Physics-Embedded Neural Networks: $\mathrm{E}(n)$-Equivariant Graph Neural PDE Solvers

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph neural network (GNN) is a promising approach to learning and predicting physical phenomena described in boundary value problems, such as partial differential equations (PDEs) with boundary conditions. However, existing models inadequately treat boundary conditions essential for the reliable prediction of such problems. In addition, because of the locally connected nature of GNNs, it is difficult to accurately predict the state after a long time, where interaction between vertices tends to be global. We present our approach termed physics-embedded neural networks that considers boundary conditions and predicts the state after a long time using an implicit method. It is built based on an $\mathrm{E}(n)$-equivariant GNN, resulting in high generalization performance on various shapes. We demonstrate that our model learns flow phenomena in complex shapes and outperforms a well-optimized classical solver and a state-of-the-art machine learning model in speed-accuracy trade-off. Therefore, our model can be a useful standard for realizing reliable, fast, and accurate GNN-based PDE solvers.

1 Introduction

Partial differential equations (PDEs) are of interest to many scientists because of their application in various fields such as mathematics, physics, and engineering. Numerical analysis is used to solve PDEs because most PDE problems in real life cannot be solved analytically. For example, predicting fluid behavior in complex shapes is an essential topic because it is helpful for product design, disaster reduction, weather forecasting, and many others; however, it is a difficult problem and takes time to solve using classical solvers. Machine learning is a promising approach to predicting such phenomena because it can utilize data similar to the state to be predicted, while classical solvers cannot utilize such information.

However, the main challenge in dealing with complex phenomena such as fluids is to guarantee generalization performance because possible states in complex systems can be huge and may not be covered using a purely data-driven approach. Therefore, we must apply appropriate inductive biases to machine learning models. Many approaches successfully introduced various inductive biases such as local connectedness using graph neural networks (GNNs) and symmetry under coordinate transformations using equivariance.

While these methods have made great progress in solving PDEs using machine learning, there is still room for improvement. First, there is need for an efficient and provable way to respect boundary conditions like Dirichlet and Neumann. Rigorous fulfillment of Dirichlet boundary conditions is indispensable because they are hard constraints and different Dirichlet conditions correspond to different problems users would like to solve. Second, there is need to reinforce the treatment of global interaction to predict the state after a long time, where interactions tend to be global. GNNs have

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

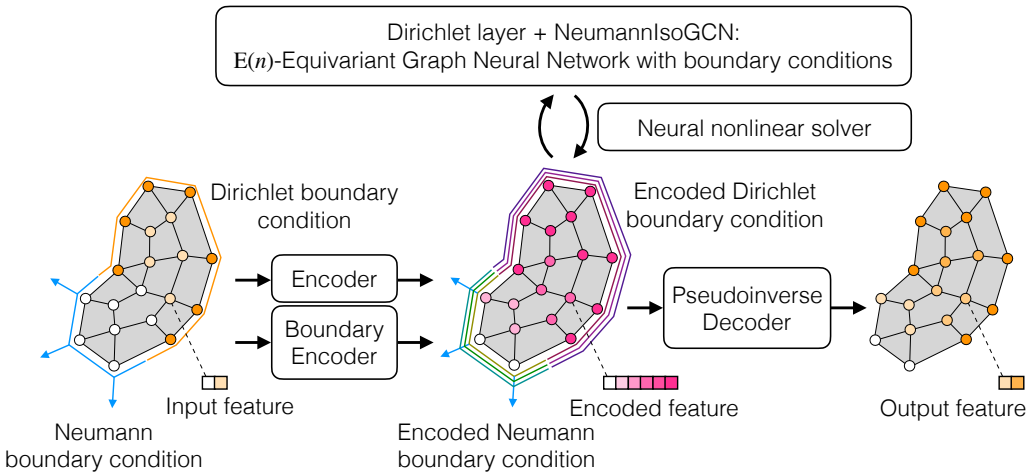

Figure 1: Overview of the proposed method. On decoding input features, we apply boundary encoders to boundary conditions. Thereafter, we apply a nonlinear solver consisting of an $\mathrm{E}(n)$-equivariant graph neural network in the encoded space. Here, we apply encoded boundary conditions for each iteration of the nonlinear solver. After the solver stops, we apply the pseudoinverse decoder to satisfy Dirichlet boundary conditions.

excellent generalization properties because of their locally-connected nature; however, they may miss global interaction due to their localness.

We propose physics-embedded neural networks (PENNs), a machine learning framework to address these issues by embedding physics in the models. We build our model based on IsoGCN (Horie et al., 2021), a lightweight $\mathrm{E}(n)$-equivariant GNN to reflect physical symmetry and realize fast prediction. Furthermore, we construct a method to consider Dirichlet and Neumann boundary conditions. Finally, we reconsider a way to stack GNNs based on an iterative method used for nonlinear optimization problems, which naturally introduces the global pooling to GNNs as the global interaction with high interpretability. In experiments, we demonstrate that our treatment of Neumann boundary conditions improves the predictive performance of the model, and our method can fulfill Dirichlet boundary conditions with no error. Our method also achieves state-of-the-art performance compared to a classical, well-optimized numerical solver and a baseline machine learning model in speed-accuracy trade-off.

