An Implicit GNN Solver for Poisson-like problems

Matthieu Nastorg, Michele Alessandro Bucci, Thibault Faney, Jean-Marc Gratien, Guillaume Charpiat, Marc Schoenauer

An Implicit GNN Solver for Poisson-like Problems

Matthieu Nastorg[a,b,*], Michele-Alessandro Bucci[a,c], Thibault Faney[b], Jean-Marc Gratien[b], Guillaume Charpiat[a], Marc Schoenauer[a]

[a] *Université Paris-Saclay, CNRS, Inria, Laboratoire interdisciplinaire des sciences du numérique, 91405, Orsay, France*
[b] *IFP Energies nouvelles, 92852, Rueil-Malmaison, France*
[c] *Safran Tech, Digital Sciences & Technologies Department, 78114, Châteaufort, France*

Abstract

This paper presents Ψ-GNN, a novel Graph Neural Network (GNN) approach for solving the ubiquitous Poisson PDE problems on general unstructured meshes with mixed boundary conditions. By leveraging the Implicit Layer Theory, Ψ-GNN models an "infinitely" deep network, thus avoiding the empirical tuning of the number of required Message Passing layers to attain the solution. Its original architecture explicitly takes into account the boundary conditions, a critical pre-requisite for physical applications, and is able to adapt to any initially provided solution. Ψ-GNN is trained using a physics-informed loss, and the training process is stable by design. Furthermore, the consistency of the approach is theoretically proven, and its flexibility and generalization efficiency are experimentally demonstrated: the same learned model can accurately handle unstructured meshes of various sizes, as well as different boundary conditions. To the best of our knowledge, Ψ-GNN is the first physics-informed GNN-based method that can handle various unstructured domains, boundary conditions and initial solutions while also providing convergence guarantees.

1. Introduction

Partial differential equations (PDEs) are highly detailed mathematical models that describe complex physical or artificial processes in engineering and science (Olver, 2013). Although they have been widely studied for many years, solving these equations at scale remains daunting, limited by the computational cost of resolving the smallest spatio-temporal scales (Morton and Mayers, 2005). One of the most common and important PDEs in engineering is the steady-state *Poisson* equation, mathematically described as $-\Delta u = f$ on a domain $\Omega \subset \mathbb{R}^n$ (usually, $n = 2$ or 3) with a given *forcing term* f, where u is the solution to be sought. Besides, special constraints (often referred to as *boundary conditions*) must be specified on the boundary $\partial\Omega$ of Ω (Langtangen and

*Corresponding author
Email address: matthieu.nastorg@inria.fr (Matthieu Nastorg)

Mardal, 2019) to ensure the existence and uniqueness of the solution. The Poisson equation appears in various fields such as Fluid mechanics (Guermond and Quartapelle, 1998), Gravity (Mannheim and Kazanas, 1994), Electrostatics (Luty et al., 1992), or Surface reconstruction (Kazhdan et al., 2006). It is ubiquitous and plays a central role in modern numerical solvers. Despite progresses in the High-Performance Computing (HPC) community, solving large Poisson problems is achievable only by employing robust yet tedious iterative methods (Saad, 2003) and remains one of the major bottlenecks in the speedup of industrial numerical simulations.

More recently, data-driven methods based on deep neural networks have been reshaping the domain of numerical simulation. Neural networks can provide faster predictions, reducing the turnaround time for workflows in engineering and science (Wiewel et al., 2019; Um et al., 2020; Kochkov et al., 2021) – see also our quick survey in Section 2. However, the lack of guarantees about the consistency and convergence of deep learning approaches makes it non-viable to implement these models in the design and production stage of new engineering solutions.

This work introduces Ψ-GNN[1], an Implicit Graph Neural Network (GNN) approach that iteratively solves a Poisson problem on general unstructured meshes with mixed boundary conditions. Leveraging Implicit Layer Theory (Bai et al., 2019), the proposed model controls, by itself, the number of Message Passing layers needed to reach the solution, yielding excellent out-of-distribution generalization to mesh sizes and shapes. The Ψ-GNN architecture, detailed in Section 3, is based on a node-oriented approach which inherently respects the boundary conditions, and an additional autoencoding process allows it to consider any initial solution and dynamically adapt to it. The method is trained end-to-end, minimizing the residual of the discretized Poisson problem, thus attempting to learn the physics of the problem. When Neumann boundary conditions are present, an additional lightly-supervised loss (MSE with ground-truth solutions) is necessary to help convergence. To ensure the stability of the model, a regularizing cost function (Bai et al., 2021) is also used to constrain the spectral radius of the Machine Learning solver, thus providing strong convergence guarantees. In Section 4, we provide a theoretical analysis, demonstrating that the proposed Ψ-GNN approach satisfies a property of universal approximation. Specifically, we prove that there exists a parametrization of our model that yields an optimal solution for the considered task, showcasing the consistency of our approach. Moreover, we verify its generalization ability in Section 5 through experiments with various geometries and boundary conditions. We also assess the performance of our approach by comparing it with a state-of-the-art Machine Learning model and discuss its complexity.

[1] Poisson Solver Implicit Graph Neural Network

Significant Contributions of this Study:

- Introduction of a novel approach that combines Graph Neural Networks and Implicit Layer Theory for effectively addressing Poisson problems with mixed boundary conditions.
- Development of a flexible model capable of accommodating diverse mesh sizes and shapes, as well as handling various boundary conditions and initial solutions, demonstrating excellent out-of-distribution generalization.
- Implementation of a robust training process that learns the physics of the equation and uses stabilization techniques to provide strong convergence guarantees.

2. Related Work

In the past few years, the use of Machine Learning models to predict solutions of PDEs has gained significant interest in the community, beginning in the 90s with some pioneer works of Lee and Kang (1990); Dissanayake and Phan-Thien (1994) and Lagaris et al. (1998). Since then, much research has focused on building more complex neural network architectures with a larger number of parameters, taking advantage of the increasing computational power as demonstrated in Smaoui and Al-Enezi (2004); Baymani et al. (2010) or Kumar and Yadav (2011).

CNNs for physics simulations Despite these convincing advances that employed fully connected neural networks, these methods were quickly overtaken by the tremendous progress in computer vision and the rise of Convolutional Neural Networks (CNNs) (LeCun et al., 1995; Gu et al., 2018). For instance, in fluid mechanics, such networks have been used to solve the Navier-Stokes equations, considering solutions on rectangular grids and treating them as images (Guo et al., 2016; Yilmaz and German, 2017; Obiols-Sales et al., 2020; Illarramendi et al., 2021). A significant amount of research has focused on using CNNs to solve the Poisson equation due to its significant engineering interest. In Tang et al. (2017), a straightforward CNN architecture predicts the potential electric distribution in a square domain by approximating the solution of 2D and 3D Poisson problems. Hsieh et al. (2019) design a Machine Learning solver with a U-Net architecture (Ronneberger et al., 2015) to mimic multigrid methods (Briggs et al., 2000) and provide theoretical convergence guarantees when applied to the resolution of a 2D Poisson equation. In Özbay et al. (2021), a convolutional neural network is trained to solve the inverse Poisson problem through supervised learning. In Cheng et al. (2021), the authors use a physics-based loss function with a deep convolutional network to solve the Poisson equation in the context of plasma flows. These different approaches have shown promising results, providing rather accurate approximate solutions that are also computed faster than traditional solvers. However, there are limitations to their application. They can only be used with structured meshes with

uniform discretization, which makes them incompatible with classical methods that rely on unstructured "*meshes*". Another issue is that the treatment of boundary conditions is often overlooked or not properly addressed. If attempts are made to consider them, it is usually done by adding an extra loss function or using an additional specialized model. Unfortunately, these approaches tend to perform poorly when dealing with new problem configurations, as they struggle to generalize effectively.

GNNs for physics simulations To address these shortcomings, recent studies have focused on Graph Neural Networks (GNNs), a class of neural networks that can learn from unstructured data. Introduced in Battaglia et al. (2016), GNNs have experienced significant growth and seen a variety of applications thanks to the development of new techniques such as graph convolution (Kipf and Welling, 2016), edge convolution (Hamilton et al., 2017), graph attention networks (Veličković et al., 2017), or graph pooling (Lee et al., 2019) to name a few. An exhaustive survey on existing GNN architectures can be found in Wu et al. (2020). With regard to physical applications, several recent works have shown the ability of GNNs to learn dynamical systems accurately. For example, in Chang et al. (2016) and Sanchez-Gonzalez et al. (2018), GNNs are used to learn the motion of discrete systems of solid particles. Sanchez-Gonzalez et al. (2020) extend this approach to learn complex physics, including fluid simulation and solid deformation, considering graph nodes as particles. In Pfaff et al. (2020), the authors simulate the time dynamics of complex systems based on unstructured data. Li et al. (2020b) and Lino et al. (2021) propose using a multi-level architecture to solve PDEs on graphs with a larger number of nodes. In Horie and Mitsume (2022), a GNN model is trained through supervised learning to approximate the solution of the incompressible Navier-Stokes equation while preserving the boundary conditions. Similar to CNN-oriented research, some studies have focused on using GNNs to solve the Poisson equation, starting with the work of Alet et al. (2019). Li et al. (2020a) introduce a graph kernel network to approximate PDEs with a specific focus on the resolution of a 2D Poisson problem, and in Chen et al. (2022), a multi-level GNN architecture is trained through supervised learning to solve the Poisson Pressure equation in the context of fluid simulations. These approaches outperform the CNN-based models due to their generalization to unstructured meshes. Nevertheless, these methods still rely solely on supervised learning, requiring computationally expensive ground truth solutions and resulting in poor performances when applied to out-of-distribution examples. Additionally, the explicit consideration of boundary conditions remains elusive, presenting a significant challenge for practical use in industrial processes.

The Physics-Informed approach In parallel to these architectural advancements, another research direction focuses on a new class of Deep Learning method called *Physics-Informed Neural Networks* (PINNs), which has emerged as a very promising tool for solving scientific computational problems (Raissi et al., 2019; Raissi and Karniadakis, 2018). These methods are mesh-free and

specifically designed to integrate the PDE residual into the training loss. To do that, PINNs leverage Automatic Differentiation (Paszke et al., 2017) to compute the PDE's derivatives. Numerous works have used this approach to solve more complex problems, such as in fluid mechanics as demonstrated in Wu et al. (2018); Mao et al. (2020); Rao et al. (2020); Jin et al. (2021), or Cai et al. (2022). Therefore, the approach of training a model by minimizing the residual equation is well known and has been combined with GNNs to solve a 2D Poisson equation as in Gao et al. (2022). However, PINNs face challenges in generalizing to new scenarios due to the significant changes that can occur in the solution to a PDE when considering different domain shapes or boundary conditions. Moreover, effectively addressing the boundary conditions often requires the inclusion of additional loss terms, which must be carefully weighted to achieve optimal performance. Furthermore, PINNs lack interpretability. In our approach, we build upon established numerical methods by directly minimizing the residual of the discretized Poisson problem. By doing so, we inherit the advantageous properties of the chosen discretization method, i.e. the Finite Element method (FEM). As a result, our model adopts a mesh-based structure and effectively incorporates the strengths and characteristics of the FEM. Consequently, our approach demonstrates enhanced performance and an improved ability to generalize across different domains, boundary conditions, and initial solutions.

Deep Statistical Solvers (DSS) Our work is closely related to Donon et al. (2020), where a GNN is used to solve a Poisson problem with Dirichlet boundary conditions efficiently. We propose a novel GNN-based architecture that can handle mixed boundary conditions, making it extensible to CFD cases. In previous work, the number of Message Passing Neural Networks (MPNN) needed to achieve convergence was fixed and was shown to be proportional to the diameter of meshes considered in the dataset. Nastorg et al. (2022) further improve upon that work by introducing a Recurrent Graph architecture which significantly reduces the model size. Besides, they show that if the model is trained with a sufficient number of MPNN iterations, it tends to converge toward a fixed point. However, the number of iterations remains fixed, and the model has poor generalization capabilities to different mesh sizes. To address this issue, we propose using the Implicit Layer Theory Bai et al. (2019) to model an infinitely deep neural network. Furthermore, we propose an architecture that employs an autoencoding process and a node-specific approach to dynamically adapt to any given initial solution and ensure that the boundary conditions are respected by design.

Building upon the advancements made by these state-of-the-art models, our research aims to create a Machine Learning algorithm that can effectively and accurately solve a wide range of Poisson problems. In particular, we aim to enhance the existing DSS framework by proposing a novel GNN-"physics-oriented" model. **This model is specifically designed to address the challenges posed by unstructured meshes with varying sizes and shapes while ensuring the explicit incorporation of boundary conditions.**

3. Methodology

In this section, we detail the methodology employed in this study. We begin by stating the problem in 3.1. Then, in 3.2, we delve deeper into the complexity of the problem by providing a graph interpretation and presenting the statistical problem at hand. In 3.3, we outline the design of our model, providing a comprehensive description of its architecture. Additionally, we discuss in 3.4 the regularization technique employed to ensure the stability of the method and provide further details on the training process in 3.5.

