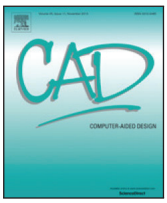
Computer-Aided Design 172 (2024) 103709

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad



Research Paper

Neural PDE Solvers for Irregular Domains

Biswajit Khara [a,1], Ethan Herron [a,1], Aditya Balu [a], Dhruv Gamdha [a], Chih-Hsuan Yang [a], Kumar Saurabh [a], Anushrut Jignasu [a], Zhanhong Jiang [a], Soumik Sarkar [a], Chinmay Hegde [b], Baskar Ganapathysubramanian [a,2], Adarsh Krishnamurthy [a,*]

[a] Iowa State University, Ames, IA, United States of America
[b] New York University, New York, NY, United States of America

ARTICLE INFO

Keywords:
Neural PDE solvers
Immersed / carved-out geometries
Error analysis

ABSTRACT

Neural network-based approaches for solving partial differential equations (PDEs) have recently received special attention. However, most neural PDE solvers only apply to rectilinear domains and do not systematically address the imposition of boundary conditions over irregular domain boundaries. In this paper, we present a neural framework to solve partial differential equations over domains with irregularly shaped (non-rectilinear) geometric boundaries. Given the shape of the domain as an input (represented as a binary mask), our network is able to predict the solution field, and can generalize to novel (unseen) irregular domains; the key technical ingredient to realizing this model is a physics-informed loss function that directly incorporates the interior-exterior information of the geometry. We also perform a careful error analysis which reveals theoretical insights into several sources of error incurred in the model-building process. Finally, we showcase various applications in 2D and 3D, along with favorable comparisons with ground truth solutions.

1. Introduction

Motivation: Many natural and engineered systems are governed by partial differential equations (PDEs) that describe how physical quantities vary in space and time. Examples of such systems include fluid dynamics, heat transfer, electromagnetism, and elasticity. Various methods have been developed that discretize the physical domain and the PDEs into a system of algebraic equations that computers can solve to obtain numerical solutions for PDEs. Among these methods, the finite difference (FDM), finite element (FEM), and spectral methods are the most widely used and studied [1–3]. These methods rely on choosing appropriate basis functions representing the solution over the discretized domain with a certain degree of accuracy and efficiency. However, a common challenge these methods face is handling complex, irregular domains that often arise in practical applications. For example, in computational fluid dynamics, one may need to simulate the flow around a curved or twisted shape, such as an aerofoil or a turbine blade. In biomedical engineering, one may need to model the blood flow or the electrical activity in a patient-specific organ, such as the heart or the brain. These domains are difficult to discretize using standard techniques, such as structured grids or simple geometric elements. Moreover, the boundary conditions imposed on the domain boundaries may be complicated, which requires special treatment in the discretization process. As a result, mesh generation and adaptivity, which are the tasks of creating and refining the discretization of the domain, are often the most time-consuming and labor-intensive steps in solving PDEs on complex domains. This problem has been recognized as one of the major bottlenecks in advancing the state-of-the-art in computational science and engineering, as stated in the NASA CFD 2030 [4] vision document.

Motivated by these challenges, this paper addresses the following problem: designing a neural PDE solver to produce field solutions across arbitrary geometries. Our approach is rooted in the immersed boundary method (IBM) [5], a well-established computational mechanics technique for solving PDEs on complex domains. The basic idea of IBM is to embed the irregular domain in a larger regular domain, usually a square (2D) or cube (3D), and impose the boundary conditions on the embedded boundary using a penalty method [6]. We adopt this basic idea with a key difference in this work: we approximate the arbitrary geometry by its axis-aligned approximant. This pixelated geometry is well suited for representation using cartesian meshes. Additionally, while conventional IBM applies Dirichlet boundary conditions (DBC) weakly on the exact boundary, this representation allows strong enforcement of DBC on the pixelated boundary. Another upshot is that

* Corresponding author.
E-mail addresses: baskarg@iastate.edu (B. Ganapathysubramanian), adarsh@iastate.edu (A. Krishnamurthy).
[1] Equal contribution.
[2] Co-corresponding author.

https://doi.org/10.1016/j.cad.2024.103709
Received 13 April 2023; Received in revised form 16 February 2024; Accepted 7 April 2024
Available online 14 April 2024

