, instead of a solution itself.
- We construct a new dataset consisting of tesselations and solutions of PDEs in the domains of electrostatics and magnetostatics under non-trivial boundary conditions.

Especially the shape generalization task which we design is considerably more complex than similar tasks in the recent literature, e.g. (Li et al., 2020) which uses only square meshes. We not only verify that the network can adapt to multiple differently-shaped meshes at the same time, but also that it generalizes to a different mesh, which has not been part of the training data. We make both the created dataset as well as the code for our experiments publicly available.

2. Related Work

Battaglia et al. (2016) introduce GNNs for the simulation of the time dynamics of physical objects under complex boundary constraints, such as a string draping over a pole under gravity. They learn the dynamics from purely observational data obtained via simulations of the physical system. They show that the learned dynamics generalize to new situations with a significant speedup over the traditional simulation. Sanchez-Gonzalez et al. (2020) extend this approach and learn the complex dynamics governing the motion of fluids and deformable bodies from simulated data. Each particle represents a node in a graph and the network learns the time update operator from the observed data. Pfaff et al. (2021) are expanding on this by simulating the time dynamics of complex processes based on learning from meshed data. They use two types of nodes to, e.g., model a piece of fabric in the wind held by a flag-pole. Moreover, they show that the learned dynamics generalize from 2k to 20k nodes in the graph despite the latter being out-of-distribution. Mayr et al. (2021) use a GNN to learn the dynamics of a granular material flowing inside a non-stationary boundary, e.g. a rotating drum, by introducing a method to update the nodes describing the geometric boundaries in the GNN.

Fang et al. (2020) use physics-inspired neural networks (PINNs) to parameterize the solution to time-independent PDEs. Making use of the auto-differentiation framework, they minimize the error in the PDE-solution. They develop solutions to common PDEs on complicated two- and three-dimensional manifolds. A similar approach is developed by Cai et al. (2022), who simulate fluid dynamics by combining simulation data and physics-based priors. Afshar et al. (2019) use a CNN architecture to predict the static flow field of a hydrodynamics problem from a triangulated version of the underlying geometry. Yao et al. (2020) leverage a PINN based on the combination of physics-based Finite Element Analysis (FEA) and CNNs to create an efficient predictor of mechanical response quantities. In contrast to this, Li et al. (2020; 2021), use a data-driven approach to learn the solution of PDEs by training a model to map an initial condition to the final output. By doing so, this *Neural Operator Network* learns the solution operator to the PDE without knowing its exact form. Salvi et al. (2021) expand the Neural Operator Network by applying it to stochastic PDEs and incorporating noise into the driving force of the dynamics. Brandstetter et al. (2022) push the Neural Operator Method further by recognizing that numerical methods of classical solvers can be expressed as a special form of autoregressive message-passing protocols in GNNs. They apply this to various time-dependent PDEs for modeling fluid-like flow.

Belbute-Peres et al. (2020) integrate an industry-grade PDE solver as a layer into a GNN simulator. They use adjoint methods to compute the gradient of the solver efficiently. They apply this approach to the simulation of complex fluid dynamics problems. Chamberlain et al. (2021) reverse the approach and leverage discretization schemes developed for numerical simulations of PDEs to design new graph aggregation operations. They show that this approach leads to superior results on some standard GNN benchmarks.

Aarts et al. (2004) use a precursor to PINNs based on fully connected layers to study the solution space of a neural network applied to simple initial as well as boundary value problems. Palade et al. (2020) use neural networks with radial basis functions and an integral objective to solve elliptic PDEs with non-local boundary conditions. However, both approaches do not leverage data to solve the problem, but rather parameterize the solution to a specific instance.

Alet et al. (2019) use a Graph Element Network (GEN) to learn a mapping from initial conditions to the final output of an FEM-based numerical simulation. They show that the GEN can be used to optimize the FEM mesh to focus on the non-linear part of the solution space. A similar approach is also followed by Chen et al. (2021) who leverage a Graph Convolutional Neural Network (GCNN) to learn the solution of steady-state laminar fluid flows around two-dimensional objects based on solutions of the Navier-Stokes equation.

In contrast to these works, we do not optimize the grid space but rather study the generalizability across meshes with different shapes and the ability of the solver to learn superpositions. We thereby show that the network approximates a solution operator rather than just a map of initial conditions to final solutions.