Our main contributions are summarized as follows:

- We construct four models to satisfy Dirichlet and Neumann boundary conditions: the *boundary encoder*, *Dirichlet layer*, *pseudoinverse decoder*, and *NeumannIsoGCN*. The considered models show provable fulfillment of boundary conditions, while existing models cannot.
- We propose *neural nonlinear solvers*, which realize global connections to stably predict the state after a long time.
- We demonstrate that the proposed model shows state-of-the-art performance in speed-accuracy trade-off.

Figure 1 shows the overview of the proposed model.

2 Background and related work

In this section, we review the foundation of PDEs to clarify the problems we solve and introduce related works where machine learning models are used to solve PDEs.

2.1 Partial differential equations (PDEs) with boundary conditions

A general form of the d-dimensional temporal PDEs that we consider can be expressed as follows:

$$\frac{\partial \boldsymbol{u}}{\partial t}(t, \boldsymbol{x}) = \mathcal{D}(\boldsymbol{u})(t, \boldsymbol{x}) \qquad (t, \boldsymbol{x}) \in (0, T) \times \Omega, \tag{1}$$

$$\boldsymbol{u}(t = 0, \boldsymbol{x}) = \hat{\boldsymbol{u}}_0(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega, \tag{2}$$

$$\boldsymbol{u}(t, \boldsymbol{x}) = \hat{\boldsymbol{u}}(t, \boldsymbol{x}) \qquad (t, \boldsymbol{x}) \in (0, T) \times \partial\Omega_{\text{Dirichlet}}, \tag{3}$$

$$\hat{\boldsymbol{f}}(\nabla \boldsymbol{u}(t, \boldsymbol{x}), \boldsymbol{n}(\boldsymbol{x})) = \boldsymbol{0} \qquad (t, \boldsymbol{x}) \in (0, T) \times \partial\Omega_{\text{Neumann}}, \tag{4}$$

where Ω is the domain, $\partial\Omega$ is the boundary of Ω, and $\partial\Omega_{\text{Dirichlet}}$ and $\partial\Omega_{\text{Neumann}}$ are boundaries with Dirichlet and Neumann boundary conditions. $\hat{\cdot}$ is a known function, and $\mathcal{D}$ is a known nonlinear differential operator, which can be nonlinear and contain spatial differential operators (see equation 18 for an example of $\mathcal{D}$). $\boldsymbol{n}(\boldsymbol{x})$ is the normal vector at $\boldsymbol{x} \in \partial\Omega$. Equation 3 is called the Dirichlet boundary condition, where the value on $\partial\Omega_{\text{Dirichlet}}$ is set as a constraint. Equation 4 corresponds to the Neumann boundary condition, where the value of the derivative $\boldsymbol{u}$ in the direction of $\boldsymbol{n}$ is set on $\partial\Omega_{\text{Neumann}}$ rather than the value of $\boldsymbol{u}$. When $\boldsymbol{u} : (0, T) \times \Omega \to \mathbb{R}^f$ satisfies Equations 1 – 4, it is called the solution of the (initial-) boundary value problem.

2.1.1 Discretization

PDEs are defined in a continuous space to make differentials meaningful. Discretization can be applied in space and time so that computers can solve PDEs easily. In numerical analysis of complex-shaped domains, we commonly use meshes (discretized representations of shapes), which can be regarded as graphs. We denote the position of the ith vertex as $\boldsymbol{x}_i$ and the value of a function f at the $\boldsymbol{x}_i$ as f_i.[1]

The simplest method to discretize time is the explicit Euler method formulated as:

$$\boldsymbol{u}(t + \Delta t, \boldsymbol{x}_i) = \boldsymbol{u}(t, \boldsymbol{x}_i) + \mathcal{D}(\boldsymbol{u})(t, \boldsymbol{x}_i)\Delta t, \tag{5}$$

which updates $\boldsymbol{u}(t, \boldsymbol{x}_i)$ with a small increment $\mathcal{D}(\boldsymbol{u})(t, \boldsymbol{x}_i)\Delta t$. Another way to have time discretization is the implicit Euler method formulated as:

$$\boldsymbol{u}(t + \Delta t, \boldsymbol{x}_i) = \boldsymbol{u}(t, \boldsymbol{x}_i) + \mathcal{D}(\boldsymbol{u})(t + \Delta t, \boldsymbol{x}_i)\Delta t, \tag{6}$$

which solves Equation 6 rather than simply updating variables to ensure the original PDE is satisfied numerically. The equation can be viewed as a nonlinear optimization problem by formulating it as:

$$\boldsymbol{R}(\boldsymbol{v}) := \boldsymbol{v} - \boldsymbol{u}(t, \cdot) - \mathcal{D}(\boldsymbol{v})\Delta t, \tag{7}$$

$$\text{Solve}_{\boldsymbol{v}}\ \boldsymbol{R}(\boldsymbol{v})(\boldsymbol{x}_i) = \boldsymbol{0},\ \forall i, \tag{8}$$