Notations used in the paper

PDE	partial differential equation
FEM	finite element method
NN	neural network
IBN	irregular boundary network
q	random variable / parameter
Ω_B	background domain
Ω_o	object domain
Ω	computational domain (where the PDE is defined)
θ	network parameters
χ	occupancy / indicator function with respect to the computational domain Ω
h	size of an element in a uniform mesh
u	unknown solution of the PDE
u^h	an h in the superscript is used to define a discrete counterpart of a continuous u
G_{ibn}	the neural network as a function

the PDE can be discretized using standard methods on the regular domain while still accounting for the effects of the irregular boundary[3]. We extend this powerful idea to neural PDE solvers, a class of methods that use neural networks to approximate the solution of PDEs [8,9].

Combining the above idea of an approximate geometry "immersed" in a regular background mesh, we propose a neural network architecture that can produce accurate and smooth solutions for a variety of geometries and also allows the natural incorporation of different boundary conditions (see Fig. 1). Our approach also allows us to compute *a priori* error estimates using a combination of techniques from neural network generalization theory [10] and finite element analysis [11]. This allows us to assess the quality and reliability of our solutions and compare them with other methods. Our main contributions are as follows:

1. **Framework:** We present a PDE-based loss function that learns robust and watertight boundary conditions imposed by complex geometries. Using this loss function, we train a deep neural network—Irregular Boundary Network (IBN)—that uses the geometric information as input and predicts a field solution that satisfies the governing PDE over the arbitrary domain. We show that a single trained IBN can produce solutions to a PDE across different arbitrary shapes.
2. **Error Analysis:** We provide an analysis of convergence and generalization error bounds of the proposed PDE-based loss function with the immersed approach.
3. **Applications:** Finally, we use this parametric PDE-based loss function to learn solutions of PDEs over irregular geometries. We illustrate the approach on two PDEs—Poisson's and Navier–Stokes—and a host of irregular geometries in both 2D and 3D.

Consequently, IBN opens up fast design exploration and topology optimization for various societally critical applications.

[3] Studies have shown that with moderate levels of discretization, the solution of elliptic PDEs on pixelated meshes converges rapidly to the solution on a body fitted mesh. Furthermore, recent mathematical advances—specifically the Shifted Boundary Method (SBM) [7]—make extending the mathematical formulation to account for the smooth geometry on a pixelated domain relatively straightforward. We defer this development to future work.

2. Related works

Neural PDE Solvers: Since neural networks are powerful nonlinear function approximators, there has been a growing interest in using neural networks to solve PDEs [8,12–24]. Unlike numerical methods, many of these methods do not require a mesh. But a common challenge most neural PDE solvers face is the efficient imposition of boundary conditions, especially on non-cartesian boundaries [25]. Furthermore, neural network function spaces are non-trivial to characterize. This makes obtaining regularity estimates for collocation-based neural solvers very difficult [26].

Immersed/Carved out Approach: Classical numerical methods such as finite difference (FDM) or finite element methods (FEM) generally employ a grid or mesh to discretize the domain geometry and function space. Solving PDEs defined on complex geometries requires a mesh to be generated before the analysis. This step, commonly known as the "mesh generation" step, is non-trivial and often expensive. One way to overcome this challenge is the immersed method [5,6,27,28]. The computational grid is simplified in immersed methods by considering a rectilinear axis-aligned grid that encloses the irregular geometry (within which we seek a PDE solution). The irregularly-shaped geometry is then "immersed" in this background mesh (see Fig. 2). Thus, a part of the background mesh forms the actual computational mesh; the rest of the mesh is considered exterior and thus not used in the computation of the PDE solution. In recent years, immersed methods have been favored for massive parallel implementations since all computations are performed on regular grids [29–32]. This fact translates to tensor-based operations in convolutional neural networks, which are unsuitable for complex geometrical contours. A key ingredient is the careful design of the loss function, along with a mechanism to determine the interior/exterior (in-out) of the computational domain.

A related but slightly different method is the "carving out" approach [33,34]. In this method, the arbitrary geometry is still "immersed" inside the background mesh, but now the geometry is *approximated* using structured grids, and then the boundary conditions are applied exactly on this approximate geometry. The main difference with the classical immersed method is that the classical immersed method applies weak boundary conditions on the exact boundary, whereas the carving out method applies strong boundary conditions on an approximate geometry. The "carving out" approach is the closest in spirit to the method we present in this paper.