3.1. Problem statement

Let $\Omega \subset \mathbb{R}^n$ be a bounded open domain with smooth boundary $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$. To ensure the existence and uniqueness of the solution, special constraints (referred to as *boundary conditions*) must be specified on the boundary $\partial\Omega$ of Ω (Langtangen and Mardal, 2019): *Dirichlet conditions* assign a known value for the solution u on $\partial\Omega_D$, and *homogeneous Neumann boundary conditions* impose that no part of the solution u is leaving the domain through $\partial\Omega_N$. More formally, let f be a continuous function defined on Ω, g a continuous function defined on $\partial\Omega_D$ and n the outward normal vector defined on $\partial\Omega$. The Poisson problem with mixed boundary conditions (i.e. Dirichlet and homogeneous Neumann boundary conditions) consists in finding a real-valued function u, defined on Ω, that satisfies:

$$\begin{cases} -\Delta u &= f & \in \Omega \\ u &= g & \in \partial\Omega_D \\ \frac{\partial u}{\partial n} &= 0 & \in \partial\Omega_N \end{cases} \tag{1}$$

Except in very specific instances, no analytical solution can be derived for the Poisson problem, and its solution must be numerically approximated: the domain Ω is first discretized into an unstructured mesh, denoted Ω_h. The Poisson equation (1) is then spatially discretized using the Finite Element Method (FEM) (Reddy, 2019). The approximate solution u_h is sought as a vector of values defined on all N degrees of freedom of Ω_h. N in turn depends on the type of elements chosen (i.e., the precision order of the approximation wanted): choosing first-order Lagrange elements, N matches the number of nodes in Ω_h. The discretization of the variational formulation using Galerkin's method leads to solving a linear system of the form:

$$AU = B \tag{2}$$

where the sparse matrix $A \in \mathbb{R}^{N\times N}$ is the discretization of the continuous Laplace operator, the vector $B \in \mathbb{R}^N$ comes from the discretization of the forcing term f and of the mixed boundary conditions, and $U \in \mathbb{R}^N$ is the solution vector to be sought. Details regarding the derivation of the linear system (2) can be found in Appendix A.

Let $\mathcal{F}$ be a set of continuous functions on Ω and $\mathcal{G}$ a set of continuous functions

on $\partial\Omega_D$. We denote as $\mathcal{P}$ a set of Poisson problems, such that any element $E_p \in \mathcal{P}$ is described as a triplet:

$$E_p = (\Omega_p,\ f_p,\ g_p)$$

where $f_p \in \mathcal{F}$ and $g_p \in \mathcal{G}$. For all $E_p \in \mathcal{P}$, let $E_{h,p} \in \mathcal{P}_h$ denote its discretization, such that:

$$E_{h,p} = (\Omega_{h,p},\ A_p,\ B_p)$$

where A_p and B_p are defined as in (2)).

The fundamental idea of Ψ-GNN is, considering a continuous Poisson problem $E_p \in \mathcal{P}$, to build a Machine Learning solver, parametrized by a vector θ, which outputs a solution U_p of its respective discretized form $E_{h,p} \in \mathcal{P}_h$:

$$U_p \ = \ \Psi\text{-GNN}_\theta\left(E_{h,p}\right) = \ \Psi\text{-GNN}_\theta\left(\Omega_{h,p},\ A_p,\ B_p\right) \tag{3}$$

3.2. Statistical problem

In (2), it should be noted that the structure of matrix A encodes the geometry of its corresponding mesh (i.e. A can be viewed as the adjacency matrix of its corresponding mesh). Indeed, for each node, using first-order finite elements leads to the creation of local stencils, which represent local connections between mesh nodes. When using higher-order finite elements, the number of degrees of freedom does not match the number of nodes in the mesh, and the previous assumptions are no longer valid. Hence, this work exclusively focuses on the use of first-order finite elements.

A crucial upside of using GNNs in physics simulations is related to the right treatment of boundary conditions. In the linear system given by (2), implementing Dirichlet boundary conditions involves modifying the sparse matrix A by setting the off-diagonal elements of the rows corresponding to Dirichlet nodes to 0 and the diagonal element to 1. Similarly, the corresponding index in vector B is set to the discrete value of g. As a result, when solving (2), Dirichlet conditions are enforced, ensuring that $u = g$. Figure 1a displays the sparsity pattern of matrix A for a problem with 17 nodes before applying Dirichlet boundary conditions. In contrast, Figure 1b illustrates the sparsity pattern of matrix A for the same problem after applying Dirichlet boundary conditions. This comparison reveals that such modifications break the symmetry of the matrix A. As a consequence, some Interior connections (blue squares in Figure 1b) and Neumann connections (yellow squares in Figure 1b), linked to Dirichlet nodes no longer have symmetrical counterparts. As a result of this specific construction, the induced graph is *directed* at those Dirichlet boundary nodes, sending information only to its neighbours without receiving any. Conversely, Interior and Neumann nodes induce an *undirected* graph with bi-directional edges, thus exchanging information with each other. To further illustrate this, Figure 1c displays the graph obtained using the adjacency matrix depicted in 1a (i.e. *before* applying Dirichlet boundary conditions). Since 1a is symmetrical, all edges

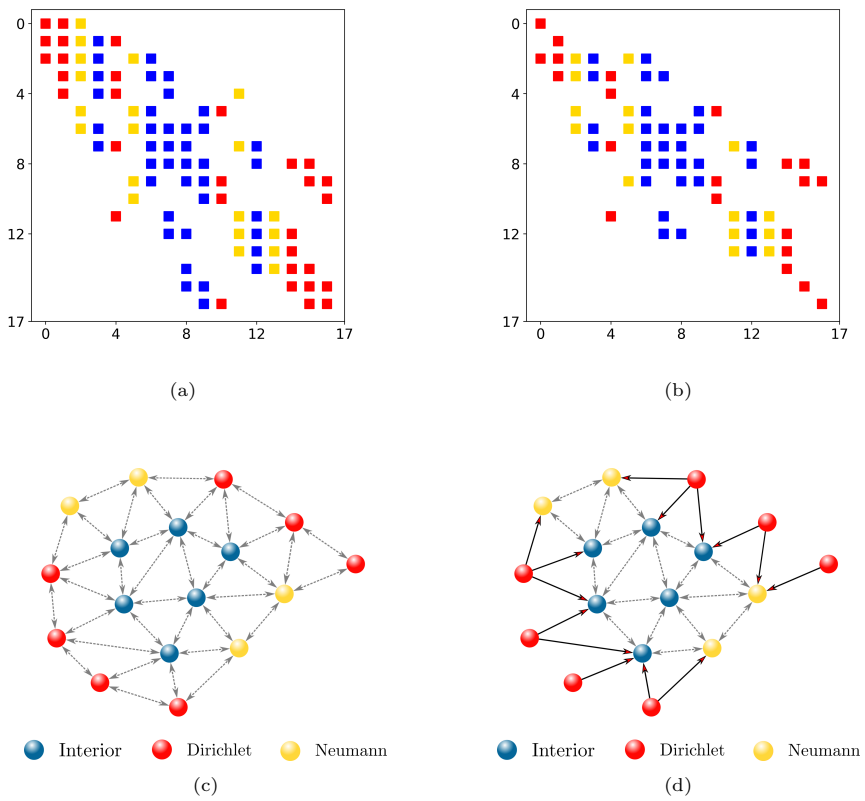


Figure 1: (1a) illustrates the sparsity pattern in matrix A, obtained by discretizing the Laplace operator in (1) using FEM for a problem with 17 nodes *before* applying Dirichlet boundary conditions. Using this matrix as an adjacency matrix, the induced graph is shown in (1c), resulting in a fully undirected graph. (1b) displays the sparsity pattern of the same matrix A *after* applying Dirichlet boundary conditions. The related graph is shown in (1d), resulting in an undirected graph for Interior and Neumann nodes (blue and yellow nodes) and a directed graph for Dirichlet nodes (red nodes). For both (1a) and (1b), Interior, Neumann and Dirichlet connections correspond to the blue, yellow and red squares, respectively.

are bi-directional. On the contrary, Figure 1d illustrates the graph obtained by considering the adjacency matrix 1b (i.e. *after* applying Dirichlet boundary conditions), showcasing the specific directionality of the edges based on the node types. Furthermore, the end of Appendix A presents a concise yet illustrative example that facilitates a comprehensive mathematical understanding of why the sparse matrix A induces such a graph structure.

A discretized Poisson problem $E_h = (\Omega_h,\ A,\ B)$ with N degrees of freedom can then be interpreted as a graph problem $G = (N,\ A,\ B)$, where N is the number of nodes in the graph, $A = (a_{ij})_{(i,j)\in[N]^2}$ is the weighted adjacency matrix that

represents the interactions between the nodes and $B = (b_i)_{i\in[N]}$ is the local external input. Vector $U = (u_i)_{i\in[N]}$ represents the state of the graph, u_i being the state of node i.

Additionally, let $\mathcal{S}$ be the set of all such graphs G, $\mathcal{U}$ the set of all states U, and $\mathcal{L}_{\text{res}}$ the real-valued function which computes the mean squared error (MSE) of the discretized residual equation:

$$\mathcal{L}_{\text{res}}(U,\ G) = \text{MSE}\left(AU - B\right) \tag{4}$$

$$= \frac{1}{N}\sum_{i\in[N]}\Big(-b_i + \sum_{j\in[N]} a_{i,j}u_j\Big)^2 \tag{5}$$

Our goal is, given a graph G in $\mathcal{S}$, to find an optimal state in $\mathcal{U}$ that minimizes (5). Therefore, we define a Machine Learning algorithm Ψ-GNN, parameterized by θ, that predicts from G a solution U in order to solve the following statistical problem:

Given a distribution $\mathcal{D}$ on space $\mathcal{S}$ and a loss function $\mathcal{L}_{res}$, solve:

$$\widehat{\theta} = \underset{\theta}{\text{argmin}} \underset{G\sim\mathcal{D}}{\mathbb{E}}\left[\mathcal{L}_{\text{res}}\left(\Psi\text{-GNN}_\theta\left(G\right), G\right)\right] \tag{6}$$

Ψ-GNN is trained using the loss function described in (5), which involves computing the matrix A and vector B in (2), obtained from the Finite Element method. However, the model only takes as input the mesh structure (converted into a graph), distances between nodes (used as edge features), and discretized values of the forcing term f and boundary function g (used as node features). As a result, once the model is trained, inference only requires information about the mesh and the original problem, eliminating the need for the expensive computation of the elements in the linear system (2).

3.3. Architecture

Figure 2 gives a global view of Ψ-GNN[2], a Graph Neural Network model with three main components: an *Encoder*, a *Processor*, and a *Decoder*. The "Encoder-Decoder" mechanism facilitates the connection between the physical space, where the solution lives, and the latent space, where the GNN layers are applied. The Processor is the core component of the model, responsible for propagating the information correctly within the graph. It is specifically designed with two key features: i) it automatically controls the number of Message Passing steps required for convergence; ii) it properly takes into account the boundary conditions by design.

Encoder The encoder E_θ, designed as a multilayer perceptron (MLP), maps an initial solution $U^0 \in \mathcal{U}$ to a $d-$dimensional latent state $H^0 \in \mathcal{H}$, $d > 1$. The

[2]code will be available after acceptance

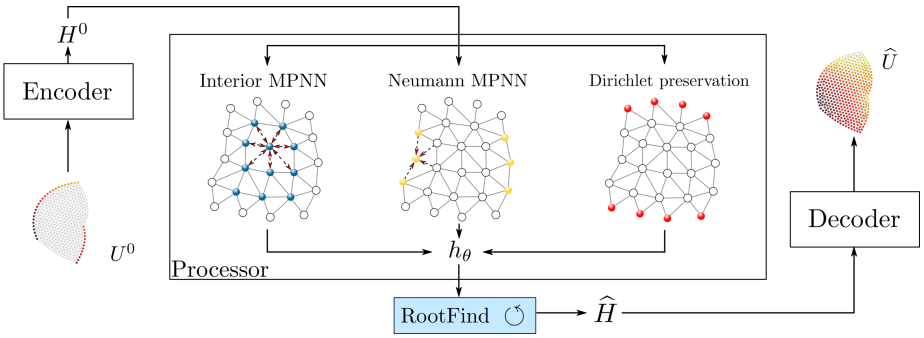


Figure 2: Diagram of Ψ-GNN: The model uses an *Encode-Process-Decode* architecture. The encoder maps an initial solution U^0 to some latent representation H^0. The processor outputs a final latent state $\widehat{H}$ by considering a different treatment for each node type. Dirichlet nodes are preserved during the process, and specific MPNN (red arrows) for Interior and Neumann nodes are computed to build a GNN function h_θ. A black-box "root-finding" solver automatically propagates the information through the graph by finding the fixed point $\widehat{H}$ of h_θ, starting from the initial guess H^0. The decoder maps $\widehat{H}$ back to the physical space to get the final solution $\widehat{U}$.

provided initial solution must fulfil the Dirichlet boundary conditions. This trainable function projects the physical space $\mathcal{U}$ to a higher dimensional latent space $\mathcal{H}$ on which the GNN layers will be applied.

Processor The processor uses a specialized approach for each node type to ensure consistency with the boundary conditions. To propagate the information, the processor constructs a GNN-based function h_θ that updates both the *Interior* and *Neumann* nodes, effectively capturing the distinct stencils of the discretized Laplace operator. For *Dirichlet* boundary nodes, the corresponding latent variable is kept constant, equal to the imposed value.