where $\boldsymbol{R}(\boldsymbol{v})$ is the residual vector of the discretized PDE. The solution of Equation 8 corresponds to the state after Δt, $\boldsymbol{u}(t + \Delta t, \boldsymbol{x})$. By letting $\nabla\phi = \boldsymbol{R}$ for an appropriate ϕ, solving Equation 8 corresponds to optimizing ϕ in an $(f \times n)$-dimensional space, where n is the number of vertices in the considered mesh. A simple way to solve such an optimization problem is by applying gradient descent formulated as:

$$\boldsymbol{v}^{(i+1)} := \boldsymbol{v}^{(i)} - \alpha^{(i)}\boldsymbol{R}(\boldsymbol{v}^{(i)}), \tag{9}$$

where $\boldsymbol{v}^{(0)} = \boldsymbol{u}(t)$ and $\alpha^{(i)} \in \mathbb{R}$ is determined using line search. However, due to the high computational cost of the search, α can be fixed to a small value, which corresponds to the explicit Euler method with the time step size $\alpha\Delta t$. Barzilai & Borwein (1988) suggested another simple yet effective way to determine the step size using a two-point approximation to the secant equation underlying quasi-Newton methods.

2.2 Neural PDE solvers

We review machine learning models used to solve PDEs called neural PDE solvers, typically formulated as $\boldsymbol{u}(t_{n+1}, \boldsymbol{x}_i) \approx \mathcal{F}_{\text{NN}}(\boldsymbol{u})(t_n, \boldsymbol{x}_i)$ for $(t_n, \boldsymbol{x}_i) \in \{t_0, t_1, \dots\} \times \Omega$, where $\mathcal{F}_{\text{NN}}$ is a machine learning model.

[1] Strictly speaking, components of the PDE e.g. $\mathcal{D}$ and Ω can be different before and after discretization. However, we use the same notation regardless of discretization to keep the notation simple.

2.2.1 Physics-informed neural networks (PINNs)

Raissi et al. (2019) made a pioneering work combining PDE information and neural networks, called PINNs, by adding loss to monitor how much the output satisfies the equations. PINNs can be used to solve forward and inverse problems and extract physical states from measurements (Pang et al., 2019; Mao et al., 2020; Cai et al., 2021). However, PINNs' outputs should be functions of space because PINNs rely on automatic differentiation to obtain loss regarding PDEs. This design constraint significantly limits the model's generalization ability because the solution of a PDE could be entirely different when the shape of the domain or boundary condition changes. Therefore, these methods are not applicable in building models that are generalizable over shape and boundary condition variations.

2.2.2 Graph neural network based PDE solvers

As discussed in Section 2.1.1, one can regard a mesh as a graph. GNNs can take any graphs as inputs (Gori et al., 2005; Scarselli et al., 2008; Kipf & Welling, 2017; Gilmer et al., 2017), having the possibility to generalize over various graphs, i.e., meshes. Therefore, GNNs are strong candidates for learning mesh-structured numerical analysis data, as seen in Alet et al. (2019); Chang & Cheng (2020); Pfaff et al. (2021). Brandstetter et al. (2022) advanced these works by suggesting temporal bundling and pushforward trick for efficient and stable prediction. Their method could also consider boundary conditions by feeding them to the models as inputs. Here, one could expect the model to learn to satisfy boundary conditions approximately, while there is no guarantee to fulfill hard constraints such as Dirichlet conditions.

2.2.3 Equivariant models

In addition to GNNs, another essential concept to help machine learning models generalize is equivariance. Equivariance is characterized by using group action as $f(g \cdot x) = g \cdot f(x)$ for $f : X \to Y$ and $g \in G$ acting on X and Y. In particular, $\mathrm{E}(n)$-equivariance is essential to predict the solutions of physical PDEs because it describes rigid body motion, i.e., translation, rotation, and reflection. Ling et al. (2016) and Wang et al. (2021) introduced equivariance to a simple neural network and CNN to predict flow phenomena. Both works showed that equivariance improved predictive and generalization performance compared to models without equivariance. Horie et al. (2021) proposed $\mathrm{E}(n)$-equivariant GNNs based on GCNs (Kipf & Welling, 2017), called IsoGCNs. A form of their model is formulated as:

$$[\nabla\psi]_i \approx [\mathrm{IsoGCN}_{0\to1}(\psi)]_i := \left[\sum_{l\in\mathcal{N}_i} \frac{\boldsymbol{x}_l - \boldsymbol{x}_i}{\|\boldsymbol{x}_l - \boldsymbol{x}_i\|} \otimes \frac{\boldsymbol{x}_l - \boldsymbol{x}_i}{\|\boldsymbol{x}_l - \boldsymbol{x}_i\|}\right]^{-1} \sum_{j\in\mathcal{N}_i} \frac{\psi_j - \psi_i}{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|} \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|}, \tag{10}$$

where $\mathcal{N}_i$ is the neighborhood of the ith vertex and $\otimes$ is the tensor product operator. Here, we denote $\mathrm{IsoGCN}_{0\to1}$ an IsoGCN layer that converts the input scalar (rank-0 tensor) field ψ to the output vector (rank-1 tensor) field. This layer corresponds to the gradient operator, which helps learn PDEs because spatial derivatives such as gradient play an essential role in PDEs. They applied the model to the heat equation problem, showing high predictive performance and fast prediction, while boundary condition treatment was out of their scope.