Neural methods for arbitrary geometries: Several methods have been proposed for solving PDEs on arbitrary domains using neural networks. PhyGeoNet [35] is a neural method that attempts to map the real geometry to an axis-aligned reference mesh and applies regular convolutional techniques to calculate derivatives and eventually solve PDEs. This is a grid-based method. On the other hand, there are collocation-based methods, where distance fields have been used to properly adjust the contribution of the domain and boundary integrals in the loss function [26,36,37]. Another technique uses two different neural networks, one for satisfying the PDE inside the domain, and another to satisfy the boundary conditions on an irregular boundary [37,38], sometimes combining the distance functions. There are also neural methods inspired by the boundary element methods [39–41]. In [41], the unknown parameters of the domain are eliminated, and the neural network is parameterized for only the values at the boundary. If the PDE is solved on a manifold, then it is possible that points that are close in the Euclidean space may or may not be close on the manifold. To deal with such applications in neural methods, the eigenvalues of the Laplace–Beltrami operator over the manifold have been considered as inputs to the neural network, as opposed to the Euclidean coordinates [42].

All the works mentioned above are collocation based methods (with the exception of PhyGeoNet [35]), and they attempt to solve a single instance of a PDE. A common issue with such methods is that it is non-trivial to apply the boundary conditions exactly. Except for

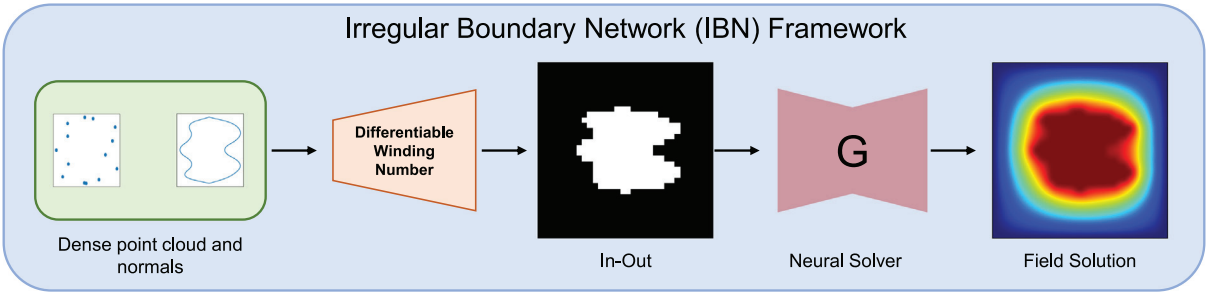


Fig. 1. An overview of the IBN framework. The framework accepts a point cloud describing the complex geometry or a Binary representation of the complex geometry denoting its in-outs. If a point cloud is provided, the differentiable winding number computation is used to compute the in-out of the complex geometry. The neural PDE solver maps the Binary representation of the complex geometry to the corresponding PDE field solution.

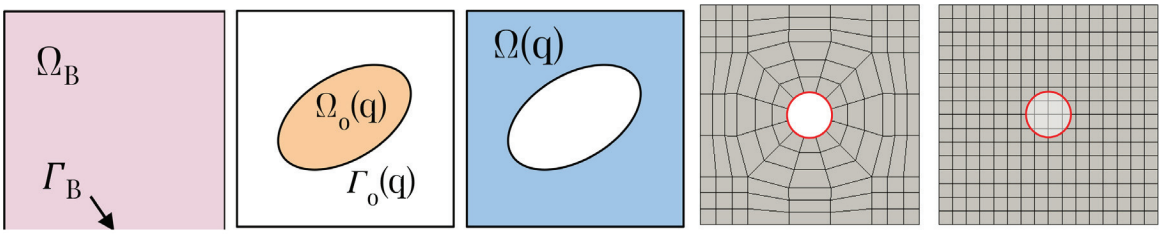


Fig. 2. (a) Schematic of a typical domain for the background mesh Ω_B (with boundary Γ_B), (b) an embedded/immersed object $\Omega_o(q)$ (with boundary $\Gamma_o(q)$), (c) the computational domain $\Omega(q)$, (d) A "body-fitted" mesh that discretely conforms to the object, and (e) an object embedded/immersed in a background mesh. This paper focuses on embedded/immersed type meshes with different object geometries.