Interior nodes messages Two separate messages are computed for each node, corresponding to the outgoing and incoming links, using trainable MLPs $\Phi^{\mathtt{I}}_{\rightarrow,\theta}$ and $\Phi^{\mathtt{I}}_{\leftarrow,\theta}$ such that:

$$\phi^{\mathtt{I}}_{\rightarrow,i} = \sum_{j\in\mathcal{N}(i)} \Phi^{\mathtt{I}}_{\rightarrow,\theta}\left(H_i, H_j, d_{ij}, \|d_{ij}\|\right) \tag{7}$$

$$\phi^{\mathtt{I}}_{\leftarrow,i} = \sum_{j\in\mathcal{N}(i)} \Phi^{\mathtt{I}}_{\leftarrow,\theta}\left(H_i, H_j, d_{ji}, \|d_{ji}\|\right) \tag{8}$$

where $j \in \mathcal{N}(i)$ stands for all the nodes j in the one-hop neighbourhood of i, and d_{ij} and $\|d_{ij}\|$ represent the relative position vector and the euclidean

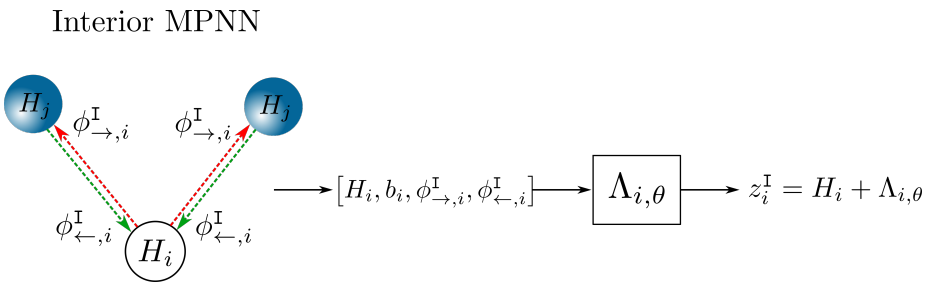


Figure 3: Process of updating the Interior latent variable $z^{\mathtt{I}}_i$: Firstly, two MPNNs, $\Phi^{\mathtt{I}}_{\rightarrow,\theta}$ (red arrows) and $\Phi^{\mathtt{I}}_{\leftarrow,\theta}$ (green arrows), are computed to account for the bi-directionality of the edges. These computed messages are then concatenated with problem-specific data b_i and the actual latent state H_i and passed through a trainable function $\Lambda_{i,\theta}$. The output of this trainable function is finally used to calculate $z^{\mathtt{I}}_i$ in a Res-Net fashion.

distance[3].The updated Interior latent variable $\mathbf{z}^{\mathtt{I}} \coloneqq (z^{\mathtt{I}}_i)_{i\in[N]}$ is computed in a Res-Net fashion of the form:

$$z^{\mathtt{I}}_i = H_i + \Lambda_{i,\theta}\left(H_i, b_i, \phi^{\mathtt{I}}_{\rightarrow,i}, \phi^{\mathtt{I}}_{\leftarrow,i}\right) \tag{9}$$

where $\Lambda_{i,\theta}$ is a trainable function designed to stabilize the updating process. The construction of $\Lambda_{i,\theta}$ is detailed in Appendix B. Figure 3 displays the process of updating the Interior node variable.

Neumann nodes messages One message from an incoming link is computed and designed to capture the stencil that ensures homogeneous Neumann boundary conditions. It is constructed in a similar manner to (8), using the MLP $\Phi^{\mathtt{N}}_{\leftarrow,\theta}$ such that:

$$\phi^{\mathtt{N}}_{\leftarrow,i} = \sum_{j\in\mathcal{N}(i)} \Phi^{\mathtt{N}}_{\leftarrow,\theta}\left(H_i, H_j, d_{ji}, \|d_{ji}\|\right) \tag{10}$$

The updated Neumann latent variable $\mathbf{z}^{\mathtt{N}} \coloneqq (z^{\mathtt{N}}_i)_{i\in[N]}$ is computed by combining message (10) with problem-related data b_i and the information on the normal vector n_i, and passing the result through an MLP Ψ_θ as follows:

$$z^{\mathtt{N}}_i = \Psi_\theta\left(H_i, b_i, n_i, \phi^{\mathtt{N}}_{\leftarrow,i}\right) \tag{11}$$

GNN-based function The GNN-based function h_θ, designed to preserve Dirichlet boundary values and separate Interior and Neumann messages, is given by:

$$h_\theta(H, G) = \begin{cases} H^0 & \text{if Dirichlet} \\ \mathrm{LN}\left(\mathbf{z}^{\mathtt{I}}\right) & \text{if Interior} \\ \mathrm{LN}\left(\mathbf{z}^{\mathtt{N}}\right) & \text{if Neumann} \end{cases} \tag{12}$$

[3]For two nodes i and j with coordinates (x_i, y_i) and (x_j, y_j), the relative position vector is $d_{ij} = (x_i - x_j, y_i - y_j)$ and the euclidean distance $\|d_{ij}\| = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

where LN stands for the Layer Normalization operation (Ba et al., 2016), which consists in normalizing each sample in the minibatch such that the features in the sample have zero mean and unit variance. This operation plays a crucial role in stabilizing h_θ by constraining its output, resulting in more efficient computation of the subsequent fixed point problem.

Fixed-point problem One step of message passing only propagates information from one node to its immediate neighbours. In order to propagate information throughout the graph, previous works (Nastorg et al., 2022) repeatedly performed the message passing step, i.e., looped over the function h_θ, either for a fixed number of iterations or until the problem converges. Iterating until convergence amounts to solving a fixed-point problem

$$\widehat{H} = h_\theta\left(\widehat{H},\ G\right) \tag{13}$$

Hence we propose to use a black-box root-finding procedure to directly solve the fixed-point problem:

$$\widehat{H} = \mathrm{RootFind}\left(h_\theta\left(H,\ G\right) - H\right) \tag{14}$$

This approach eliminates the need for a predefined number of iterations of h_θ and only requires a threshold precision of the root-finding solver, resulting in a more adaptable and flexible approach. Additional information concerning the motivations behind the use of a fixed point solver can be found in Appendix C, particularly in Figure C.11. To solve (14), Newton's method is the method of choice, thanks to its fast convergence guarantees. However, to avoid the costly computation of the inverse Jacobian at each Newton iteration, we will use the quasi-Newton Broyden algorithm (Broyden, 1965), which uses low-rank updates to maintain an approximation of the Jacobian. It is important to note that, regardless of the chosen root-finding solver, the algorithm starts with the initial guess H^0.

Decoder The decoder D_θ, designed as an MLP, maps a final latent variable $\widehat{H} \in \mathcal{H}$ to a meaningful physical solution $\widehat{U} \in \mathcal{U}$. This trainable function is designed as an inverse operation to the encoder. Indeed, the decoder projects back the latent space $\mathcal{H}$ into the physical space $\mathcal{U}$ so that we can calculate the various losses used to train the model.

3.4. Stabilization

In Section 3.3, we modelled a GNN-based network with an "infinite" depth by using a black-box solver to find the fixed point of function h_θ, enabling for unrestricted information flow throughout the entire graph. However, such implicit models suffer from two significant downsides: they tend to be unstable during the training phase and are very sensitive to architectural choices, where minimal changes to h_θ can lead to large convergence instabilities. The stability of the model around the fixed point $\widehat{H}$ is determined by the spectral radius ρ of the

Jacobian $J_{h_\theta}(\widehat{H})$. Following Bai et al. (2021), who add a constraint on ρ, we add a penalization term (17) in the loss function described in 3.5. However, since computing the spectral radius is far too computationally costly, and because the Frobenius norm of the Jacobian is an upper bound for its spectral radius, we adopt the method outlined in Bai et al. (2021), which estimates this Frobenius norm using the Hutchinson estimator (Hutchinson, 1990). By doing so, we build a model that satisfies, post-training, $\rho < 1$: the function h_θ becomes contractive, thus ensuring the global asymptotic convergence of the model. As a result, during inference, one can simply iterate on the Processor (in an RNN-like manner) until convergence (i.e. loop over the h_θ function). More importantly, the model could work with any kind of root-finding solver. Overall, this approach offers strong convergence guarantees and addresses the stability issues commonly encountered in implicit models.

3.5. Training

Loss function The entire Ψ-GNN model is trained by minimizing the following cost function:

$$\begin{aligned}
\mathcal{L} \; &= \mathcal{L}_{\text{res}}(\widehat{U}, G) && (15)\\
&+ \lambda \times MSE\left(\widehat{U} - U^{\text{ex}}\right) && (16)\\
&+ \beta \times ||J_{h_\theta}(\widehat{H})||_{\text{F}} && (17)\\
&+ \text{MSE}\left(E_\theta(\widehat{U}) - \widehat{H}\right) && (18)\\
&+ \text{MSE}\left(D_\theta(E(\widehat{U})) - \widehat{U}\right) && (19)
\end{aligned}$$

Line (15) represents the residual loss described in (5), and line (16) is an additional supervised loss (U^{ex} being the LU ground truth and λ a small weight, see Section 5). Line (17) is the regularizing term defined in 3.4. Lines (18) and (19) are designed to learn the autoencoding mechanism together: Line (18) aims to properly encode a solution while Line (19) steers the decoder to be the inverse of the encoder. To handle the minimization of the overall structure, a single optimizer is used with two different learning rates, one for the autoencoding process and one for the Message Passing process. This ensures that the autoencoding process is solely used for the purpose of bridging the physical and latent spaces, with no direct impact on the accuracy of the computed solution.

Backpropagation The training of the proposed model has been found to be computationally intensive or even not feasible when backpropagating through all the operations of the fixed-point solver. However, using the approach outlined in Theorem 1 in Bai et al. (2019) significantly enhances the training process by differentiating directly at the fixed point, thanks to the implicit function theorem. This methodology requires the resolution of two fixed-point problems, one during the inference phase and the other during the backpropagation phase. In contrast to traditional methods, this approach removes the need to

open the black-box, and only requires constant memory. Supplementary materials concerning the regularization, the training process and the architecture and implementation of Ψ-GNN can be found in Appendix B and Appendix C.

4. Theoretical properties

This section investigates several theoretical properties of the proposed approach. Specifically, we demonstrate that Ψ-GNN satisfies a property of universal approximation, i.e. the model is able to approximate the optimal solution to the considered statistical problem (6) up to any arbitrary precision, provided the network is large enough. Following the approach and notations outlined in Section 3.2, a discretized Poisson problem $E_h = (\Omega_h,\ A,\ B)$ can be seen as a graph problem $G = (N,\ A,\ B)$, where A and B are respectively the interaction and individual terms on the N nodes of the graph G. The original problem at hand searches for an optimal solution $U_G^\star \in \mathbb{R}^N$ as follows:

$$U_G^\star = \underset{U \in \mathbb{R}^N}{\operatorname{argmin}}\ \mathcal{L}_{\text{res}}\left(U,\ G\right) \tag{20}$$

However, instead of searching directly for $U_G^\star$, we seek for a function $h_G^\star : \mathbb{R}^N \to \mathbb{R}^N$ whose fixed point is $U_G^\star$:

$$h_G^\star = \underset{h:\ \mathbb{R}^N \to\ \mathbb{R}^N}{\operatorname{argmin}}\ \mathcal{L}_{\text{res}}\left(\text{FixedPoint}(h),\ G\right) \tag{21}$$

The first step in proving the consistency of the approach is to determine whether problems (20) and (21) are equivalent, i.e., if solving (21) yields the solution to the original problem (20).

Proposition 1 (Equivalence of direct and fixed-point formulations)**.** *Problems* (20) *and* (21) *are equivalent, i.e., for any problem G, any solution $U_G^\star$ of* (20) *can be turned into a solution $h_G^\star$ of* (21) *and vice versa.*

Proof. See 1 in Appendix D. □

The next step investigates whether the Ψ-GNN architecture is able to find an approximation of $h_G^\star$. To address this, we begin by proving that problem (20) satisfies the hypotheses of Corollary 1 of DSS - Donon et al. (2020).

Proposition 2 (Satisfying the hypotheses of Corollary 1 in DSS)**.** *Problem* (20)*, which can be rewritten as searching for the function:*

$$\begin{array}{rccl} & \mathcal{S} & \to & \mathbb{R}^N \\ \varphi: & G = (N,\ A,\ B) & \mapsto & U^\star(G) := \underset{U}{\operatorname{argmin}}\ \mathcal{L}_{res}(U,\ G) \end{array}$$

satisfies the hypothesis of Corollary 1 *in Donon et al. (2020).*

Proof. See 2 in Appendix D. □

Let $\mathcal{H}$ be the space of functions that can be realized by Graph Neural Networks, as defined in Donon et al. (2020). The previously mentioned Corollary 1 in Donon et al. (2020) establishes the existence of a network with a GNN-based architecture, denoted as $\widehat{\varphi} \in \mathcal{H}$, that can approximate $\varphi : G \mapsto U_G^\star$ with arbitrary precision for any given graph G from the considered problem distribution. We have the following corollary:

Corollary 1 (Existence of a GNN model approximating φ)**.**
For any $\varepsilon > 0$, there exists a GNN-based model $\widehat{\varphi} \in \mathcal{H}$ such that for any problem G from our problem distribution $\mathcal{D}$:

$$\|\widehat{\varphi}(G) - \varphi(G)\| \leqslant \varepsilon$$

Based on Proposition 1 and its proof, this result can be extended to approximate the solution of (21), yielding the following universal approximation property for our Ψ-GNN method:

Theorem 1 (Universal Approximation Property)**.**
For any precision $\varepsilon > 0$, there exists a parameterization θ_ε of a Ψ-GNN architecture with sufficiently large layers, namely, a function $\Psi\text{-GNN}_{\theta_\varepsilon} : \mathcal{S} \to \mathbb{R}^N$, which, for any problem G (i.e., for any mesh, boundary conditions and force terms), approximates the optimal solution of problem (20) *with precision less than ε:*

$$\forall\ G, \quad \|\Psi\text{-GNN}_{\theta_\varepsilon}(G) - \varphi(G)\| \leqslant \epsilon$$

Proof. See 4 in Appendix D. □

Lastly, the set of functions h that admit a unique fixed point close to $\widehat{\varphi}_{\text{GNN}}(G)$ (see 4) can be big (although all these solutions lead approximately to the same fixed point). It is possible to select an optimal solution within this space that is also contractive. This can be achieved in a Lagrangian spirit instead of a supplementary constraint by introducing a penalty term to the loss function, as shown in (17). However, it is important to note that the resulting function can only be assumed to be predominantly contractive with respect to H:

Proposition 3 (Contractivity of $h_{\theta\varepsilon}$)**.**
For any precision $\varepsilon > 0$, the function $h_{\theta_\varepsilon}(H, G)$ in the Processor of the Ψ-GNN architecture obtained by Theorem 1 can be assumed to be contractive with respect to H for any pair of points farther than $\sqrt{\varepsilon}$.