3. Background

In this paper we consider a small but nevertheless important class of PDEs subject to boundary conditions. Let $u \in C^2(\mathbb{R})$ be a twice-differentiable function of n variables over an open domain $\Omega \subset \mathbb{R}^n$. Then, the *Poisson Equation* with *Dirichlet* boundary conditions is given by

$$\begin{aligned} \nabla^2 u(x) &= f(x), & \forall x \in \Omega, \\ u(x) &= g(x), & \forall x \in \partial\Omega, \end{aligned} \tag{1}$$

where f and g are sufficiently smooth functions defined on Ω and $\partial\Omega$, respectively, where $\partial\Omega$ denotes the boundary of the domain Ω. The Poisson equation is often encountered in electrostatics or stationary wave (resonance) problems. Equation (1) is characterized by the absence of a time-component and the presence of a constraint given by g.

To solve equation (1) using an FEM solver, we cast it into its weak form using a test function v. We homogenize the boundary conditions with the transformation $y(x) = u(x) - g(x)$ and define $h(x) = f(x) - \nabla^2 g(x)$. In the last step, we used the continuation of g into the domain Ω. The weak form is then obtained using Green's first identity (also know as Stokes' theorem) and results in

$$a(y, v) = \hat{h}(v), \tag{2}$$

where

$$\begin{aligned} a(y, v) &= \int_\Omega \mathrm{d}x \sum_{i=1}^{n} \frac{\partial y}{\partial x_i} \frac{\partial v}{\partial x_i}, \\ \hat{h}(v) &= \int_\Omega \mathrm{d}x h(x) v(x). \end{aligned} \tag{3}$$

The key insight of FEM is to choose the test function v such that it is well behaved and can be represented as a sum over compact, local basis functions on Ω, i.e. $v(x) = \sum_t \phi_t(x)$, where the basis elements ϕ_t are piece-wise linear functions or simplices, known as *finite elements*. The solution $y(x)$ is then expressed as $y(x) = \sum_t y_t \phi_t(x)$ and y_t is representing the value of the solution at a defined node of the simplex, i.e. a finite element. Using this basis expansion, equation (2) reduces to the linear algebraic equation

$$\mathcal{A}\mathbf{y} = \mathbf{h}, \tag{4}$$

where $\mathbf{y} = (y_t)_{t=1\ldots T}$ and $\mathcal{A}$ can be calculated from the weak-form integral.

The choice of the finite element grid is still open, but it can be seen that the relations between the neighborhood of y_t can be concisely expressed as edges between corresponding nodes in a graph structure G. Using this relationship, we use a GNN and graph convolutions to learn the solution operator of (4).

Poisson Equation for electrostatic and magnetostatic problems: The *Poisson Equation* can be used to explain many physical phenomena. In the following, we focus on experiments with electrostatics and magnetostatics simulations on two-dimensional meshes. The derivations we present are based on Langtangen et al. (2017). The electric potential U can be described with the following variant of the *Poisson Equation*:

$$-\nabla^2 U = \frac{\rho}{\epsilon}, \tag{5}$$

where ρ is the charge density and ϵ is the permittivity of the material. With the *Dirichlet* boundary condition, we set U at the boundary to a fixed value of zero. Note that we can still include PDEs with non-zero boundary conditions in this configuration: Our final operator would be applicable to non-zero boundary conditions by including the boundary to the solution domain and placing corresponding charges on those nodes. The solution can then be found by creating a new boundary outside of the solution domain, which can be discarded afterwards. After solving equation (5), the electric field can be computed via the gradient of the potential:

$$-\nabla U = \mathbf{E}. \tag{6}$$

Let x and y be the spatial coordinates of the mesh. To simulate magnetostatic effects on a two-dimensional mesh, we

	PDE	Quantities
Input	es	Distance to border (dx, dy) Charge (ρ) Boundary condition
Output	es	Electric potential (U) Electric field (E_x, E_y)
Input	ms	Distance to border (dx, dy) Electric current (I) Boundary condition
Output	ms	Magnetic vector potential (A_z) Magnetic field (B_x, B_y)

Table 1. Overview of the node attributes used as input and target for training of the GNNs. es: electrostratics, ms: magnetostatitcs.

consider electric currents along the z axis that are orthogonal to the mesh. The magnetic vector potential $\mathbf{A}$ can be described as

$$-\nabla^2 \mathbf{A} = \mu \mathbf{I}, \tag{7}$$

where $\mathbf{I}$ is the current density and μ is the permeability of the material. We consider the case $I_x = I_y = 0$. Our magnetostatics experiments can thus be viewed as sets of infinite wires that are orthogonal to the two-dimensional mesh. As the cross section of the wires looks the same irrespective of the z coordinate, we find that A_x and A_y cannot depend on z. For symmetry reasons, the magnetic field is zero in z direction and the interesting part of the vector potential can be reduced to a scalar value A_z. With the *Dirichlet* boundary condition, we set A_z at the boundary to a fixed value of zero. With a similar argument as above, our trained solution operator can generalize to non-zero boundary conditions as well. After solving equation (7), the magnetic field can be derived as the curl of $\mathbf{A}$, or in our case, simply:

$$B_x = \frac{\partial A_z}{\partial y}, B_y = -\frac{\partial A_z}{\partial x}. \tag{8}$$