3 Proposed method

In this section, we present our model architecture. We adopt an encode-process-decode architecture, proposed by Battaglia et al. (2018), which has been applied successfully in various previous works, e.g., Horie et al. (2021); Brandstetter et al. (2022). Our key concept is to encode input features, including information on boundary conditions, apply a GNN-based nonlinear solver loop reflecting boundary conditions in the encoded space, then decode carefully to satisfy boundary conditions in the output space.

3.1 Dirichlet boundary model

As demonstrated theoretically and experimentally in literature (Hornik, 1991; Cybenko, 1992; Nakkiran et al., 2021), the expressive power of neural networks comes from encoding in a higher-dimensional space, where the corresponding boundary conditions are not trivial. However, if there

are no boundary condition treatments in layers inside the processor, which resides in the encoded space, the trajectory of the solution can be far from the one with boundary conditions. Therefore, boundary condition treatments in an encoded space are essential for obtaining reliable neural PDE solvers that fulfill boundary conditions.

To ensure the same encoded space between variables and boundary conditions, we use the same encoder for variables and the corresponding Dirichlet boundary conditions, which we term the *boundary encoder*, as follows:

$$\boldsymbol{h}_i = \boldsymbol{f}_{\text{encode}}(\boldsymbol{u}_i) \text{ in } \Omega, \quad \hat{\boldsymbol{h}}_i = \boldsymbol{f}_{\text{encode}}(\hat{\boldsymbol{u}}_i) \text{ on } \partial\Omega_{\text{Dirichlet}} \tag{11}$$

One can easily apply Dirichlet boundary conditions in the aforementioned encoded space using the *Dirichlet layer* defined as:

$$\text{DirichletLayer}(\boldsymbol{h}_i) = \begin{cases} \boldsymbol{h}_i, & \boldsymbol{x}_i \notin \partial\Omega_{\text{Dirichlet}} \\ \hat{\boldsymbol{h}}_i, & \boldsymbol{x}_i \in \partial\Omega_{\text{Dirichlet}} \end{cases} \tag{12}$$

This process is necessary to return to the state respecting the boundary conditions after some operations in the processor, which might disrespect the conditions.

After the processor layers, we decode the hidden features using functions satisfying:

$$\boldsymbol{f}_{\text{decode}} \circ \boldsymbol{f}_{\text{encode}}(\hat{\boldsymbol{u}}_i) = \hat{\boldsymbol{u}}_i \text{ on } \partial\Omega_{\text{Dirichlet}} \tag{13}$$

This condition ensures that the encoded boundary conditions correspond to the ones in the original physical space. Demanding that Equation 13 holds for arbitrary $\hat{\boldsymbol{u}}$; we obtain $\boldsymbol{f}_{\text{decode}} \circ \boldsymbol{f}_{\text{encode}} = \text{Id}_{\boldsymbol{u}}$, resulting in $\boldsymbol{f}_{\text{decode}} = \boldsymbol{f}^{+}_{\text{encode}}$, which we call the *pseudoinverse decoder*. It is pseudoinverse because $\boldsymbol{f}_{\text{encode}}$, in particular encoding in a higher-dimensional space, may not be invertible. Therefore, we construct $\boldsymbol{f}^{+}_{\text{encode}}$ using pseudoinverse matrices and inverse activation functions. For more details, see Appendix A.

3.2 Neumann boundary model

Matsunaga et al. (2020) proposed a wall boundary model to deal with Neumann boundary conditions for the least squares moving particle semi-implicit (LSMPS) method (Tamai & Koshizuka, 2014), a framework to solve PDEs using particles. The LSMPS method is the origin of the IsoGCN's gradient operator, so one can imagine that the wall boundary model may introduce a sophisticated treatment of Neumann boundary conditions into IsoGCN. We modified the wall boundary model to adapt to the situation where the vertices are on the Neumann boundary, which differs from the situation of particle simulations. For more details of the model's derivation, see Appendix A. Our formulation of IsoGCN with Neumann boundary conditions, which is termed *NeumannIsoGCN*, is expressed as:

$$\text{NeumannIsoGCN}_{0\to 1}(\psi) := \boldsymbol{M}_i^{-1} \left[\sum_{j \in \mathcal{N}_i} \frac{\psi_j - \psi_i}{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|} \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|} + w_i \boldsymbol{n}_i \hat{g}_i \right] \tag{14}$$

$$\boldsymbol{M}_i := \sum_{l \in \mathcal{N}_i} \frac{\boldsymbol{x}_l - \boldsymbol{x}_i}{\|\boldsymbol{x}_l - \boldsymbol{x}_i\|} \otimes \frac{\boldsymbol{x}_l - \boldsymbol{x}_i}{\|\boldsymbol{x}_l - \boldsymbol{x}_i\|} + w_i \boldsymbol{n}_i \otimes \boldsymbol{n}_i, \tag{15}$$

where $\hat{g}_i$ is the value of the Neumann boundary condition at $\boldsymbol{x}_i$, and $w_i > 0$ is an untrainable parameter to control the strength of the Neumann constraint. As $w_i \to \infty$, the model strictly satisfies the given Neumann condition. However, as the directional derivatives can be computed in various directions, i.e., $(\boldsymbol{x}_j - \boldsymbol{x}_i)$, we keep the value of w_i moderate to respect information in the neighborhood. In particular, we set $w_i = 10.0$, assuming that around ten vertices may virtually exist "outside" the boundary on a flat surface in a 3D space. NeumannIsoGCN is a straightforward generalization of the original IsoGCN by letting $\boldsymbol{n}_i = \boldsymbol{0}$ when $\boldsymbol{x}_i \notin \partial\Omega_{\text{Neumann}}$. This model can also be generalized to vectors or higher rank tensors, similarly to the original IsoGCN's construction. Therefore, NeumannIsoGCN can express any spatial differential operator, constituting $\mathcal{D}$ in PDEs.