some limited cases [9], it is generally difficult to construct a solution ansatz that satisfies the boundary conditions. Therefore, most of these methods attempt to satisfy the boundary conditions weakly through a penalty method or by another neural network [8,43]. Moreover, the loss function in the optimization problem is a combination of both the interior PDE loss, and the loss incurred on the boundaries. But these two quantities can be of different orders of magnitude and may have different convergence behavior; thus it is important to combine them with suitable weights, which is generally a hyperparameter that is open to adjustment [44–46]. Our method on the other hand, is a grid-based method; therefore, the exact imposition of the Dirichlet boundary conditions is straightforward. And as a consequence, no hyperparameter needs to be adjusted.

PhyGeoNet [35] falls under the grid based methods, and it attempts to solve both non-parametric and parametric problems. This makes PhyGeoNet the closest in spirit to our present work. The primary idea in PhyGeoNet is to transform domains with curved geometries to axis-aligned domains and solve the PDE using convolutional neural networks. It is thus applicable to geometries that are homeomorphic to hyperrectangles. For example, in 2D, PhyGeoNet can solve PDEs on domains that have four C^0-continuous boundaries. As such, it faces the limitation that a complex shape which is not homeomorphic to a hyperrectangle, cannot be solved by PhyGeoNet. Our present method excels in this respect since we do not assume any special property of the geometry, instead, we make an approximation in representing the boundary. This also obviates any necessity of transformation between different domains.

3. Mathematical preliminaries

3.1. Partial differential equations

Consider a probability space (Q, F, ψ), where Q is the sample/event space, F is the σ-algebra of the subsets of Q, and ψ is a probability measure. Consider also the domain $\Omega_B \subset \mathbb{R}^d$ $(d \in \{2,3\})$ with a rectilinear boundary Γ_B. Without loss of generality, the d-dimensional unit interval $[0,1]^d$ can be considered an example of Ω_B. Finally consider a subdomain $\Omega_o(q) \subset \Omega_B$ with an irregular boundary $\Gamma_o(q)$, where $q \in Q$. This means that the object and its boundary Γ_o depend on the random variable q. Define $\Omega(q) = \Omega_B \backslash \Omega_o(q)$ (see Fig. 2(a)–2(c) for an illustration in 2D Euclidean space). In this paper, we will primarily focus on a family of equations where the object-boundary $\Gamma_o(q)$ varies according to some given probability distribution. Then we consider an abstract PDE (Eq. (1a)) with boundary conditions (Eq. (1b)–(1c)) given by:

$$\mathcal{N}[u(\underline{x}; q)] = f(\underline{x}) \quad \text{in} \quad \Omega(q), \tag{1a}$$

$$u(\underline{x}, q) = 0, \quad \text{on} \quad \Gamma_B, \tag{1b}$$

$$\alpha u + \beta(\nabla u \cdot \hat{\underline{n}}) = g(\underline{x}), \quad \text{on} \quad \Gamma_o(q). \tag{1c}$$

Here, $u : \Omega(q) \to \mathbb{R}$ is unknown, $\mathcal{N}$ is a differential operator (possibly nonlinear) and $f(\underline{x}), g(\underline{x})$ are known functions of the domain variable $\underline{x}$. Eq. (1b) describes a Dirichlet boundary condition on the exterior rectilinear boundary of Ω_B. Eq. (1c) prescribes the conditions on the (irregular) object boundary Γ_o which depends on the random parameter q. We assume that the boundary $\Gamma(q)$ is Lipschitz continuous with respect to $\underline{x}$. The constants $\alpha, \beta \in R$, and $g \in L^2(\Gamma(q))$. Specific boundary conditions can be written by taking specific combinations of α and β (see Section 3.1.1 for detail). In this paper, we look at two concrete examples of Eq. (1): Poisson's and Navier–Stokes equations.