4. GNN Solution Operator Experiments

After discussing the variants of the Poisson equation that we aim to solve in section 3, we now turn to the description of the concrete experiments and the data generation process. The triangulation of the solution domain for the use of FEM solutions can naturally be expressed as a graph. Each node p_i of the triangulation can be described by a binary feature $b_i \in \{0, 1\}$, denoting whether the point is a boundary point or interior point, and additional features q_i describing the inhomogeneities of the PDE, i.e. $p_i = (b_i, q_i)$. We also use relative distances $\mathcal{D}_{ij}$ between connected nodes as edge attributes in the graph. See table 1 for an overview of the node attributes. In figure 2 we show two examples of generated inhomogeneities and PDE solutions within our data.

	Mesh augmentation	Number of charges / currents
set1	✗	1–3
set2	✓	1–3
set3	✗	4–5

Table 2. For each mesh, we generate sets of 2500 samples each: we generate data with and without mesh augmentation and also vary the number of positive charges or currents. For each set in the table we generate a total of 12 500 samples, as we use 5 meshes. The data generation is applied equally for the electrostatics and magnetostatics simulations.

The goal of our approach is to learn a neural network, such that

$$\mathbf{y} = \mathbb{NN}_\theta[\mathbf{p}, \mathcal{D}], \text{s.t. } \mathcal{A}\mathbf{y} = \mathbf{h}, \tag{9}$$

where $\mathbb{NN}_\theta$ is a neural network parametrized by θ that operates on the node features $\mathbf{p}$ and edge features $\mathcal{D}$ defined above. We train this network such that, given a triangulation mesh $\mathbf{x}$ and additional features, it predicts the solution values $\mathbf{y}$ of the underlying PDE. Note that an individual value y_i can be a scalar or a vector depending on the underlying system described by the PDE. This corresponds to learning the solution operator in equation (4). Our final network is able to predict solutions for a variety of geometries with only a few graph convolution operations and faster than an FEM solver.

We parameterize the model using a GNN where each node in the triangulation maps to a node in the GNN. In this way, our approach is extensible to modern FEM triangulation meshes that are typically not equidistant but dynamically refined as e.g. in (Pfaff et al., 2021). We train the network in a supervised fashion to predict the solution to a specific instance of the corresponding PDE that was obtained from an exact simulation via an FEM solver. Adapting the mesh resolution to underlying properties of the physical simulation would be a straightforward extension of our method, which we defer to future work.

4.1. Dataset Generation

We generate datasets for several physical quantities over a variety of geometries using different simple simulations with the FEniCS library (Scroggs et al., 2021), a free FEM solver.

Mesh geometries and mesh augmentations: We use five different mesh geometries on which we solve the PDEs numerically. Our geometries are comparable to recent work (Hsieh et al., 2019): A square mesh, a disk with and without a hole, an L-shaped mesh and a U-shaped mesh. See

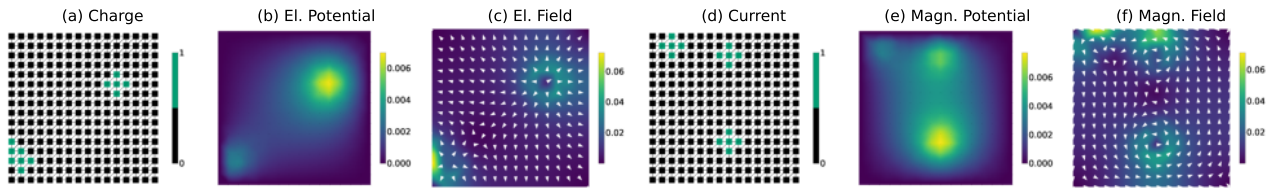
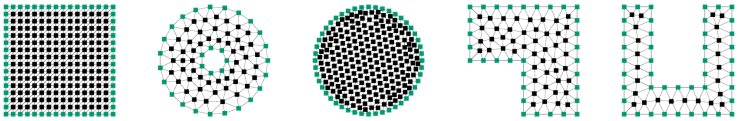
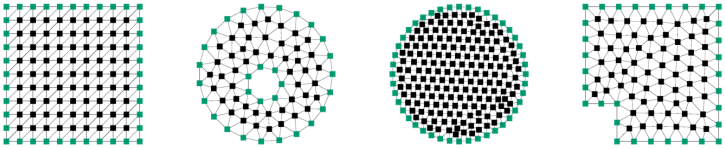


Figure 2. Visualized input and output quantities for the square mesh. For the scalar fields (potentials) we plot the magnitude. For the vector fields, we visualize the magnitude of the field and overlay it with arrows depicting the field orientation. (a) The charge input for the electrostatics problem. (b–c) Prediction targets in form of the potential and the electric field. (d) Current input for the magnetostatics problem. (e–f) Prediction targets in form of the magnetic vector potential (only z component) and the magnetic vector field.