3.3 Neural nonlinear solver

As reviewed in Section 2.1, one can regard solving PDEs as optimization. Here, we adopt the Barzilai–Borwein method (Barzilai & Borwein, 1988) to solve Equation 8 in the encoded space. In

Table 1: MSE loss ($\pm$ the standard error of the mean) on test dataset of gradient prediction. $\hat{g}_{\mathrm{Neumann}}$ is the loss computed only on the boundary where the Neuman condition is set.

Method	$\nabla\phi(\times 10^{-3})$	$\hat{g}_{\mathrm{Neumann}}(\times 10^{-3})$
Original IsoGCN	192.72 ± 1.69	1390.95 ± 7.93
NeumannIsoGCN (Ours)	6.70 ± 0.15	3.52 ± 0.02

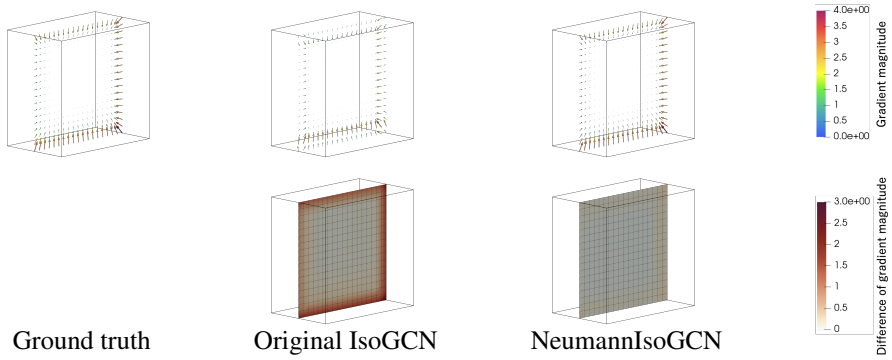

Figure 2: Gradient field (top) and the magnitude of error between the predicted gradient and the ground truth (bottom) of a test data sample, sliced on the center of the mesh.

our case, the step size $\alpha^{(i)}$ of gradient descent is approximated as:

$$\alpha^{(i)} \approx \alpha_{\mathrm{BB}}^{(i)} := \frac{\left\langle \boldsymbol{h}^{(i)} - \boldsymbol{h}^{(i-1)}, \boldsymbol{R}(\boldsymbol{h}^{(i)}) - \boldsymbol{R}(\boldsymbol{h}^{(i-1)}) \right\rangle_{\Omega}}{\left\langle \boldsymbol{R}(\boldsymbol{h}^{(i)}) - \boldsymbol{R}(\boldsymbol{h}^{(i-1)}), \boldsymbol{R}(\boldsymbol{h}^{(i)}) - \boldsymbol{R}(\boldsymbol{h}^{(i-1)}) \right\rangle_{\Omega}}, \tag{16}$$

wherer $\boldsymbol{R}(\boldsymbol{h})$ is the residual vector in the encoded space and $\langle \boldsymbol{f}, \boldsymbol{g} \rangle_{\Omega} := \sum_{\boldsymbol{x}_i \in \Omega} \boldsymbol{f}(\boldsymbol{x}_i) \cdot \boldsymbol{g}(\boldsymbol{x}_i)$ denotes the inner product over the mesh. Because the inner product is taken all over the mesh (graph), computing $\alpha_{\mathrm{BB}}^{(i)}$ corresponds to global pooling. With that view, one can find similarities between Equation 9 and deep sets (Zaheer et al., 2017), which is a successful method to learn point cloud data and has a strong background regarding permutation equivariance.

Our aim is to use Equation 16, approximating the nonlinear differential operator $\mathcal{D}$ in Equation 7 with NeumannIsoGCNs. By doing this, we expect the processor to consider both local and global information, which may have an advantage over simply stacking GNNs corresponding to the explicit method as discussed in Section 2.1.1. Combinations of solvers and neural networks are already suggested in, e.g., NeuralODE (Chen et al., 2018). The novelty of our study is the extension of existing methods for solving PDEs with spatial structure and the incorporation of global pooling into the solver, enabling us to capture global interaction, which we refer to as the *neural nonlinear solver*. Finally, the update step in the neural nonlinear solver is expressed as:

$$\boldsymbol{h}^{(i+1)} = \mathrm{DirichletLayer}\left(\boldsymbol{h}^{(i)} - \alpha_{\mathrm{BB}}^{(i)}\left[\boldsymbol{h}^{(i)} - \boldsymbol{h}^{(0)} - \mathcal{D}_{\mathrm{NeumannIsoGCN}}(\boldsymbol{h}^{(i)})\Delta t\right]\right), \tag{17}$$

where $\boldsymbol{h}^{(0)}$ is the output of the encoder and $\mathcal{D}_{\mathrm{NeumannIsoGCN}}$ is an $\mathrm{E}(n)$-equivariant GNN reflecting the structure of $\mathcal{D}$ using differential operators provided by NeumannIsoGCN. Here, Equation 17 enforces hidden features to satisfy the encoded PDE, including boundary conditions, motivating us to call our model *physics-embedded neural networks* because it embeds physics (PDEs) in the model rather than in the loss.