Proof. See 5 in Appendix D. □

5. Experiments, Results, and Discussion

This section presents an in-depth evaluation of Ψ-GNN based on experiments on synthetic data. The first Section 5.1 presents an overview of the synthetic

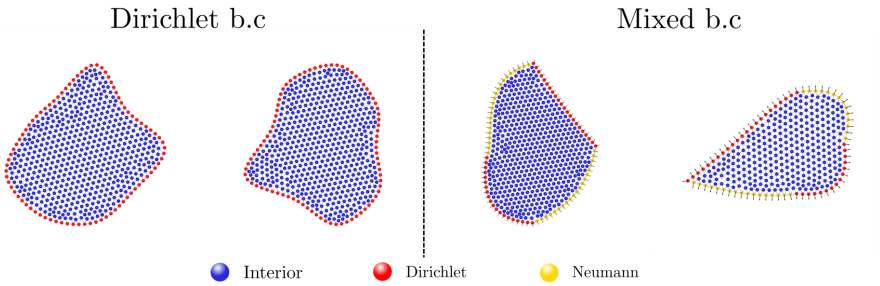
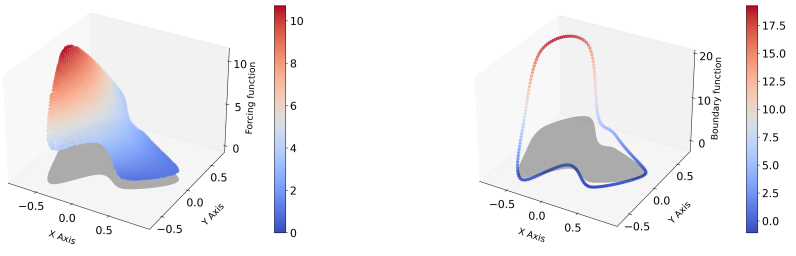


Figure 4: Geometries extracted from the synthetic dataset colored with respect to their node type (blue: Interior, red: Dirichlet, yellow: Neumann). These geometries are used to solve Poisson problems with Dirichlet boundary conditions only (on the left side) and mixed boundary conditions (on the right side). The arrows on the boundary of the two domains on the right represent the normal vectors.

dataset used in these experiments and introduces the standard experimental setup. The performance evaluation of the proposed approach is then performed in three other sections: 5.2 is focused on addressing Poisson problems with Dirichlet boundary conditions only, enabling a direct comparison with the state-of-the-art Deep Statistical Solvers (DSS) method, while 5.3 is focused on Poisson problems with mixed boundary conditions. Furthermore, Section 5.4 proposes various generalization tests to conduct a thorough assessment of Ψ-GNN performance, showcasing its originality. Finally, Section 5.5 investigates the inference complexity of the proposed model.

5.1. Dataset & Experimental setup

Synthetic dataset The dataset used in all experiments consists of 6000 / 2000 / 2000 training/validation/test samples of Poisson problems (1) generated as follows. Random 2D domains Ω are generated using 10 points, randomly sampled in the unit sphere. These points are then connected using Bezier curves to form the boundary of the domain Ω_h. The "MeshAdapt" mesher from GMSH (Geuzaine and Remacle, 2009) is used to discretize Ω into an unstructured triangular mesh Ω_h. Our analysis includes several datasets to assess the performance of the model, which differ regarding the type of boundary conditions. For each dataset, the meshes have approximately 500 nodes (the automatic mesh generator does not include precise control of the number of nodes N). When considering Poisson problems with Dirichlet boundary conditions only, Dirichlet boundary nodes are applied to the entire boundary of the meshes in the dataset (left part in Figure 4). When considering Poisson problems with mixed boundary conditions (1), the boundary is randomly divided into four sections and Dirichlet boundary conditions are applied on two opposite sections, while Neumann boundary conditions are imposed on the remaining opposite sections (right part of Figure 4). Forcing functions f and boundary functions g from



(a) Forcing function

(b) Boundary function

Figure 5: Figures 5a and 5b display the discrete values of a forcing term f and a boundary function g, on a mesh sampled from the test set. The coefficients of f and g, uniformly sampled in $[-10, 10]$, are $r_1 = 3.2$, $r_2 = -7.5$, $r_3 = 1.1$, $r_4 = 5.7$, $r_5 = -9.5$, $r_6 = 0.47$, $r_7 = -8.8$, $r_8 = 9.11$, and $r_9 = 3.5$. The considered mesh is shown as a grey shadow in the plot.

(1) are defined as random quadratic polynomials with coefficients sampled from uniform distributions such as:

$$f(x, y) = r_1(x - 1)^2 + r_2 y^2 + r_3 \qquad (x, y) \in \Omega \tag{22}$$

$$g(x, y) = r_4 x^2 + r_5 y^2 + r_6 xy + r_7 x + r_8 y + r_9 \qquad (x, y) \in \partial\Omega \tag{23}$$

where $r_{i \in [1,\cdots,9]}$ are randomly sampled in $[-10, 10]$. Figure 5 illustrates examples of such f and g functions on a sampled mesh.

Metrics Throughout these experiments, we consider the solution of the discretized Poisson problem given by the classical LU decomposition method as the "ground truth". The reported metrics are the residual loss (5) and the mean squared error (MSE) between the output of the model and the LU solution.

Common setup Ψ-GNN is implemented in Pytorch using the Pytorch-Geometric library (Fey and Lenssen, 2019) to handle graph data. The dimension d of the latent space $\mathcal{H}$ is set to 10. Each neural network block in the architecture (equations 7 to 11) has one hidden layer of dimension 10 with a ReLU activation function. All model parameters are initialized using Xavier initialization (Glorot and Bengio, 2010). For the training, the provided initial solution is set to zero everywhere except at the Dirichlet nodes, which are assigned to the corresponding discrete value of g. Gradient clipping is employed to prevent exploding gradient issues and set to 10^{-2}. The model requires, at each iteration, the solution of two fixed point problems, one for the forward pass and one for the backward pass, as outlined in Section 3.5. These problems are solved using Broyden's method with a relative error as the stopping criteria. The latter is set to 10^{-5} with a maximum of 500 iterations for the forward pass and to 10^{-8} with a maximum of 500 iterations for the backward pass.

5.2. Poisson problems with Dirichlet boundary conditions

This section aims to assess the performance of Ψ-GNN for solving Poisson problems with Dirichlet boundary conditions, allowing for direct comparison with the state-of-the-art Deep Statistical Solvers (DSS) model.

Deep Statistical Solvers As outlined in Section 2, our study proposes to evaluate the efficiency of Ψ-GNN by conducting a comparative analysis with the state-of-the-art Deep Statistical Solvers (DSS) model (Donon et al., 2020). The authors of the DSS model have demonstrated that the convergence of their model is reliant upon a number of MPNNs that is directly proportional to the diameter[4] of the meshes considered within the dataset. The architecture of the DSS model can be described as follows: Initially, a latent state initialized to 0 everywhere is assigned to each node within the network. This latent state then undergoes a fixed sequence of MPNNs, illustrated schematically in Figure C.11a. At each layer, a multilayer perceptron (MLP) is employed to decode the latent state, transforming it into a physical state, upon which an intermediate residual loss (5) is calculated. The final training loss is computed as a weighted cumulative sum of all intermediate losses. It is noteworthy that the authors of Donon et al. (2020) have compared their model trained with either a residual (5) or an MSE loss, demonstrating enhanced generalization capabilities when using a residual loss function. Additionally, we recall several drawbacks of the DSS model: i) the fixed number of MPNNs hampers its ability to generalize across various mesh sizes, ii) the weights associated with the MPNNs and the MLP decoder differ at each layer, resulting in a growing model if the number of iterations needed for convergence increases, iii) it exclusively addresses Poisson problems with Dirichlet boundary conditions, and iv) no explicit treatment of boundary conditions is considered in the model architecture. With the introduction of Ψ-GNN, we propose solutions to address these challenges.

Experimental setup In this experiment, both Ψ-GNN and DSS are trained and tested using a dataset of Poisson problems with Dirichlet boundary conditions only, generated following the process described in Section 5.1. Note that this dataset is similar to the one used in the original DSS study, allowing for fair comparison. Ψ-GNN follows the architectural specifications outlined in Section 3.3, with the exception of excluding functions related to Neumann nodes. During training, the model is optimized using the loss function 15, with $\lambda = 0$ (no additional "ground-truth" MSE minimization) and $\beta = 1$. Training is done using Nvidia P100 GPUs and the Adam optimizer with its default Pytorch hyperparameters, except for the initial learning rate, which was set to 0.05 for the autoencoding process and 0.01 for the main process, as discussed in Section 3.5. The *ReduceLROnPlateau* scheduler from Pytorch is used to progressively reduce the learning rate from a factor of 0.5 during the process, enhancing the training. The remaining hyperparameters adhere to the common setup detailed

[4]The shortest path (node-wise) between the two farthest nodes in a mesh.

	Residual (10^{-3})	MSE w/LU (10^{-2})	Nb of weights
Ψ-GNN	2.69 ± 0.4	$\mathbf{0.85 \pm 2}$	**1444**
DSS	$\mathbf{0.23 \pm 0.2}$	3.0 ± 2	36930

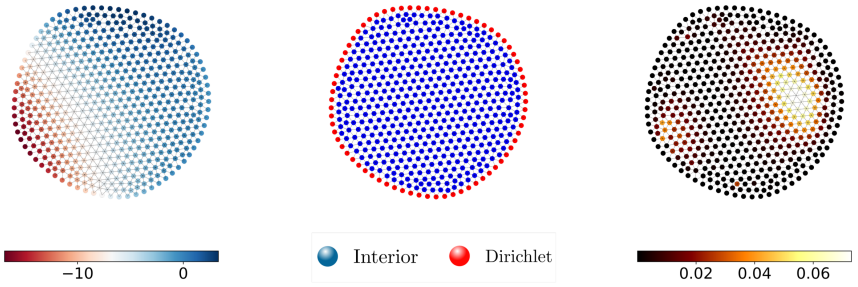
Table 1: Results of Ψ-GNN and DSS averaged over the whole test set

in Section 5.1. Regarding the DSS model, it remains consistent with the original study, i.e. trained with 30 layers.

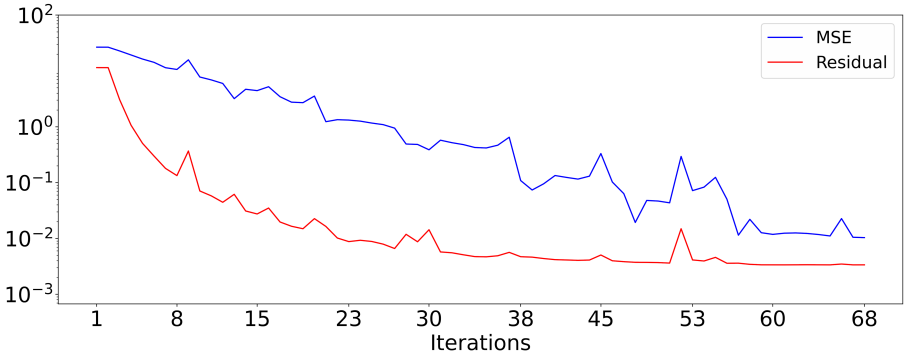
Results Analysis

Table 1 presents the residual and MSE w/LU averaged over the entire test set for Ψ-GNN and DSS. From this table, it is possible to observe that DSS provides slightly better results than Ψ-GNN in terms of residual. This is because, in Ψ-GNN, the training loss not only minimizes the residual but also integrates several other components within the training procedure, such as the stabilization process, which imposes additional constraints on the model weights. Additionally, in DSS, the training loss is computed as a weighted sum of intermediate residual losses, calculated at each iteration of the model, while in Ψ-GNN, the residual loss is computed only at the last layer due to the implicit nature of the iterations. However, the results take a different flavor when looking at the MSE w/LU. There, we can see that Ψ-GNN outperforms DSS. This can be explained by the ability of the model to automatically adjust its number of Message-Passing steps, as well as the autoencoding mechanism that better encodes and decodes Dirichlet boundary conditions up to a precision of order 10^{-6}. This enhanced feature allows for better information flow since the initial latent state is properly initialized and results in a better MSE w/LU. Additionally, Ψ-GNN is able to outperform DSS with only 1444 parameters, in contrast to the DSS model, which requires 36930 iterations (performing only 30 iterations!). Finally, Ψ-GNN is trained on the same dataset as DSS, which contains meshes of approximately 500 nodes. However, Ψ-GNN, even though it is trained on fixed-size meshes, has the remarkable capability of adjusting its iteration count by itself to handle meshes of varying sizes, a feature analyzed in detail in Section 5.4.

Figure 6 illustrates the resolution with Ψ-GNN of a test instance, a Poisson problem with 527 nodes. The figure presents the evolution of the Residual (in the loss function) and the MSE w/LU (though it was not used during training) along the 68 iterations of the Broyden algorithm. At convergence, the Residual reaches a value of 3.36e-3 and an MSE w/LU of 1.03e-2. Similar results on all test examples validate the Ψ-GNN approach, showcasing its ability to solve Poisson problems with Dirichlet boundary conditions.



(a) Data-driven solution (b) Node types (c) Error map



(d) Residual (red) and MSE w/LU (blue) losses

Figure 6: Illustration of the resolution of a Poisson problem extracted from the test set using Ψ-GNN. Figure 6a shows the data-driven solution obtained at the last iteration, while Figure 6c displays the map of squared errors between the data-driven solution and the LU solution. Figure 6b illustrates the different types of nodes. At the bottom, Figure 6d depicts the evolution of the Residual (in red) and MSE w/LU (in blue) across the 68 iterations of the model.