(a) Visualization of the five mesh types without any mesh augmentation.



(b) Examples of meshes with mesh augmentation. The U-mesh is not augmented.

Figure 3. Examples of the different mesh geometries used for solving the PDEs and constructing the dataset. The augmentations are applied to (i) the disk mesh and square mesh, where we vary the node density (ii) the disk with a hole, where we vary the hole size and location (iii) the L-mesh, where we control the size of the cutout. The nodes where *Dirichlet* boundary conditions are active are marked in green. Note that the different meshes also vary in resolution.

figure 3(a) for a visualization of the meshes. We use normalized coordinates in the range [0,1] to define our meshes and thus do not report units for the coordinates. Our default square mesh has 256 evenly distributed nodes, the disk mesh has 252 nodes. To construct a disk with a hole, we use a disk with radius 0.5 and center (0.5, 0.5) and a circular cutout with center (0.5, 0.5) and radius 0.12. This yields 82 nodes for the disk with hole. For the L-mesh, we cut out one quarter of a square which yields 80 nodes. We construct a regular U-mesh with 68 nodes as a cutout from a unit square. The absolute positions of the nodes in the meshes are not relevant, as we do not use those during training. We use Delaunay triangulation to obtain the tesselations for all shapes.

In addition to the basic meshes, we induce variability to the data using *mesh augmentations*. These augmentations act as a regularizer to the network similar to standard data augmentation techniques in other domains. We hypothesize that these augmentations are useful for training a network that is able to generalize to a variety of meshes with different geometries. The transforms consist of random variations of the mesh density of the disk and square shape, the location and radius of the hole in the disk as well as the size of the cutout in the L-mesh. A set of exemplary augmentations are depicted in figure 3(b). We do not apply augmentations to the U-mesh, as it is only used for testing generalization.

Specifically, for the square shape, we vary the number of nodes and thereby the resolution of the mesh ranging from 64 to 441 nodes. Similarly for the disk shape, we vary the number of nodes ranging from 63 to 411. For the disk with a hole, we vary the x and y coordinates of the cutout, both in the range of 0.35 to 0.65. We also vary the radius of the cutout from 0.05 to 0.25. For the L-mesh, we vary the width and height of the cutout rectangle both in the range of 0.2 to 0.8.

During training, we compare the proposed mesh augmentations with other standard approaches like node or edge dropout or dropout of embeddings (see section 4.3 for a more in-depth description).

Physical problems: Having described the mesh geometries underlying the dataset, we now turn to the physical PDEs we aim to solve on those meshes. We focused on simulating the two-dimensional electrostatics and magnetostatics problems described in section 3. The magnetic vector potential as well as the electric potential are described by Poisson equations with current or charge inhomogeneities respectively. We also experiment with training the networks to predict other quantities like the magnetic or electric vector fields.

In the case of electrostatics, we generated up to 3 circular charges for each instance. The charges are distributed randomly over the mesh and all have the same positive charge density. We consider the permittivity ϵ to be constant. Generalizing across different values for the permittivity and charge density should be possible with a minor modification of our dataset, but we chose to omit this for simplicity. We then solved the two-dimensional electrostatics problem numerically and saved the mesh with its metadata as well as the resulting field and potential solution as a single instance of the resulting dataset. For the magnetostatics experiments, we followed the same approach like above, but replaced the circular charges with an electric current and assume a constant permeability μ. Again, generalizing across different values of the permeability has been omitted for simplicity.

For each of the meshes and PDEs, we created three groups of 2500 simulations: The first group was generated without any mesh augmentation. The second group uses mesh augmentation as described above. The third group includes more difficult simulations that are composed of simpler ones, such as examples with a higher number of charges or currents. The different settings are summarized in table 2. We generated 7500 simulations for each mesh, i.e. 37 500 simulations in total.

4.2. Experiments

We create two different tasks to test the capabilities of our model with respect to its approximation quality and generalizibility. More specifically, we test (i) superposition, i.e. the networks' ability to generalize to a larger number of inhomogeneities by summing up the results seen during training. (ii) Shape generalization, i.e. the model's ability to generalize to previously unseen geometries. In both cases, we test the performance with and without mesh augmentation to demonstrate the importance of data-driven regularization. To evaluate these tasks, we need to split the data into training, validation and test sets accordingly. The splitting strategies apply equally to the magnetostatics and electrostatics experiments. We make the full dataset and experiment splits publicly available.