4 Experiments

Using numerical experiments, we demonstrate the proposed model's validity, expressibility, and computational efficiency. We use two types of datasets: 1) the gradient dataset to verify the correctness of NeumannIsoGCN and 2) the incompressible flow dataset to demonstrate the speed and accuracy of the model. We also present ablation study results to corroborate the effectiveness of the

Table 2: MSE loss ($\pm$ the standard error of the mean) on test dataset of incompressible flow. If "Trans." is "Yes," it means evaluation is done on randomly rotated and transformed test dataset. $\hat{\cdot}_{\mathrm{Dirichlet}}$ is the loss computed only on the boundary where the Dirichlet condition is set for each $\boldsymbol{u}$ and p. MP-PDE's results are based on the time window size equaling 40 as it showed the best performance in the tested MP-PDEs. For complete results, see Appendix C.

Method	Trans.	$\boldsymbol{u}$ ($\times 10^{-4}$)	p ($\times 10^{-3}$)	$\hat{\boldsymbol{u}}_{\mathrm{Dirichlet}}$ ($\times 10^{-4}$)	$\hat{p}_{\mathrm{Dirichlet}}$ ($\times 10^{-3}$)
MP-PDE TW = 20	No	$\mathbf{1.30} \pm 0.01$	1.32 ± 0.01	0.45 ± 0.01	0.28 ± 0.02
	Yes	1953.62 ± 7.62	281.86 ± 0.78	924.73 ± 6.14	202.97 ± 3.81
PENN (Ours)	No	4.36 ± 0.03	$\mathbf{1.17} \pm 0.01$	$\mathbf{0.00} \pm 0.00$	$\mathbf{0.00} \pm 0.00$
	Yes	$\mathbf{4.36} \pm 0.03$	$\mathbf{1.17} \pm 0.01$	$\mathbf{0.00} \pm 0.00$	$\mathbf{0.00} \pm 0.00$

proposed method. The implementation of our model is based on the original IsoGCN's code.[2] Our implementation is in the supplemental material, which will be online upon publication. All the details of the experiments can be found in Appendix B and C.

4.1 Gradient dataset

As done in Horie et al. (2021), we conducted experiments to predict the gradient field from a given scalar field to verify the expressive power of NeumannIsoGCN. We generated cuboid-shaped meshes randomly with 10 to 20 cells in the X, Y, and Z directions. We then generated random scalar fields over these meshes using polynomials of degree 10 and computed their gradient fields analytically. Our training, validation, and test datasets consisted of 100 samples. Table 1 and Figure 2 show that the proposed NeumannIsoGCN improves gradient prediction, especially near the boundary, showing that our model successfully considers Neumann boundary conditions.

4.2 Incompressible flow dataset

We tested the expressive power our model by learning incompressible flow in complex shapes. The corresponding nonlinear differential operator is denoted as:

$$\mathcal{D}_{\mathrm{NS}}(\boldsymbol{u}) := -(\boldsymbol{u} \cdot \nabla)\boldsymbol{u} + \frac{1}{\mathrm{Re}} \nabla \cdot \nabla \boldsymbol{u} - \nabla p, \tag{18}$$

with the incompressible condition $\nabla \cdot \boldsymbol{u} = 0$, where, in the present case, $\boldsymbol{u}$ is the flow velocity field, p is the pressure field, and Re is the Reynolds number.

4.2.1 Data

To generate the dataset, we first generated pseudo-2D shapes, with one cell in the Z direction, by changing design parameters, starting from three template shapes. Thereafter, we performed numerical analysis using a classical solver, OpenFOAM,[3] with $\Delta t = 10^{-3}$, and the initial conditions were the solutions of potential flow, which can be computed quickly and stably using the classical solver. The Reynolds number Re was around 10^3. Template shapes, design parameters, and boundary conditions used can be found in Appendix C.

To confirm the expressive power of the proposed model, we used coarse input meshes for machine learning models. We generated these coarse meshes by setting cell sizes roughly four times larger than the original numerical analysis. We obtained ground truth variables using interpolation. The task was to predict flow velocity and pressure fields at $t = 4.0$ using information available before numerical analysis, e.g., initial conditions and the geometries of the meshes. Training, validation, and test datasets consisted of 203, 25, and 25 samples, respectively. We generated the dataset by randomly rotating and translating test samples to monitor the generalization ability of machine learning models.