5.3. Poisson problems with mixed boundary conditions

This section aims to assess the performance of Ψ-GNN when extended to solve Poisson problems with mixed boundary conditions.

Experimental setup In this experiment, Ψ-GNN is trained and tested using a dataset of Poisson problems with mixed boundary conditions, generated following the process described in Section 5.1. Similar to the previous test case, training is performed using Nvidia P100 GPUs and the Adam optimizer with its default PyTorch hyperparameters. However, the initial learning rates differ: 0.01 is used for the autoencoding process, while 0.005 is used for the main process. To enhance the training, we employ the PyTorch *ReduceLROnPlateau*

Metrics	Residuals (10^{-3})	MSE w/LU	Nb of weights
Ψ-GNN	3.16 ± 0.2	0.15 ± 0.04	2175

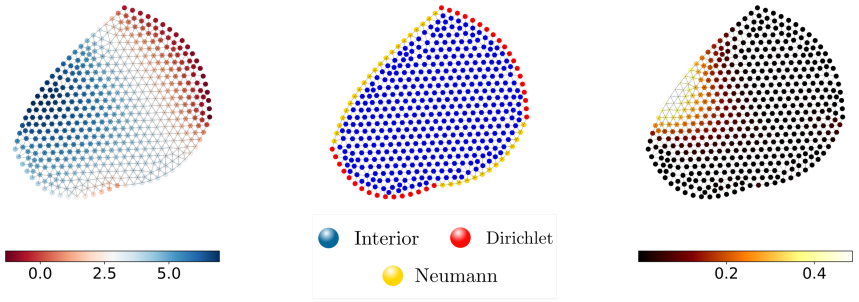
Table 2: Results of Ψ-GNN averaged over the whole test set.

scheduler, gradually reducing the learning rate by a factor of 0.8 during the process. The full Ψ-GNN architecture, as described in 3.3, is trained using the loss function 5 with $\lambda = 0.001$ and $\beta = 1$. The remaining hyperparameters are set according to the common framework described in 5.1.

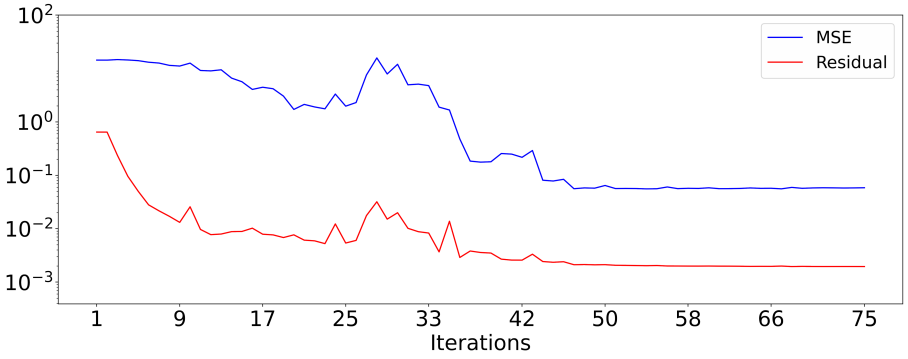
Results Analysis

Table 2 presents the averaged Residual and MSE w/LU errors obtained by Ψ-GNN on the entire test set. These results are comparable in terms of Residual and slightly higher in terms of MSE w/LU compared to those obtained in Section 5.2. This disparity can be attributed to the increased difficulty of learning problems with Neumann boundary conditions, which have higher conditioning. As a result, even with the same residual error, the MSE w/LU can vary due to the challenging conditioning of the mixed boundary condition problem. However, these results demonstrate the accuracy of Ψ-GNN in solving Poisson problems with mixed boundary conditions. Notably, the model has 2175 parameters, which is slightly more than the one developed in Section 5.2, due to the additional networks required to take into account the Neumann boundary conditions.

Figure 7 displays the solution to a specific problem with 470 nodes extracted from the test set. At convergence, the model reaches a Residual of 1.9e-3 and an MSE w/LU of 5.8e-2, showcasing the effectiveness of the method on this test sample. The autoencoding process ensures accurate encoding and decoding of the Dirichlet boundary conditions, thereby preserving them throughout the iterations with an error of magnitude 10^{-6}. The error map reveals that the highest errors are primarily concentrated near the Neumann boundary nodes, aligned with expectations. As discussed in Section 3.2, the graphs considered in this study are directed from the Dirichlet boundary nodes toward the interior of the graph. Consequently, the flow of information is propagated from these nodes towards the inner region of the domain. However, in the case of Neumann nodes, which involve bidirectional edges, the task of propagating information across the entire graph becomes more challenging compared to the fully Dirichlet problem. This is because, in the full Dirichlet problem, information can flow from the entire boundary of the domain. However, in the presence of Neumann nodes, the bidirectional edges complicate the propagation of information throughout the graph, and the Neumann values must be gradually approached. This is in contrast to Dirichlet values, which are considered "exact". As a result, the model requires more iterations to attain the solution.



(a) Data-driven solution (b) Node types (c) Error map



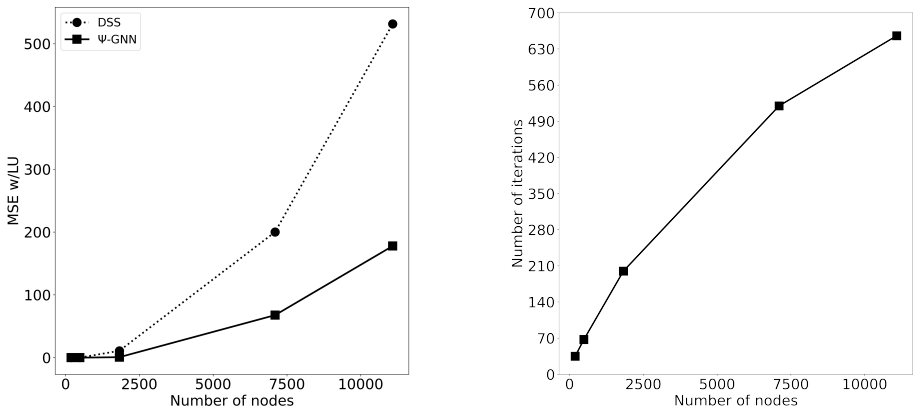
(d) Residual (red) and MSE w/LU (blue) losses

Figure 7: Illustration of the resolution of a Poisson problem extracted from the test set using Ψ-GNN. Figure 7a shows the data-driven solution obtained at the last iteration, while Figure 7c displays the map of squared errors between the data-driven solution and the LU solution. Figure 7b illustrates the different types of nodes. At the bottom, Figure 7d depicts the evolution of the Residual (in red) and MSE w/LU (in blue) across the 75 iterations of the model.

5.4. Sensitivity analyses

This section exhibits several generalizations aspects of Ψ-GNN. First, we demonstrate that Ψ-GNN remains consistent and outperforms DSS when solving problems with an increasing number of nodes in the graph. Then, we show the flexibility and insensitivity of Ψ-GNN with respect to any initial solution. Moreover, we provide evidence of the contractive nature of the constructed GNN function h_θ, as discussed in Section 3.4. Lastly, we address the resolution of an out-of-distribution problem, effectively highlighting the generalization capabilities of Ψ-GNN.

Size of the mesh This paragraph explores the performance and generaliza-



(a) (b)

Figure 8: Figure 8a illustrates the averaged MSE w/LU for different mesh sizes for DSS, and Ψ-GNN. Figure 8b displays the averaged number of iterations performed by the Broyden solver in Ψ-GNN to reach its target threshold with respect to the number of nodes per mesh.

tion capabilities of Ψ-GNN in solving Poisson problems with Dirichlet boundary conditions on meshes with a growing number of nodes, even though it was initially trained on meshes with approximately 500 nodes. To conduct this experiment, we address the resolution of multiple Poisson problems on meshes with varying numbers of nodes. To maintain consistency in the distribution of inputs, we keep the same element sizes, similar to those used to generate the dataset, while varying the number of nodes by increasing the mesh radius. The force and boundary functions are randomly sampled following Section 5.1 but rescaled according to the selected radius. We consider six different setups corresponding to meshes with approximately 200, 500, 2000, 7000, and 11000 nodes per mesh. For each setup, we solve 200 Poisson problems using DSS and Ψ-GNN. The Broyden method used in the Ψ-GNN model is configured with a stopping criterion of 10^{-5} and a maximum of 1000 iterations. The DSS model, as defined in Section 5.2, infers solutions using only 30 iterations. Notably, the Ψ-GNN model adjusts its iteration count thanks to the root-finding procedure, while DSS has a fixed number of iterations, and increasing them would necessitate training a new model. Figure 8a displays the MSE w/LU evolution (averaged over the 200 problems) for each setup and each of the three models. The figure clearly shows that the DSS model diverges for larger meshes. In contrast, the Ψ-GNN model remains consistent when applied to larger meshes. Furthermore, Figure 8b displays the averaged iteration counts of the Broyden solver in the Ψ-GNN model with respect to the number of nodes per mesh. This figure clearly demonstrates that Ψ-GNN can adapt its number of Message-Passing layers to attain a solution.

Initialization This paragraph aims to demonstrate the flexibility of Ψ-GNN and its insensitivity regarding its initialization. By "flexible", we mean that

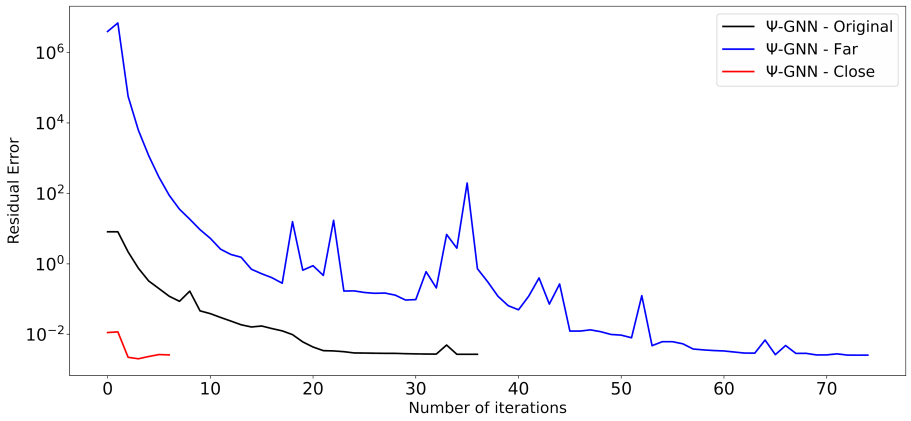


Figure 9: Evolution of the Residual error across iteration of Ψ-GNN for the same Poisson problem but considering three different initial solutions, demonstrating the adaptability of Ψ-GNN with respect to various initial solutions.

Ψ-GNN can dynamically adapt its number of iterations based on the distance from the initial solution to the final solution. And by "insensitive", we refer to the property that, regardless of the initial solution, the model consistently converges to the same fixed point, representing the desired solution. These advantageous characteristics are enabled by the auto-encoding process that maps the initial physical state to the latent space, where GNN layers are applied, and back. Flexibility is demonstrated thanks to Figure 9, which displays the evolution of the residual error across the iterations of the Broyden algorithm for the same Poisson problem considering three different initial conditions: the "Original" initial condition (black curve) is the one used during training, where the initial state is initialized to 0. The "Far" initial condition (blue curve) involves applying a large random noise (uniform noise in the range $[-1000, 1000]$) to the solution obtained from the LU "ground-truth" method. The "Close" condition (red curve) incorporates a small perturbation (uniform noise in the range $[0, 1]$) to the target "ground-truth" solution. For all these initial conditions, the true value of Dirichlet boundary conditions is preserved. This experiment showcases that the number of iterations required for convergence varies depending on the proximity of the initial solution to the final solution. This adaptive behaviour enables the model to adjust its iteration count effectively, optimizing convergence efficiency. Insensitivity to initial solutions is demonstrated in the second column of Table 3, which evaluates the performance of Ψ-GNN using various initial solutions. This experiment demonstrates that Ψ-GNN is robust to the choice of initial solution: regardless of the initial solution provided, the algorithm consistently converges to the desired solution. Specifically, the "Random Initialization" entry in Table 3 reports the metrics averaged over the entire test set, using an initial solution randomly generated by perturbing the ground-truth solution with uniformly random values ranging from -1000 to 1000. Remark-

	Original	*Random Initializer*	*Forward Iteration*
Residual (10^{-3})	3.16 ± 0.2	3.18 ± 0.3	3.14 ± 0.3
MSE w/LU	0.15 ± 0.04	0.16 ± 0.09	0.16 ± 0.04

Table 3: Results averaged over the whole test set for different experiments. First column: original results of Section 5.3; Second column: random initial solution; Third column: replacing Boyden with forward iterations.

ably, the results obtained with random initialization are almost identical to those obtained with the "Original setup" (first column of 3 identical to the results from Section 5.3) indicating that our model is insensitive to the specific choice of the initial solution. In particular, this is a significant improvement compared to the DSS approach, which lacks this capability since it does not have any encoding process.

Contractivity The regularization term (17) serves the purpose of constraining the spectral radius of the Jacobian $J_{h_\theta}(\widehat{H})$ in order to ensure the stability of the model around the fixed point $\widehat{H}$, as discussed in Section 3.4. The results obtained on the test set indicate the effectiveness of this regularization, with an average spectral radius of $\mathbf{0.993 \pm 4e\text{-}4 < 1}$. This value is estimated a posteriori using the Power Iteration method (Golub and Van der Vorst, 2000). Furthermore, we demonstrate this generalization results in the third column of Table 3. In this experiment, we evaluate Ψ-GNN on the entire test set, but instead of using the root-finding Broyden algorithm employed during training, we simply iterate on the Processor described in the architecture in Section 3.3. Once again, the obtained results are almost identical to those obtained in the original setup. This demonstrates the contractive nature of the GNN-function h_θ and highlights the robustness and flexibility of Ψ-GNN in its ability to adapt to different solvers.