We design a single experiment to test superposition: Considering all meshes, we test if the model can generalize from simple problems (e.g. electrostatics simulations with up to three charges) to more complex compositions (e.g. electrostatics simulations with up to five charges). For testing generalization across meshes of different shapes, we train on a subset of all meshes and use a single unseen mesh for testing. To make the shape generalization task more difficult, we chose to exclude the U-mesh, which exhibits less symmetries than others.

In total, we split the generated data (see table 2) in two test sets and two training and validation sets:

Training / validation set without mesh augmentation: We select all generated data without mesh augmentation and 1–3 charges (set1 in table 2) from the square mesh, L-mesh, circle and circle with a hole (all meshes except the U-mesh). We split this data into training and validation using a 80:20 split. There are a total of 2000 samples in the validation set and 8000 samples in the training set without mesh augmentation.

Training / validation set with mesh augmentation: We select all generated data with mesh augmentation and 1–3 charges (set2 in table 2) from the square mesh, L-mesh, circle and circle with a hole (all meshes except the U-mesh). We split this data into training and validation using a 80:20 split. There are a total of 2000 samples in the validation set and 8000 samples in the training set with mesh augmentation.

Shape generalization test set: We select all generated data without mesh augmentation (set1 in table 2) from the U-mesh. We have 2500 samples in this test set.

Superposition test set: We select all generated data without mesh augmentation and 4–5 charges (set3 in table 2) from the square mesh, L-mesh, circle and circle with hole (all meshes except the U-mesh). To be of the same size like the shape generalization test set, we randomly sample 25% of this data, such that there are 2500 samples in the superposition set.

The two tasks can be summarized as follows: For the *superposition task* we train and validate two times, each time using one of the train and validation sets. We test each experiment on the superposition test set. For the *shape generalization task* we train and validate two times, each time using one of the train and validation sets. We test each experiment on the shape generalization test set.

4.3. GNN Architecture and Training Details.

Our model follows the encoder-processor-decoder architecture proposed in recent work on learning physical simulations with GNNs (Pfaff et al., 2021). The encoder and decoder are two-layer MLPs with 128 hidden units each and ReLU nonlinearities except for the output layer of the decoder, after which we do not apply any nonlinearity. We use

the spectral graph convolution operator introduced by Defferand et al. (Defferrard et al., 2016) with 128-dimensional hidden features. We set the number of hops for the graph convolutions in the processor to $K = 5$. We use 3 consecutive graph convolutions with a ReLU nonlinearity after each step. The model has a total of 280 835 parameters. We implement the GNN based on PyTorch (Paszke et al., 2019) using the PyTorch Lightning framework (Falcon & the PyTorch Lightning team, 2019) as well as PyTorch Geometric (Fey & Lenssen, 2019).

We use the Adam optimizer (Kingma & Ba, 2014) with learning rate 1E-3 and betas 0.9 and 0.999. We use the mean squared error over all predicted quantities for training and reporting results. We use a batch size of 32 and train on an NVIDIA T4 with 16GB VRAM in a Google Kubernetes cluster. The node attributes, i.e. the input of the encoder, contain the charge (current) of the node, a flag indicating whether or not it is part of the boundary and the node's distance to the closest boundary node. We use relative distances to neighboring nodes as edge attributes.

As a normalization strategy, all features are divided by their respective maximum absolute values, which are obtained from both accumulated train splits (see section 4.2). The training data for all quantities thus lies in the range [-1,1]. We report results relative to the normalization constants without units. The network is tasked to predict the electric (magnetic) potential as well as the electric (magnetic) vector fields.

We train each experiment for 150 epochs and validate after each epoch. For the final evaluation, we use the model parameters from the epoch with the lowest validation loss. We use the same random seed for all experiments.

We compare our proposed mesh augmentation with other regularization and augmentation techniques. We use dropout of node features before feeding them to the encoder with a probability of 0.2 (feature dropout), dropout of nodes in the graph with a probability of 0.1 and dropout of edges in the graph with a probability of 0.2.

4.4. Results

We first show that the capability to solve PDEs on meshes with varying shapes is greatly enhanced by mesh augmentation during training. Table 3 summarizes the results for the shape generalization task for both the electrostatics and magnetostatics problems. Note that, due to the normalization of all quantities, we report results without units as described in section 4.3. For this task, the model is presented with an unseen mesh (the U-mesh) during testing. Comparing the results for the electric (magnetic) potential, the version of the model with mesh augmentation outperforms the baseline without mesh augmentation by at least two orders of magnitude. For the derived quantities, i.e. the electric (magnetic) field, there is a smaller but still substantial improvement. Overall, mesh augmentation seems to be a crucial component for generalizing to unseen shapes.