[2] https://github.com/yellowshippo/isogcn-iclr2021, Apache License 2.0.
[3] https://www.openfoam.com/

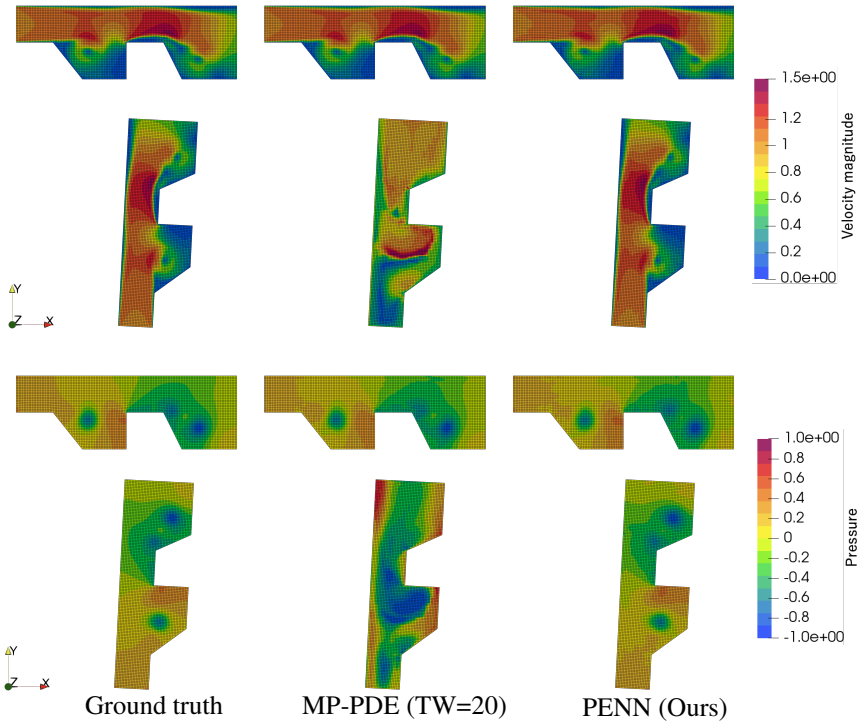

Figure 3: Comparison of the velocity field (top two rows) and the pressure field (bottom two rows) without (first and third rows) and with (second and fourth rows) random rotation and translation. PENN prediction is consistent under rotation and translation due to the $\mathrm{E}(n)$-equivariance nature of the model, while MP-PDE's predictive performance degrades under transformations.

4.2.2 Machine learning models

We constructed the PENN model corresponding to the incompressible Navier–Stokes equation. In particular, we adopted the fractional step method, where the pressure field was also obtained as a PDE solution along with the velocity field.

For the state-of-the-art baseline model, we selected MP-PDE (Brandstetter et al., 2022) as it also provides a way to deal with boundary conditions. We used the authors' code[4] with minimum modification to adapt to the task. We tested various time window sizes such as 2, 4, 10, and 20, where one step corresponds to time step size $\Delta t = 0.1$. With changes in time window size, we changed the number of hops considered in one operation of the GNN of the baseline to have almost the same number of hops visible from the model when predicting the state at $t = 4.0$. Both models were trained for up to 24 hours using one GPU (NVIDIA A100 for NVLink 40GiB HBM2).

4.2.3 Results

Table 2 and Figure 3 show the comparison between MP-PDE and PENN. The predictive performances of both models are at almost the same level when evaluated on the original test dataset. The results show the great expressive power of the MP-PDE model because we kept most settings at default as much as possible and applied no task-specific tuning. However, when evaluating them on the transformed dataset, the predictive performance of MP-PDE significantly degrades. Nevertheless, PENN shows the same loss value up to the numerical error, confirming $\mathrm{E}(n)$-equivariance. In addition, PENN exhibits no error on the Dirichlet boundaries, showing that our treatment of Dirichlet boundary conditions is rigorous.

Figure 4 shows the speed-accuracy trade-off for OpenFOAM, MP-PDE, and PENN. We varied mesh cell size and the time step size for OpenFOAM to have different computation speeds and accuracy.

[4] https://github.com/brandstetter-johannes/MP-Neural-PDE-Solvers

Table 3: Ablation study on the incompressible flow dataset. The value represents MSE loss ($\pm$ standard error of the mean) on the test dataset. "Divergent" means the implicit solver does not converge and the loss gets extreme value ($\sim 10^{14}$).

Method	$\boldsymbol{u}$ ($\times 10^{-4}$)	p ($\times 10^{-3}$)	$\hat{\boldsymbol{u}}_{\text{Dirichlet}}$ ($\times 10^{-4}$)	$\hat{p}_{\text{Dirichlet}}$ ($\times 10^{-3}$)
Without encoded boundary	Divergent	Divergent	Divergent	Divergent
Without neural nonlinear solver	31.03 ± 0.19	9.81 ± 0.04	$\mathbf{0.00} \pm 0.00$	$\mathbf{0.00} \pm 0.00$
Without boundary condition input	20.08 ± 0.21	3.61 ± 0.02	59.60 ± 0.89	1.43 ± 0.05
Without Dirichlet layer	8.22 ± 0.07	1.41 ± 0.01	18.20 ± 0.28	0.38 ± 0.01
Without pseudoinverse decoder	8.91 ± 0.06	2.36 ± 0.02	1.97 ± 0.06	$\mathbf{0.00} \pm 0.00$
Without pseudoinverse decoder with Dirichlet layer after decoding	6.65 ± 0.05	1.71 ± 0.01	$\mathbf{0.00} \pm 0.00$	$\mathbf{0.00} \pm 0.00$
PENN	$\mathbf{4.36} \pm 0.03$	$\mathbf{1.17} \pm 0.01$	$\mathbf{0.00} \pm 0.00$	$\mathbf{0.00} \pm 0.00$

The proposed model achieved the best performance in speed-accuracy trade-off between all the tested methods under fair comparison conditions.