Out-of-distribution sample To further illustrate this, we conduct an experiment on a geometry representing a caricatural Formula 1 with 1219 nodes. This geometry includes "holes" (such as a cockpit and front and rear wing stripes) and is larger (1219 nodes) and significantly different than those seen in the training dataset, providing a challenging test of the model's ability to generalize to out-of-distribution examples. We impose Dirichlet boundary conditions on all exterior nodes (pink nodes in the vertical plot at the left of Figure 10) and Neumann conditions on the nodes within the "holes" (yellow nodes). Functions f and g of (1) are randomly sampled from the same distribution as for the training set. The equilibrium of the model is found by simply iterating on the Processor instead of relying on Broyden's algorithm (see previous paragraph). The stopping criteria is the relative error set to 10^{-4}. The results

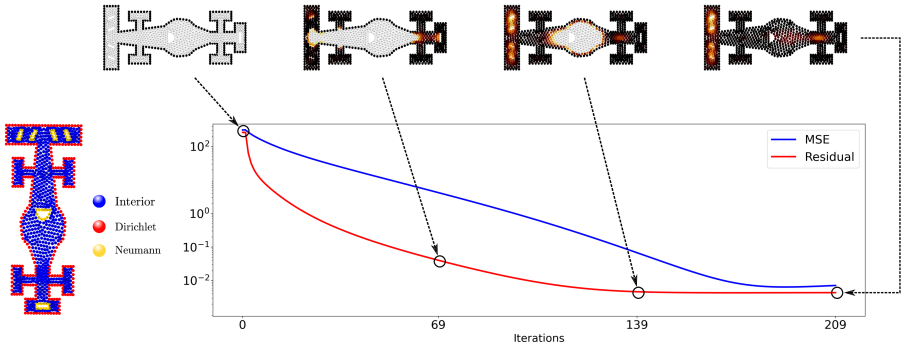


Figure 10: Generalization on the "out-of-distribution" F1 shape, 1219 nodes. **Central plot**: Residual and MSE during the 253 iterations of the Processor (i.e., without using RootFind), demonstrating the contractivity of h_θ. **Left**: The boundary conditions. **Top**: the visual evolution of the squared error, displaying the flow of information from Dirichlet nodes inward.

Figure 10 again shows the contracting nature of the learned function, that converges to the fixed point when iterated. Furthermore, it also gives an example of the generalization capacity of the learned model to some out-of-distribution examples. Additionally, the figure also illustrates how the information propagates through the graph, starting from the Dirichlet nodes to gradually filling the whole domain.

5.5. Inference complexity

To determine the expected performance of Ψ-GNN as the number of nodes N in a graph varies, it is essential to analyze its complexity. In Ψ-GNN, all neural networks have a single hidden layer of dimension d, so the complexity of applying a neural network to one node is $\mathcal{O}\left(d^3\right)$. Let us assume that m is the average number of neighbours for each node in the graph. The complexity of computing the output of a GNN layer relative to one node in a mesh is then of $\mathcal{O}\left(md^3\right)$. Considering all N nodes in the graph and iterating for K updates, the complexity becomes $\mathcal{O}\left(KNmd^3\right)$. This represents the theoretical complexity of Ψ-GNN when the Processor is iterated upon for K iterations.

When using Ψ-GNN with Broyden's algorithm, the complexity is found to be of a higher order. Broyden's method is a quasi-Newton method that computes the next iterate H^{k+1} as follows:

$$H^{k+1} = H^k + J^{-1}_{|M_\theta}(H^k)M_\theta(H^k) \tag{24}$$

In Equation (24), we already know the complexity of $M_\theta(H^k)$, which is computed in a similar manner as previously explained and has a complexity of

$\mathcal{O}\left(Nmd^3\right)$. In his paper (Broyden, 1965), Broyden suggests using the Sherman-Morrison formula (Sherman and Morrison, 1950) to update the inverse of the Jacobian matrix directly, as follows:

$$B^k_{|M_\theta} = B^{k-1}_{|M_\theta} + \frac{\Delta H^k - B^{k-1}_{|M_\theta}\Delta M^k_\theta}{(\Delta H^k)^T B^{k-1}_{|M_\theta}\Delta M^k_\theta}(\Delta H^k)^T B^{k-1}_{|M_\theta} \tag{25}$$

Here, $B^k_{|M_\theta} = J^{-1}_{|M_\theta}(H^k)$, $\Delta H^k = H^k - H^{k-1}$, and $\Delta M^k_\theta = M_\theta(H^k) - M_\theta(H^{k-1})$. In Equation (25), all operations are matrix-vector products of size N, so the complexity of updating the Jacobian matrix is $\mathcal{O}\left(N^2\right)$. Back to equation (24), the total complexity to compute the next iterate is then of $\mathcal{O}\left(N^2\right) + \mathcal{O}\left(Nmd^3\right)$, which corresponds to the cost of updating the Jacobian matrix and the cost of evaluating the GNN model. This is applied for M iterations of the Broyden algorithm, resulting in a global complexity of $\mathcal{O}\left(MN^2\right)$, since the quadratic complexity dominates the linear one.

6. Conclusions and Future Work

This paper has introduced Ψ-GNN, a novel and consistent Machine Learning based approach that combines Graph Neural Networks and Implicit Layer Theory to effectively solve a wide range of Poisson problems. The model, trained in a "physics-informed" manner, is found to be robust, stable, and highly adaptable to varying mesh sizes, domain shapes, boundary conditions, and initialization. To the best of our knowledge, this approach is distinct from any previous Machine Learning based methods. Furthermore, Ψ-GNN can be extended to other steady-state partial differential equations and its application to 3-dimensional domains is straightforward. Overall, the results of this study demonstrate the significant potential of Ψ-GNN in solving Poisson problems in a consistent and efficient manner.

The long-term objective of the program driving this work is to accelerate industrial Computation Fluid Dynamics (CFD) codes on software platforms such as OpenFOAM Chen et al. (2014). In future work, we aim to evaluate the efficiency of the proposed approach by implementing it in an industrial CFD code, and assessing its impact on performance acceleration. However, despite its flexibility, the proposed method faces limitations in terms of graph size. Scaling up the model to handle industrial-scale cases poses a significant challenge that Ψ-GNN alone, in its current state, cannot overcome. This challenge arises due to the increasing number of iterations required and the limited expressiveness of GNN models when applied to large graphs. Nonetheless, there are promising directions to scale up the model. One approach is to employ it as a solver for domain decomposition algorithms, leveraging the batch parallel framework of Machine Learning models to solve multiple sub-problems simultaneously (Nastorg et al., 2024). Another potential direction for future research is the exploration of a fully hierarchical architecture.

Acknowledgment

This research was supported by DATAIA Convergence Institute as part of the "Programme d'Investissement d'Avenir", (ANR- 17-CONV-0003) operated by INRIA and IFPEN.

During the preparation of this work, the authors used OpenAI ChatGPT in order to rectify grammatical errors. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

Alet, F., Jeewajee, A.K., Villalonga, M.B., Rodriguez, A., Lozano-Perez, T., Kaelbling, L., 2019. Graph element networks: adaptive, structured computation and memory, in: International Conference on Machine Learning, PMLR. pp. 212–222.

Ba, J.L., Kiros, J.R., Hinton, G.E., 2016. Layer normalization. arXiv preprint arXiv:1607.06450 .

Bai, S., Kolter, J.Z., Koltun, V., 2019. Deep equilibrium models. Advances in Neural Information Processing Systems 32.

Bai, S., Koltun, V., Kolter, J.Z., 2021. Stabilizing equilibrium models by jacobian regularization. arXiv preprint arXiv:2106.14342 .

Battaglia, P., Pascanu, R., Lai, M., Jimenez Rezende, D., et al., 2016. Interaction networks for learning about objects, relations and physics. Advances in neural information processing systems 29.

Baymani, M., Kerayechian, A., Effati, S., 2010. Artificial neural networks approach for solving stokes problem. Applied Mathematics 1, 288.

Briggs, W.L., Henson, V.E., McCormick, S.F., 2000. A multigrid tutorial. SIAM.

Broyden, C.G., 1965. A class of methods for solving nonlinear simultaneous equations. Mathematics of computation 19, 577–593.

Cai, S., Mao, Z., Wang, Z., Yin, M., Karniadakis, G.E., 2022. Physics-informed neural networks (pinns) for fluid mechanics: A review. Acta Mechanica Sinica , 1–12.

Chang, M.B., Ullman, T., Torralba, A., Tenenbaum, J.B., 2016. A compositional object-based approach to learning physical dynamics. arXiv preprint arXiv:1612.00341 .

Chen, G., Xiong, Q., Morris, P.J., Paterson, E.G., Sergeev, A., Wang, Y., 2014. Openfoam for computational fluid dynamics. Notices of the AMS 61, 354–363.

Chen, R., Jin, X., Li, H., 2022. A machine learning based solver for pressure poisson equations. Theoretical and Applied Mechanics Letters 12, 100362.

Cheng, L., Illarramendi, E.A., Bogopolsky, G., Bauerheim, M., Cuenot, B., 2021. Using neural networks to solve the 2d poisson equation for electric field computation in plasma fluid simulations. arXiv preprint arXiv:2109.13076 .

Dissanayake, M., Phan-Thien, N., 1994. Neural-network-based approximations for solving partial differential equations. communications in Numerical Methods in Engineering 10, 195–201.

Donon, B., Liu, Z., Liu, W., Guyon, I., Marot, A., Schoenauer, M., 2020. Deep statistical solvers. Advances in Neural Information Processing Systems 33, 7910–7921.

Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428 .

Gao, H., Zahr, M.J., Wang, J.X., 2022. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. Computer Methods in Applied Mechanics and Engineering 390, 114502.

Geuzaine, C., Remacle, J.F., 2009. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. International journal for numerical methods in engineering 79, 1309–1331.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings. pp. 249–256.

Golub, G.H., Van der Vorst, H.A., 2000. Eigenvalue computation in the 20th century. Journal of Computational and Applied Mathematics 123, 35–65.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al., 2018. Recent advances in convolutional neural networks. Pattern recognition 77, 354–377.

Guermond, J.L., Quartapelle, L., 1998. On stability and convergence of projection methods based on pressure poisson equation. International Journal for Numerical Methods in Fluids 26, 1039–1053.

Guo, X., Li, W., Iorio, F., 2016. Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 481–490.

Hamilton, W., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. Advances in neural information processing systems 30.

Horie, M., Mitsume, N., 2022. Physics-embedded neural networks: E (n)-equivariant graph neural pde solvers. arXiv preprint arXiv:2205.11912 .

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

Hutchinson, M.F., 1990. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. Communications in Statistics-Simulation and Computation 19, 433–450.

Illarramendi, E.A., Bauerheim, M., Cuenot, B., 2021. Performance and accuracy assessments of an incompressible fluid solver coupled with a deep convolutional neural network. arXiv preprint arXiv:2109.09363 .

Jin, X., Cai, S., Li, H., Karniadakis, G.E., 2021. Nsfnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations. Journal of Computational Physics 426, 109951.

Kazhdan, M., Bolitho, M., Hoppe, H., 2006. Poisson surface reconstruction, in: Proceedings of the fourth Eurographics symposium on Geometry processing.

Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 .

Kochkov, D., Smith, J.A., Alieva, A., Wang, Q., Brenner, M.P., Hoyer, S., 2021. Machine learning–accelerated computational fluid dynamics. Proceedings of the National Academy of Sciences 118, e2101784118.

Kumar, M., Yadav, N., 2011. Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: a survey. Computers & Mathematics with Applications 62, 3796–3811.

Lagaris, I.E., Likas, A., Fotiadis, D.I., 1998. Artificial neural networks for solving ordinary and partial differential equations. IEEE transactions on neural networks 9, 987–1000.

Langtangen, H.P., Mardal, K.A., 2019. Introduction to Numerical Methods for Variational Problems. doi:10.1007/978-3-030-23788-2.

Larson, M.G., Bengzon, F., 2013. The finite element method: theory, implementation, and applications. volume 10. Springer Science & Business Media.

LeCun, Y., Bengio, Y., et al., 1995. Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks 3361, 1995.

Lee, H., Kang, I.S., 1990. Neural algorithm for solving differential equations. Journal of Computational Physics 91, 110–131.

Lee, J., Lee, I., Kang, J., 2019. Self-attention graph pooling, in: International conference on machine learning, PMLR. pp. 3734–3743.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., Anandkumar, A., 2020a. Neural operator: Graph kernel network for partial differential equations. arXiv preprint arXiv:2003.03485 .

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., Anandkumar, A., 2020b. Multipole graph neural operator for parametric partial differential equations. Advances in Neural Information Processing Systems 33, 6755–6766.

Lino, M., Cantwell, C., Bharath, A.A., Fotiadis, S., 2021. Simulating continuum mechanics with multi-scale graph neural networks. arXiv preprint arXiv:2106.04900 .

Luty, B.A., Davis, M.E., McCammon, J.A., 1992. Solving the finite-difference non-linear poisson–boltzmann equation. Journal of computational chemistry 13, 1114–1118.

Mannheim, P.D., Kazanas, D., 1994. Newtonian limit of conformal gravity and the lack of necessity of the second order poisson equation. General Relativity and Gravitation 26, 337–361.

Mao, Z., Jagtap, A.D., Karniadakis, G.E., 2020. Physics-informed neural networks for high-speed flows. Computer Methods in Applied Mechanics and Engineering 360, 112789.