3.1.1. Poisson's equation

Poisson's equation is frequently used to model steady-state mass/heat diffusion, electrostatics, surface reconstruction, etc. Poisson's equation is frequently posed in parametric geometries, and can be stated using the notation developed in Section 3.1 as below:

$$-\nu\Delta u(\underline{x}, q) = f(x) \quad \text{in} \quad \Omega(q), \tag{2a}$$

$$u(\underline{x}, q) = 0, \quad \text{on} \quad \Gamma_B, \tag{2b}$$

$$\alpha u + \beta(\nabla u \cdot \hat{\underline{n}}) = g(\underline{x}) \quad \text{on} \quad \Gamma_o(q), \tag{2c}$$

where u is a scalar function that, depending on the underlying physics, may represent the mass density, temperature, electric potential, or

the surface indicator function, respectively. The variable $\nu > 0$ is the diffusivity of the material in the domain, and is assumed to be constant in this work. Finally, different boundary conditions (Dirichlet, Neumann, Robin) are produced by varying α and β, e.g., $(\alpha, \beta) = (1, 0)$ denotes Dirichlet boundary condition, whereas $(\alpha, \beta) = (0, 1)$ represents Neumann condition.

3.1.2. Navier–Stokes equations

The Navier–Stokes equations are widely used to model fluid flow. The steady incompressible Navier–Stokes equations are given by:

$$\underline{u} \cdot \underline{\nabla} \underline{u} - \nu \Delta \underline{u} + \underline{\nabla} p = \underline{f} \quad \text{in} \quad \Omega \tag{3a}$$

$$\underline{u} = \underline{g} \quad \text{on} \quad \Gamma \tag{3b}$$

where $\underline{u} : \Omega \to \mathbb{R}^d$ is a vector valued function that represents the velocity field, and $p : \Omega \to \mathbb{R}$ is a scalar representing the pressure in the fluid. The coefficient ν is the viscosity of the fluid. Note that $\underline{g}$ can be a function of the spatial location.

3.2. Finite element discretization

Eqs. (2) and (3) provide the continuous form of the respective equations. This means that whenever applicable, these equations apply to every point in the continuous medium or material. In engineering applications, however, the problem is usually reduced to obtaining a numerical solution on a *finite* number of points (as opposed to *infinite* number of points). This process is known as "discretization", and in this work, we rely on FEM for this process.

Essentially, we use FEM to discretize the domain spaces (Ω_B or Ω) and the functions defined on them (such as u). Let $\mathcal{K}^h$ be a discretization of Ω_B into n_{el} finite elements and N nodes. The ith element is K_i and $\cup_{n_{el}} K_i = \Omega_B$. The nodes are represented by their coordinates $\mathbf{X} = \{X_i\}_{i=1}^N$.

Following standard FEM analysis, we define a function space

$$V = \left\{ v \in H^1(\Omega) : v|_{\Gamma_B} = 0,\ v|_{\Gamma_o(q)} = g \right\}. \tag{4}$$

The associated norm is given as

$$\|v\|_V = \mathbb{E}_{q\sim Q} \left[\int_\Omega \nu(\underline{x}) \left| \nabla v(\underline{x}, q) \right|^2 d\underline{x} \right]^{1/2}. \tag{5}$$

Then we can define the discrete function space

$$V^h = \left\{ v^h \in V : v^h|_K \in Poly_m(K),\ K \in \mathcal{K}^h \right\}, \tag{6}$$

where $V^h \subset V$, and $Poly_m(K)$ denotes the set of polynomial functions of degree m defined on K. Since the background mesh is regular, $\mathcal{K}^h$ is composed of a rectilinear axis-aligned grid with N nodes. Now, suppose $\{B_i(\underline{x})\}_{i=1}^N$ is a suitable basis that span V^h. Any function $u^h \in V^h$ can be written as

$$u^h(\underline{x}) = \sum_{i=1}^{N} \mathcal{B}_i(\underline{x}) U_i, \tag{7}$$

where U_i are the solution values at the nodal points in the mesh $\mathcal{K}^h$. Also, define $\mathbf{U} = \{U_i\}_{i=1}^N$. In the sequel, when we refer to the discrete approximation of the solution, we mean either u^h (the functional representation) or the discrete set of nodal values $\mathbf{U}$ or $\{U_i\}$ (the vector representation). The two will be used interchangeably under the assumption that the basis functions $\mathcal{B}(\underline{x})$ are known.