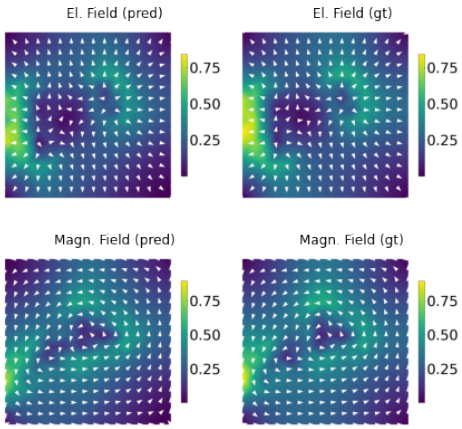
PDE	Task	Mesh aug.	MSE_{pot}	$\text{MSE}_{\text{field}}$
es	shape	✗	0.327E−3	2.723E−3
es	shape	✓	**0.006E−3**	**1.821E−3**
ms	shape	✗	0.161E−3	2.640E−3
ms	shape	✓	**0.006E−3**	**1.338E−3**
es	sup.	✗	0.272E−3	0.602E−3
es	sup.	✓	**0.102E−3**	**0.267E−3**
ms	sup.	✗	0.256E−3	0.434E−3
ms	sup.	✓	**0.172E−3**	**0.276E−3**

Table 3. Test results for electrostatics (es) and magnetostatics (ms) PDEs. Mesh augmentation (mesh aug.) denotes if the training/validation set including or excluding mesh augmentation is used. The best results are highlighted. The data for all experiments is normalized and thus results are multiples of normalization constants and reported without physical units. The top half shows the rest results in the shape generalization (shape) task: The shape generalization test set is used for these experiments. The bottom half shows the test results for the superposition (sup.) task: The superposition test set is used in these experiments.

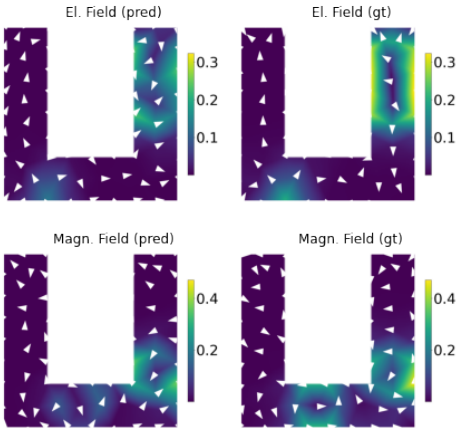
PDE	Regularization	MSE_{pot}	$\text{MSE}_{\text{field}}$
es	feature drop	1.094E−3	4.104E−3
es	none	0.327E−3	2.723E−3
es	node drop	0.077E−3	3.077E−3
es	edge drop	0.041E−3	2.826E−3
es	mesh augmentation	**0.006E−3**	**1.821E−3**

Table 4. Study comparing mesh augmentation with various other regularization techniques for the electrostatics (es) PDE. The shape generalization test set is used in all experiments. The best results for each quantity and problem are highlighted.

Secondly, we show that mesh augmentation improves the performance for the superposition task as well. Table 3 summarizes the results for the superposition task for both the electrostatics and magnetostatics problems. For both the prediction of the potentials as well as the derived quantities, the electric and magnetic fields, we notice substantial improvements when using mesh augmentation. We therefore argue that mesh augmentation is a general technique to improve generalization for different kinds of tasks. More generally, diversity in the training data seems to improve the performance of the model with respect to a variety of tasks, which suggests that training on a diverse dataset with different meshes is a key ingredient for learning generalizable solution operators for static PDEs.



(a) Largest errors for Superposition.



(b) Largest errors for shape generalization.

Figure 4. Visualizations of the largest testing errors for our best performing models. We compare predictions of the model (pred) with ground truth data from the FEM simulation (gt). The background color depicts the magnitude of the respective field and the orientation is denoted by the arrows. The mean squared errors (vector field) for the shape experiments are $7.327\text{E}{-3}$ (electrostatics) and $10.05\text{E}{-3}$ (magnetostatics). The mean squared errors (vector field) for the superposition experiments are $1.013\text{E}{-3}$ (electrostatics) and $1.154\text{E}{-3}$ (magnetostatics).

To investigate this further, we study other techniques to augment and regularize training and compare those with the proposed mesh augmentation. We exemplarily select the electrostatics problem to carry out this study. In general, those techniques that augment and vary the graph structure lead to improvements. Table 4 shows the results for this experiment comparing mesh augmentation with random dropout of entire nodes or edges as well as dropout of node features and no augmentation at all. The dropout of features, which does not change or augment the graph structure, performs worse than no augmentation at all. Both dropout of edges and nodes change the mesh structure and thus contribute to better generalization. Mesh augmentation clearly outperforms all other augmentation techniques. This supports our argument that changes in the mesh structure during training are essential for learning generally applicable solution operators.