Morton, K., Mayers, D., 2005. Numerical Solution of Partial Differential Equations: An Introduction. Cambridge University Press. URL: https://books.google.fr/books?id=0uf_iQIBFKgC.

Nastorg, M., Gratien, J.M., Faney, T., Bucci, M.A., Charpiat, G., Schoenauer, M., 2024. Multi-level gnn preconditioner for solving large scale problems. arXiv preprint arXiv:2402.08296 .

Nastorg, M., Schoenauer, M., Charpiat, G., Faney, T., Gratien, J.M., Bucci, M.A., 2022. Ds-gps: A deep statistical graph poisson solver (for faster cfd simulations). arXiv preprint arXiv:2211.11763 .

Obiols-Sales, O., Vishnu, A., Malaya, N., Chandramowliswharan, A., 2020. Cfdnet: A deep learning-based accelerator for fluid simulations, in: Proceedings of the 34th ACM international conference on supercomputing, pp. 1–12.

Olver, P., 2013. Introduction to Partial Differential Equations. Undergraduate Texts in Mathematics, Springer International Publishing. URL: https://books.google.fr/books?id=aQ8JAgAAQBAJ.

Özbay, A.G., Hamzehloo, A., Laizet, S., Tzirakis, P., Rizos, G., Schuller, B., 2021. Poisson cnn: Convolutional neural networks for the solution of the poisson equation on a cartesian mesh. Data-Centric Engineering 2.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A., 2017. Automatic differentiation in pytorch .

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., Battaglia, P.W., 2020. Learning mesh-based simulation with graph networks. arXiv preprint arXiv:2010.03409 .

Raissi, M., Karniadakis, G.E., 2018. Hidden physics models: Machine learning of nonlinear partial differential equations. Journal of Computational Physics 357, 125–141.

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. 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.

Rao, C., Sun, H., Liu, Y., 2020. Physics-informed deep learning for incompressible laminar flows. Theoretical and Applied Mechanics Letters 10, 207–212.

Reddy, J.N., 2019. Introduction to the finite element method. McGraw-Hill Education.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer. pp. 234–241.

Saad, Y., 2003. Iterative methods for sparse linear systems. SIAM.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., Battaglia, P., 2020. Learning to simulate complex physics with graph networks, in: International Conference on Machine Learning, PMLR. pp. 8459–8468.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J.T., Merel, J., Riedmiller, M., Hadsell, R., Battaglia, P., 2018. Graph networks as learnable physics engines for inference and control, in: International Conference on Machine Learning, PMLR. pp. 4470–4479.

Sherman, J., Morrison, W.J., 1950. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. The Annals of Mathematical Statistics 21, 124–127.

Smaoui, N., Al-Enezi, S., 2004. Modelling the dynamics of nonlinear partial differential equations using neural networks. Journal of Computational and Applied Mathematics 170, 27–58.

Tang, W., Shan, T., Dang, X., Li, M., Yang, F., Xu, S., Wu, J., 2017. Study on a poisson's equation solver based on deep learning technique, in: 2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS), IEEE. pp. 1–3.

Um, K., Brand, R., Fei, Y.R., Holl, P., Thuerey, N., 2020. Solver-in-the-loop: Learning from differentiable physics to interact with iterative pde-solvers. Advances in Neural Information Processing Systems 33, 6111–6122.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., 2017. Graph attention networks. arXiv preprint arXiv:1710.10903 .

Wiewel, S., Becher, M., Thuerey, N., 2019. Latent space physics: Towards learning the temporal evolution of fluid flow, in: Computer graphics forum, Wiley Online Library. pp. 71–82.

Wu, J.L., Xiao, H., Paterson, E., 2018. Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework. Physical Review Fluids 3, 074602.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y., 2020. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems 32, 4–24.

Yilmaz, E., German, B., 2017. A convolutional neural network approach to training predictors for airfoil performance, in: 18th AIAA/ISSMO multidisciplinary analysis and optimization conference, p. 3660.

Appendix A. Brief introduction to the Finite Element method

This section provides a brief introduction to the fundamental concepts of the Finite Element method (FEM), explaining how to construct the discrete system described in (2) from the considered Poisson problem (1).

Problem (1) consists in solving the following 2d boundary-value problem:

$$\left\{\begin{array}{rcll} -\Delta u & = & f & \in \Omega \\ u & = & g & \in \partial\Omega_D \\ \frac{\partial u}{\partial n} & = & 0 & \in \partial\Omega_N \end{array}\right. \tag{A.1}$$

where $u = u(x, y)$ is the unknown function, $f = f(x, y)$ is the forcing function and $g = g(x, y)$ is the Dirichlet boundary function defined on the 2d domain Ω with boundary $\partial\Omega = \Omega_D \cup \Omega_N$. Besides, n denotes the outward normal vector on $\partial\Omega$ and Δ is the Laplace operator defined as:

$$\Delta u = \frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y}$$

The FEM usually consists of the following steps: i): conversion of the PDE into a variational formulation ii): discretization of the domain Ω into a mesh Ω_h iii): choice of interpolation functions iv): formulation of the discretized system v): resolution of the system.

The variational form of the PDE is obtained by multiplying the PDE by a function v and integrating it over the domain Ω. The function v is known as the test function, while the solution function u is referred to as the trial function. Both test and trial functions belong to some suitable function spaces denoted as V for the trial function space and $\widehat{V}$ for the test function space. Specifically, V requires that $u = g$ on $\partial\Omega_D$ while $\widehat{V}$ requires that v vanishes on $\partial\Omega_D$. Using Green's formula, this setting yields the following variational problem (for more advanced theoretical details, refer to Langtangen and Mardal (2019)):

Find $u \in V$ such that:

$$\int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x = \int_\Omega f v \, \mathrm{d}x \qquad \forall v \in \widehat{V} \tag{A.2}$$

The variational formulation (A.2) poses a continuous problem that defines the solution u in an infinite-dimensional space V. In contrast, the FEM pursues an approximate solution u_h of u by substituting the infinite-dimensional function spaces with finite ones. To that purpose, the continuous domain Ω is first discretized into an unstructured triangular mesh Ω_h with N nodes. Next, let V_h be a finite approximation space constructed using first-order piecewise polynomials defined on each triangle K of Ω_h. Furthermore, let $(\phi_i)_{(1 \leq i \leq N)}$ be a set of basis functions for V_h. Notably, these basis functions are constructed to satisfy the

following property:

$$\phi_i(x_j) = \delta_{i,j} \quad \text{with} \quad \delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

One can consider an extension of u_h on the basis of V_h such that:

$$u_h = \sum_{i=1}^{N} u_i \phi_i \tag{A.3}$$

Taking $v_i = \phi_i$, the variational form A.2 can be discretized such that:

$$\sum_{j=1}^{N} u_j \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, \mathrm{d}x = \int_\Omega f_i \phi_i \, \mathrm{d}x \qquad \forall\ 1 \leq i \leq N \tag{A.4}$$

Introducing the *stiffness* matrix

$$A = (a_{ij})_{(1 \leq i,j \leq N)} \quad \text{with} \quad a_{ij} = \int_\Omega \nabla\phi_j \cdot \nabla\phi_i \, \mathrm{d}x$$

the solution vector $U = (u_i)_{(1 \leq i \leq N)}$ and the right-hand side vector

$$B = (b_i)_{(1 \leq i \leq N)} \quad \text{with} \quad b_i = \int_\Omega f_i \phi_i \ \mathrm{dx}$$

then (A.4) corresponds to a linear system to solve of the form:

$$AU = B \tag{A.5}$$

It is important to note that using this specific discretization approach does not impose any boundary conditions. In the case of homogeneous Neumann boundary conditions, the variational formulation A.2 suggests that these conditions should not be considered during the discretization process, which is consistent with the previous configuration. However, the treatment of Dirichlet boundary conditions is not handled automatically. Instead, in practice, the Dirichlet boundary conditions are incorporated by modifying the linear system (A.5) manually: for the indices corresponding to Dirichlet boundary conditions, the corresponding row of matrix A is set to 0, and a 1 is placed on the diagonal. Additionally, the value in vector B at the corresponding index is changed to match the discrete value of g.

To further illustrate this, let's consider the resolution of the following 1-d Poisson problem with mixed boundary conditions, defined on $\Omega = (0, 1)$:

$$\begin{cases} -u''(x) &= f(x) \qquad x \in (0,1) \\ u(0) &= g_0 \\ u'(1) &= 0 \end{cases} \tag{A.6}$$

In this example, we assume that Ω is divided into 4 cells with equal lengths, denoted as $h = \frac{1}{4}$. Using the previously described approach, we arrive at the following linear system A.5:

$$\underbrace{\frac{1}{h}\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}}_{U} = \underbrace{\begin{bmatrix} hf_0 \\ hf_1 \\ hf_2 \\ hf_3 \\ hf_4 \end{bmatrix}}_{B} \tag{A.7}$$

As mentioned previously, solving A.7 does not fulfil Dirichlet boundary conditions (i.e. $u_0 \neq g_0$). Manually changing the matrix to impose these conditions yields the following system:

$$\underbrace{\frac{1}{h}\begin{bmatrix} h & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}}_{U} = \underbrace{\begin{bmatrix} g_0 \\ hf_1 \\ hf_2 \\ hf_3 \\ hf_4 \end{bmatrix}}_{B}$$

Considering the sparse matrix A as an adjacency matrix for its corresponding mesh, it is possible to analyse the structure of the induced graph. The first row of A implies $u_0 = g_0$, which indicates that u_0 remains independent of information from other nodes. However, the second row of A results in:

$$u_1 = \frac{h}{2}\left(\frac{1}{h}u_0 + \frac{1}{h}u_2 + hf_1\right) \tag{A.8}$$

(A.8) implies that computing u_1 requires information from both u_0 and u_2, resulting in an undirected graph for this particular node. By applying the same analysis, one can deduce that matrix A represents a directed graph at Dirichlet nodes, where information is only sent, and an undirected graph for Interior (rows 2 to 4) and Neumann (row 5) nodes, where information is both sent and received.

Appendix B. Supplementary materials for training

This Appendix provides additional training materials detailing the architecture of some trainable functions and explaining the format of vectors considered in 3.3. It also informs of how the normalization is performed on the data.

Construction of $\Lambda_{i,\theta}$ Equation (9) details the process of updating an Interior node. As a reminder, the updated Interior latent state $z_i^{\mathtt{I}}$ is computed such that:

$$z_i^{\mathtt{I}} = H_i + \Lambda_{i,\theta} \tag{B.1}$$

To compute $\Lambda_{i,\theta}$, we use two MLPs, Ψ_θ^1 and Ψ_θ^2, which both take the same inputs. These inputs include the current latent state H_i, computed MPNNs (8) and (7), and problem-specific data b_i (refer to the details below). The MLPs compute α_i and ζ_i, respectively. The final $\Lambda_{i,\theta}$ is obtained by multiplying α_i and ζ_i as follows:

$$\alpha_i = \Psi_\theta^1(H_i, b_i, \phi^{\mathtt{I}}_{\rightarrow,i}, \phi^{\mathtt{I}}_{\leftarrow,i}) \tag{B.2}$$

$$\zeta_i = \Psi_\theta^2(H_i, b_i, \phi^{\mathtt{I}}_{\rightarrow,i}, \phi^{\mathtt{I}}_{\leftarrow,i}) \tag{B.3}$$

$$\Lambda_{i,\theta} = \alpha_i \times \zeta_i \tag{B.4}$$

The key distinction lies in the construction of Ψ_θ^1 and Ψ_θ^2. Although they share the same architecture as described in the "General experimental setup" from Section 5.1 (i.e. one hidden layer with dimension 10), Ψ_θ^1 uses a Sigmoid activation function, while Ψ_θ^2 uses a ReLU activation function. As a result, α_i is restricted to $(0, 1)$, preventing the updating process (B.1) from exploding. Furthermore, α_i can be interpreted as an attention layer, prioritizing the most crucial connections to enhance the performance of our model.

Data information The architecture of the model described in 3.3 relies on several inputs whose format needs to be precise. B.2, B.3 and 11, require as input problem-related data, encoded into a vector b_i such that:

$$b_i = \begin{cases} \begin{bmatrix} f_i & 0 & 0 \end{bmatrix} & i \text{ is Interior} \\ \begin{bmatrix} 0 & g_i & 0 \end{bmatrix} & i \text{ is Dirichlet} \\ \begin{bmatrix} 0 & 0 & f_i \end{bmatrix} & i \text{ is Neumann} \end{cases} \tag{B.5}$$

where f_i and g_i are the discretized values of the volumetric function f and the Dirichlet boundary function g from the Poisson problem (1).

Data normalization In such problems, normalizing the input features is essential to enhance the model's performance during the training phase. To achieve this, the distances d_{ij} in equations (8), (7), (10), the problem-related vector b_i in equations (B.2), (B.3), and (11), as well as the normal vector n_i in equation (11), are all normalized. This normalization is accomplished by subtracting the mean and dividing by the standard deviation, which are computed

based on the entire dataset. When dealing with problem-related data, the normalization is conducted column-wise, taking into account the statistics of the entire dataset. This approach is necessary as f_i and g_i may represent a multimodal distribution, rendering it challenging to achieve proper normalization if treated as a whole. **Consequently, all inputs used for solution inference are normalized.**

Appendix C. Implicit Models

This section delves into the Implicit Layer Theory, providing supplementary information on its implementation. The motivations for its use are outlined in Appendix C.1, followed by a detailed description of the training procedures in Appendix C.2, emphasising backpropagation and exploiting Theorem 1 in Bai et al. (2019). Furthermore, a comprehensive analysis of the Hutchinson method (Hutchinson, 1990) used to calculate the regularizing term as defined equation (17) is presented in Appendix C.3.