Table 3 presents the results of the ablation study. Comparison between models with and without the proposed components shows that the proposed components, i.e., the boundary encoder, Dirichlet layer, pseudoinverse decoder, and neural nonlinear solver, significantly improve the models. The neural nonlinear solver in the encoded space turned out to have the biggest impact on the performance, while the Dirichlet layer ensured reliable models that strictly respect Dirichlet boundary conditions.

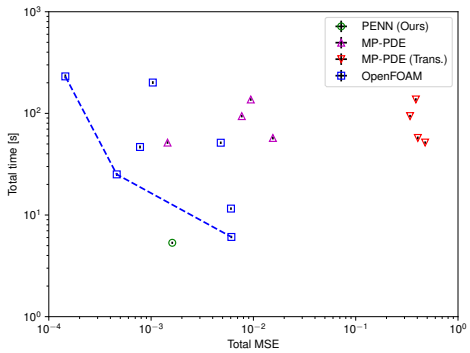

Figure 4: Comparison of computation time and total MSE loss ($\boldsymbol{u}$ and p) on the test dataset (with and without transformation) between OpenFOAM, MP-PDE, and PENN. The error bar represents the standard error of the mean. All computation was done using one core of Intel Xeon CPU E5-2695 v2@2.40GHz.

5 Conclusion

We have presented an $\mathrm{E}(n)$-equivariant, GNN-based neural PDE solver, PENN, which can fulfill boundary conditions required for reliable predictions. The model has superiority in embedding the information of PDEs (i.e., physics) in the model and speed-accuracy trade-off. Therefore, our model can be a useful standard for realizing reliable, fast, and accurate GNN-based PDE solvers. Although the property of our model is preferable, it also limits the applicable domain of the model. For instance, the proposed model cannot exploit its potential to solve inverse problems where explicit forms of the governing PDE are not available for such tasks. Therefore, combining PINNs and our model, PENNs, could be the next direction of the research community.

6 Potential negative societal impacts

We have built a foundation to learn PDEs in a steerable manner rather than focusing on a specific application. Because of that, we envisage minimal risk of direct abusing our present work. However, as mentioned in the introduction, solving PDEs has many impacts on various domains, from both positive and negative aspects. Thus, our work and possible successive ones may be abused, aiming to harm lives and the environment. Therefore, the research community, including us, must be careful in using them and control the research direction to prevent abusing machine learning and simulation technologies.

References

Ferran Alet, Adarsh Keshav Jeewajee, Maria Bauza Villalonga, Alberto Rodriguez, Tomas Lozano-Perez, and Leslie Kaelbling. Graph element networks: adaptive, structured computation and memory. In *ICML*, 2019.

Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1):141–148, 1988.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=vSix3HPYKSU.

Shengze Cai, Zhicheng Wang, Frederik Fuest, Young Jin Jeon, Callum Gray, and George Em Karniadakis. Flow over an espresso cup: inferring 3-d velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks. *Journal of Fluid Mechanics*, 915, 2021.

Kai-Hung Chang and Chin-Yi Cheng. Learning to simulate and design for structural engineering. *arXiv preprint arXiv:2003.09103*, 2020.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control. Signals Syst.*, 5(4): 455, 1992.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1263–1272. JMLR. org, 2017.

Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.

Deguang Han, Keri Kornelson, Eric Weber, and David Larson. *Frames for undergraduates*, volume 40. American Mathematical Soc., 2007.

Masanobu Horie, Naoki Morita, Toshiaki Hishinuma, Yu Ihara, and Naoto Mitsume. Isometric transformation invariant and equivariant graph convolutional networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=FX0vR39SJ5q.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

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

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

Takuya Matsunaga, Axel Södersten, Kazuya Shibata, and Seiichi Koshizuka. Improved treatment of wall boundary conditions for a particle method with consistent spatial discretization. *Computer Methods in Applied Mechanics and Engineering*, 358:112624, 2020.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

Tasuku Tamai and Seiichi Koshizuka. Least squares moving particle semi-implicit method. *Computational Particle Mechanics*, 1(3):277–305, 2014.

Rui Wang, Robin Walters, and Rose Yu. Incorporating symmetry into deep dynamics models for improved generalization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=wta_8Hx2KD.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the abstract and Section 1.
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 6.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Our paper, in particular Section 6, is written in accordance with the guideline.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] They are included in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] They are included in the supplemental material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4, Table 1, 2, 3, and Figure 4.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Footnote 2.
   (b) Did you mention the license of the assets? [Yes] See Footnote 2.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] It is included in the supplemental material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The dataset we used in the paper is generated by ourselves.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The dataset is based on numerical analysis, so there is no concern about personally identifiable information nor offensive content.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]