We further investigate the root cause of the comparatively high prediction error for the electric and magnetic fields. This behavior can be observed especially for the shape generalization task in table 3. In figure 4, we analyze the largest errors of our best-performing model for both the shape generalization and the superposition tasks. For the shape generalization task, we observe, that there is a mismatch in the prediction of the directions for both the magnetic and electric field. We also find that the model's predictions violate physical constraints like $\nabla \times \mathbf{E} = 0$ for the electric field.

5. Summary and Conclusion

To the best of our knowledge, we are the first to investigate the role of diversity in training data for learning accurate solutions operators in the case of static PDE problems. We show, that graph networks as approximators for static FEM simulations can generalize to unseen meshes with different shapes as well as an increased number of inhomogeneities with respect to charges or currents.

One of our key findings is that augmentation techniques on meshes are essential to enhance the ability of the neural operator to generalize. This suggests that graph neural networks can be used as universal solution operators for classes of PDEs, but only if they are trained carefully. Especially, diversity in the training data seems to be a key ingredient to enable robust generalization.

In all our tasks, the electric or magnetic potential can be approximated with high accuracy. We therefore show that our system can solve the Poisson equation for a variety of different constraints. For the prediction of derived quantities like the electric or magnetic field, we observe comparatively large errors. As mentioned in section 4.4, we observe that for these quantities the model violates physical constraints.

We conclude that further research is needed to enable the learning of explicit or implicit physical constraints. As a main limitations of our work, it remains to be shown that more general classes of PDEs besides Poisson equations can be solved using our method. We aim at extending this study for a range of different problems and believe that our dataset should be used to verify the robustness for other neural operators in the future. The meshes that we use are generated using Delaunay triangulation with similar distances between all connected nodes. It remains to be shown that our approach generalizes to meshes with dynamically refined node densities.

References

Aarts, L. P. and der Veer, P. V. Neural network method for solving partial differential equations. *Neural Processing Letters*, 14:261–271, 2004.

Afshar, Y., Bhatnagar, S., Pan, S., Duraisamy, K., and Kaushik, S. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64:525–545, 2019.

Alet, F., Jeewajee, A. K., Bauzá, M., Rodriguez, A., Lozano-Perez, T., and Kaelbling, L. P. Graph element networks: adaptive, structured computation and memory. *ArXiv*, abs/1904.09019, 2019.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., and Kavukcuoglu, K. Interaction networks for learning about objects, relations and physics. *ArXiv*, abs/1612.00222, 2016.

Brandstetter, J., Worrall, D. E., and Welling, M. Message passing neural pde solvers. *ArXiv*, abs/2202.03376, 2022.

Cai, S., Mao, Z., Wang, Z., Yin, M., and Karniadakis, G. E. Physics-informed neural networks (pinns) for fluid mechanics: A review. *ArXiv*, abs/2105.09506, 2022.

Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., and Veličković, P. Combinatorial optimization and reasoning with graph neural networks. *arXiv preprint arXiv:2102.09544*, 2021.

Chamberlain, B. P., Rowbottom, J. R., Gorinova, M. I., Webb, S., Rossi, E., and Bronstein, M. M. Grand: Graph neural diffusion. In *ICML*, 2021.

Chen, J., Hachem, E., and Viquerat, J. Graph neural networks for laminar flow prediction around random two-dimensional shapes. *Physics of Fluids*, 2021.

de Avila Belbute-Peres, F., Economon, T. D., and Kolter, J. Z. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *ICML*, 2020.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

Falcon, W. and the PyTorch Lightning team. Pytorch lightning, 2019. URL https://www.pytorchlightning.ai.

Fang, Z. and Zhan, J. Z. A physics-informed neural network framework for pdes on 3d surfaces: Time independent problems. *IEEE Access*, 8:26328–26335, 2020.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Henderson, R., Clevert, D.-A., and Montanari, F. Improving molecular graph neural network explainability with orthonormalization and induced sparsity, 2021.

Hsieh, J.-T., Zhao, S., Eismann, S., Mirabella, L., and Ermon, S. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019.

Jumper, J. M., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Zídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., Back, T., Petersen, S., Reiman, D. A., Clancy, E., Zielinski, M., Steinegger, M., Pacholska, M., Berghammer, T., Bodenstein, S., Silver, D., Vinyals, O., Senior, A. W., Kavukcuoglu, K., Kohli, P., and Hassabis, D. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Langtangen, H. P. and Logg, A. *Solving PDEs in python: the FEniCS tutorial I*. Springer Nature, 2017.