Appendix C.1. Motivations

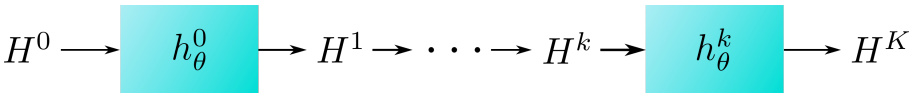
Many architectures are available for training Graph Neural Network (GNN) models, as demonstrated in Wu et al. (2020). In the context of this work, where we aim to solve a Poisson problem on unstructured meshes by directly minimizing the residual of the discretized equation, the number of layers required to achieve convergence should be proportional to the diameters of the meshes under consideration. Figure C.11a and C.11b illustrate the two common GNN architectures traditionally used for this purpose. In Figure C.11a, the final solution H^K is obtained after iterating a fixed number of times K on different Message Passing layers h^i_θ, each with distinct weights. This architecture is employed in approaches such as in Donon et al. (2020). In Figure C.11b, the architecture is of recurrent type, where the final solution is obtained by also iterating a fixed number of times K but on the same neural network h_θ. This architecture has the advantage of significantly reducing the size of the model, as employed in the work of Nastorg et al. (2022). In both methods, the number of iterations K (i.e. the number of Message Passing steps) is fixed, limiting the model's ability to generalize to different mesh sizes. **However, using the recurrent architecture, it has been experimentally demonstrated that the model, trained with a sufficient number of iterations, tends towards a fixed point.**

Building upon these previous experimental results, we propose to enhance the recurrent architecture by incorporating a black-box root-finding solver to directly find the equilibrium point of the model, as illustrated in figure C.11c. In this approach, the iterations (i.e. the flow of information) are performed implicitly within the RootFind solver. This method eliminates the need for a fixed number of iterations, as the number of required Message Passing steps is now exclusively determined by the precision imposed on the solver, resulting in a more adaptable and flexible approach.

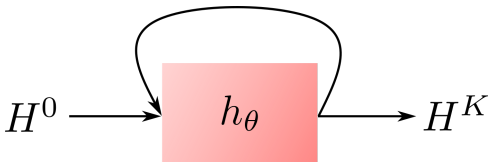
Appendix C.2. Training an Implicit Model

Following Bai et al. (2019), the training of an implicit model requires, at each iteration, the resolution of two fixed point problems: one for the forward pass and one for the backward pass.

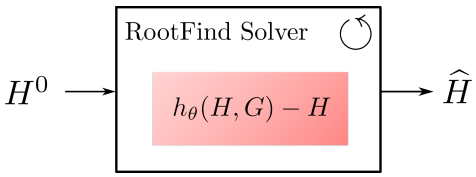
Forward pass The resolution of a fixed point for the forward pass is clear



(a) Stacking GNN layers : $H^{k+1} = h_\theta^k\left(H^k\right)$



(b) Recurrent GNN : $H^{k+1} = h_\theta\left(H^k\right)$



(c) Ψ-GNN : $\widehat{H} = \text{RootFind}\left(h_\theta(H, G) - H\right)$

Figure C.11: Example of three types of GNN architecture. Figure C.11a displays the DSS approach, which consists of stacking different GNN layers until the final state is reached. Figure C.11b represents the Recurrent-GNN method, iterating over a GNN function h_θ for a fixed number K of iterations the final state is reached. Figure C.11c illustrates the proposed Ψ-GNN approach. The final solution $\widehat{H}$ is computed as the equilibrium point of the h_θ function using a black-box RootFind solver. The solver is initialized with an initial condition H^0, and the Message Passing iterations are performed implicitly within the solver.

and represents the core of our approach. A root-finding solver (i.e. a quasi-Newton method to avoid computing the inverse Jacobian of a Newton method at each iteration step) is used to determine the fixed point $\widehat{H}$ of the function h_θ (12) such that:

$$\widehat{H} = \text{RootFind}\left(h_\theta(H, G) - H\right) \tag{C.1}$$

Backward pass However, using a black-box solver prevents the use of explicit backpropagation through the exact operations performed in the forward pass. Thankfully, Bai et al. (2019) proposes a simpler alternative procedure that requires no knowledge of the black-box solver by directly computing the gradient at the fixed point. The gradient of the loss $\mathcal{L}$ with respect to the weights θ is then given by:

$$\frac{\partial \mathcal{L}}{\partial \theta} = -\frac{\partial \mathcal{L}}{\partial \widehat{H}}\left(J_{h_\theta}^{-1}|_{\widehat{H}}\right)\frac{\partial h_\theta(\widehat{H}, G)}{\partial \theta}. \tag{C.2}$$

where $\left(J_{h_\theta}^{-1}|_{\widehat{H}}\right)$ is the inverse Jacobian of h_θ evaluated at $\widehat{H}$. To avoid computing the expensive $-\frac{\partial \mathcal{L}}{\partial \widehat{H}}\left(J_{h_\theta}^{-1}|_{\widehat{H}}\right)$ term in (C.2), one can alternatively solve the following root finding problem using Broyden's method (or any other root-finding solver) and the autograd packages from Pytorch:

$$\left(J_{h_\theta}^{T}|_{\widehat{H}}\right) x^T + \left(\frac{\partial \mathcal{L}}{\partial \widehat{H}}\right)^T = 0 \tag{C.3}$$

Consequently, the model is trained using a RootFind solver to compute the linear system (C.3) and directly backpropagate through the equilibrium using (C.2).

Appendix C.3. Computing the regularizing term

In Section 3.4, we propose to rely on the method outlined in Bai et al. (2021) to regularize the model's conditioning. The spectral radius of the Jacobian $\rho\left(J_{h_\theta}\right)$, responsible for the stability of the model around the fixed point $\widehat{H}$, is constrained by directly minimizing its Frobenius norm since it is an upper bound for the spectral radius. This method prevents the use of computationally expensive methods (i.e. Power Iteration Golub and Van der Vorst (2000) for instance). The Frobenius norm is estimated using the classical Hutchinson estimator (Hutchinson (1990)):

$$||J_{h_\theta}||_F^2 = \mathbb{E}_{\epsilon \in \mathcal{N}(0, I_d)}\left[||\epsilon^T J_{h_\theta}||_2^2\right] \tag{C.4}$$

where $J_{h_\theta} \in \mathbb{R}^{d \times d}$. The expectation (C.4) can be estimated using a Monte-Carlo method for which (empirically observed) a single sample suffices to work well.

Appendix D. Theoretical proofs

Proof (Proposition 1).
If $h_G^\star$ is a solution to the problem (21), then its fixed point is a candidate solution to the problem (20) and we have:

$$\mathcal{L}_{\text{res}}(\text{FixedPoint}(h_G^\star),\ G) \geqslant \mathcal{L}_{\text{res}}(U_G^\star,\ G)$$

Reciprocally, for a fixed G, if $U_G^\star$ is a solution to the problem (20), then the function $h_G(H) = U_G^\star$ which always outputs the same value has a unique fixed point, namely $U_G^\star$. Considering this function as a candidate to the problem (21), we have:

$$\mathcal{L}_{\text{res}}(U_G^\star,\ G) = \mathcal{L}_{\text{res}}(\text{FixedPoint}\left(h_G\right),\ G) \geqslant \mathcal{L}_{\text{res}}(\text{FixedPoint}(h_G^\star),\ G)$$

Consequently,

$$\mathcal{L}_{\text{res}}(U_G^\star,\ G) = \mathcal{L}_{\text{res}}(\text{FixedPoint}(h_G^\star),\ G)$$

and the problems are equivalent.

Proof (Proposition 2).
Corollary 1 in Donon et al. (2020) assumes that four hypotheses must be fulfilled:

1. The loss function $\mathcal{L}_{\text{res}}$ is continuous and permutation-invariant.
2. The solution $U_G^\star$ is unique.
3. The problem distribution G satisfies permutation-invariance, compactness, connectivity (each graph having a single connected component), and separability of external outputs (ensuring node identifiability).
4. The solution is continuous with respect to G.

The first hypothesis is immediately satisfied by the specific loss function, which is suitable for problems involving GNNs and similar to the one described in Donon et al. (2020). Additionally, the existence and uniqueness of the solution are guaranteed thanks to a FEM analysis of problem (1) which includes at least one Dirichlet boundary condition. This analysis, based on the Lax-Milgram theorem (Larson and Bengzon, 2013), validates the second hypothesis. Regarding the properties of the problem distribution, our dataset generator (see Section 5.1) generates graphs with smooth boundaries within a bounded domain, following a rotation-equivariant law and ensuring that its inside has only one connected component. Node identifiability within a graph is achieved through the use of edge features representing distances to the closest neighbors, which are never equal, thereby validating the third hypothesis. If full identifiability is desired (not just almost surely), an additional descriptor can be added to each node. Experiments have actually been run in that setting, with no observable difference in performance, thereby validating the third hypothesis. The continuity of the solution $U_G^\star$ with respect to G depends on the specific choice of the

loss function $\mathcal{L}_{\text{res}}$. For Poisson-like problems similar to those considered in this study, continuity can be established based on the following Lemma 1, and this will conclude the proof.

Lemma 1 (Continuity of φ)**.**
The mapping

$$\varphi : \begin{array}{rcl} \mathcal{S} & \to & \mathbb{R}^N \\ G = (N, A, B) & \mapsto & U^\star(G) := \underset{U}{\operatorname{argmin}} \, \|AU - B\|^2 \end{array}$$

is continuous w.r.t. A and B.

Proof (Lemma 1).
Linear systems such as (2), which result from the FEM discretization of the Poisson problem (1), have a unique solution given by $U^\star(G) = A^{-1}B$. This solution is linear in B and thus continuous with respect to B. The question at hand is whether the mapping from A to its inverse A^{-1} is also continuous. We can express A^{-1} as

$$A^{-1} = \frac{\text{adj}(A)}{\det(A)}$$

where $\text{adj}(A)$ denotes the adjoint of A and $\det(A)$ represents the determinant of A. Both the adjoint operation and the determinant are continuous operations. The remaining aspect to consider is whether the determinant of A can be zero or not. In the specific settings where A arises from the discretization of (1), including boundary conditions, the determinant of A is always non-negative. This completes the proof.

Proof (Theorem 1).
The architecture of $\widehat{\varphi}$ obtained by Corollary 1 can be decomposed into two blocks: a $\widehat{\varphi}_{\text{GNN}}$ which consists of GNN layers that are applied to a hidden state H, and a $\widehat{\varphi}_{\text{Dec}}$ block that represents the decoder (see Section 3.3). Here, we attempt to build a function whose fixed point w.r.t. H approximates the output of the $\widehat{\varphi}_{\text{GNN}}$ block. To do this, we consider the function

$$h_{\theta_\varepsilon} : \begin{array}{rcl} \mathbb{R}^{N\times d} \times \mathcal{S} & \to & \mathbb{R}^{N\times d} \\ (H,\ G) & \mapsto & \widehat{\varphi}_{\text{GNN}}(G) \end{array}$$

where d is the dimension of the hidden space. For any fixed G, this function h_{θ_ε} is constant w.r.t. H, and, consequently has a unique fixed point w.r.t. H, which is $\widehat{\varphi}_{\text{GNN}}(G)$. As Ψ-GNN encompasses the $\widehat{\varphi}$ architecture, we can indeed represent exactly h_{θ_ε} using the Ψ-GNN Processor defined in Section 3.3. The complete architecture of Ψ-GNN can be written as follows:

$$\widehat{U} = \text{Dec}\left(\text{FixedPoint}(h_{\theta_\varepsilon}\left(\text{Enc}\left(U\right), G\right)\right)$$

where the decoder Dec is $\widehat{\varphi}_{\text{Dec}}$. The encoder Enc, which is an additional feature of Ψ-GNN architecture, can be chosen arbitrarily since, in this proof, h_{θ_ε} always outputs the same value, regardless of H. This completes the proof.

Proof (PROPOSITION 3).
The trainable function $h_{\theta_\varepsilon}(H, G)$ from Theorem 1 can be assumed to be an approximation of a contractive function f (e.g., as built in the proof). For the sake of simplicity, we omit the Decoder here. Thus, there exists $\lambda < 1$ such that, for any problem G and any latent values H, H':

$$d(f(H), f(H')) \leqslant \lambda\, d(H, H')$$

where d denotes the Euclidean distance. Note that $\lambda \in [0, 1[$ can even be 0, e.g. for $f(H) = U_G^\star$. Consequently, we have :

$$d(h(H), h(H')) \leqslant d(f(H), f(H')) + 2\varepsilon \leqslant \lambda\, d(H, H') + 2\varepsilon$$

given that $\|h(H) - f(H)\| \leqslant \varepsilon$ and $\|h(H') - f(H')\| \leqslant \varepsilon$. Suppose ε is small enough such that $\mu = \lambda + 2\sqrt{\varepsilon} < 1$. Then, for H, H' satisfying $d(H, H') > \sqrt{\varepsilon}$, we have:

$$d(h(H), h(H')) \leqslant \mu\, d(H, H')$$

since $\lambda\, d(H, H') + 2\varepsilon < \mu\, d(H, H')$ is equivalent to $d(H, H') > \frac{2\varepsilon}{\mu - \lambda} = \sqrt{\varepsilon}$. Thus h is contractive for all pairs of points farther than $\sqrt{\varepsilon}$ from each other. In particular, if $d(H, h_G^\star) > \sqrt{\varepsilon}$, then $d(h(H), h(h_G^\star)) \leqslant \mu\, d(H, h_G^\star)$, which implies the exponential convergence of an iterative power method from any initialization H to the ball of radius $\sqrt{\varepsilon}$ around the fixed point $h_G^\star$, which is itself at distance at most ε from the true optimal solution $U_G^\star$. Thus the function h is predominantly contractive in that sense. Using $\varepsilon' = \varepsilon + \sqrt{\varepsilon}$ then gets rid of square roots.