Li, Z.-Y., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *ArXiv*, abs/2003.03485, 2020.

Li, Z.-Y., Kovachki, N. B., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *ArXiv*, abs/2010.08895, 2021.

Mayr, A., Lehner, S., Mayrhofer, A., Kloss, C., Hochreiter, S., and Brandstetter, J. Boundary graph neural networks for 3d simulations. *ArXiv*, abs/2106.11299, 2021.

Maziarka, L., Danel, T., Mucha, S., Rataj, K., Tabor, J., and Jastrzebski, S. Molecule attention transformer, 2020.

Palade, V., Petrov, M. S., and Todorov, T. D. Neural network approach for solving nonlocal boundary value problems. *Neural Computing and Applications*, pp. 1–19, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc,

F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. *ArXiv*, abs/2010.03409, 2021.

Salvi, C. and Lemercier, M. Neural stochastic partial differential equations. *ArXiv*, abs/2110.10249, 2021.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. Learning to simulate complex physics with graph networks. *ArXiv*, abs/2002.09405, 2020.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20:61–80, 2009.

Scroggs, M. W., Dokken, J. S., Richardson, C. N., and Wells, G. N. Construction of arbitrary order finite element degree-of-freedom maps on polygonal and polyhedral cell meshes, 2021. URL fenicsproject.org.

Yao, H., Gao, Y., and Liu, Y. Fea-net: A physics-guided data-driven model for efficient mechanical response prediction. *ArXiv*, abs/2002.01893, 2020.

6. Appendix

6.1. Elasticity data

In an effort to create a more challenging dataset to foster future research, we create data for another PDE corresponding to gravity-induced deflection within a linear elasticity problem. The following derivations are based on Langtangen et al. (2017). The initial equations, describing small elastic deformations, are

$$-\nabla \cdot \sigma = f \tag{10}$$

$$\sigma = \lambda \mathrm{Tr}\,(\epsilon)\,\mathrm{I} + 2\mu\epsilon \tag{11}$$

$$\epsilon = \frac{1}{2}\left[\nabla u + (\nabla u)^{\mathrm{T}}\right]. \tag{12}$$

Here, f is the force acting on the body, λ and μ are the Lamé coefficients, I is the identity matrix, ϵ is the strain tensor, σ is the stress tensor and u is the displacement vector field. By combining the last two equations we obtain

$$\sigma\,(u) = \lambda(\nabla \cdot u)I + \mu(\nabla u + (\nabla u)^{\mathrm{T}}). \tag{13}$$

When solving these equations numerically, it is useful to work in a variational formulation. For this reason, we introduce an arbitrary test vector v and integrate over the whole body to obtain the condition that must hold for any v:

$$a\,(u, v) = L\,(v) \tag{14}$$

where

$$a\,(u, v) = \int_{\Omega} \sigma\,(u) : \epsilon\,(v)\; dx, \tag{15}$$

$$L\,(v) = \int_{\Omega} f \cdot v\, dx + \int_{\partial\Omega_T} T \cdot v\, ds, \tag{16}$$

$$\epsilon\,(v) = \frac{1}{2}\left[\nabla v + (\nabla v)^{T}\right] \tag{17}$$

The colon operator represents the inner product between tensors (summed pairwise product of all elements) and $\partial\Omega_T$ is the part of the boundary where we prescribe the boundary condition $\sigma \cdot n = T$, with n being the outward pointing surface normal vector.

The data generation process for the elasticity problem is exactly the same as described in section 4.1. See figure 5 and table 5 for an overview of the input and output quantities we use for the linear elasticity data. For set1 and set2 in table 2, we use 1–3 fixed vertical lines (via boundary conditions). For set3, we use an increased number of 4–5 fixed vertical lines to test superposition. For future reference, we will also publish training/validation/test splits like described in section 4.2 for this part of the dataset, although we did not conduct any experiments yet.

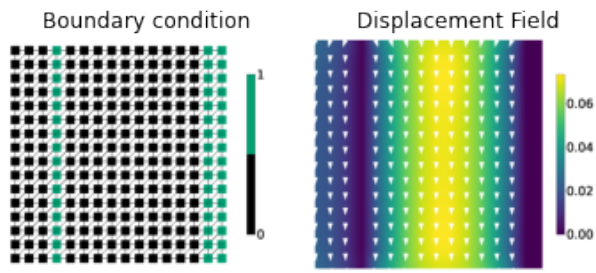


Figure 5. Elasticity input boundary conditions and displacement field.

	Linear Elasticity
Input	Distance to border (dx, dy) Distance to boundary condition (dx, dy) Boundary condition
Output	Displacement field (u_x, u_y)

Table 5. Overview of the node attributes used as input and target for the linear elasticity problem.