3.3. Immersed object

In this paper, we formulate our neural PDE solver based on an immersed FEM discretization [5,6,27,28]. In terms of the notations developed above, we discretize Ω_B using an axis-aligned grid (Fig. 2(d)–2(e)), and we call this a "background mesh". To incorporate the effect of Ω_o, the boundary curve Γ_o is considered "immersed" in this background mesh. However, *instead of weakly enforcing* the Dirichlet boundary conditions on exact Γ_o (which is the norm in the classical immersed methods), *we apply the boundary conditions strongly* on $\tilde{\Gamma}_o$ which is a discrete approximation of Γ_o (see Section 4.2.2 for details). This formulation can be naturally extended to accommodate several flavors of immersed formulations popular in literature (see also Section 8), including immersogeometric methods [47,48], as well as the more recent shifted boundary methods [7].

3.3.1. Occupancy function

The occupancy function (or indicator function) χ is a simple function defined as $\chi : \Omega_B \to \{0, 1\}$, such that

$$\chi(x; q) = \begin{cases} 1, & x \in \Omega(q) \\ 0, & x \in \Omega_B \backslash \Omega(q). \end{cases} \tag{8}$$

We denote the discrete counterpart of χ by χ^h (functional representation, similar to u^h) and $\boldsymbol{\chi}$ (vector representation, similar to $\mathbf{U}$).

4. Our approach: Irregular boundary network (IBN)

The method we present, named the *irregular boundary network* (IBN) produces a (field) solution to a given PDE while adhering to the boundary conditions imposed by a *family* of complex geometries.

4.1. Description of the mapping

If we fix Ω_B, f, g and (α, β), then Eq. (1) is a family of equations parameterized by q. So, it is natural to seek an operator that maps the occupancy function $\chi^h(q)$ to the solution $u^h \in V^h$. So, conceptually, we look for an operator $\mathcal{T}$ such that

$$\mathcal{T}[\chi^h(\underline{x}, q)] = u^h(\underline{x}, q). \tag{9}$$

Our goal in this paper is to find a neural approximation G_{ibn} to $\mathcal{T}$. As discussed in Section 3.3.1, the object can be discretely represented by χ^h (or the vector representation $\boldsymbol{\chi}$); and the solution by u^h (or the vector representation $\mathbf{U}$) (see Section 3.2, and also Eq. (7)). So, the IBN operator map can be written as

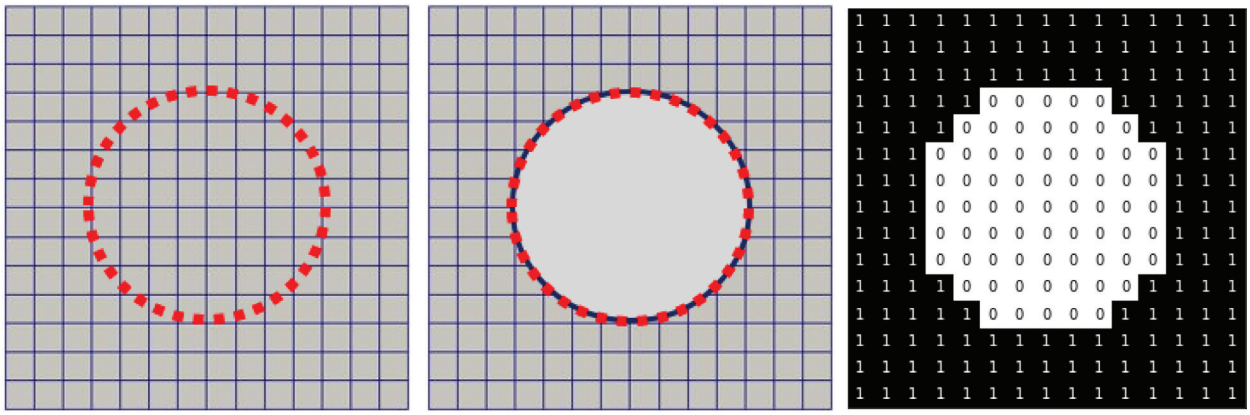
$$\mathbf{U}(q) = G_{ibn}(\boldsymbol{\chi}(q); \theta), \tag{10}$$

where θ is a set of tunable network weights. In other words, G_{ibn} represents an (approximate) operator that "learns" the complex relation between the distribution of object shapes (given by $\Omega_o(q)$) and their respective solutions $u^h(\underline{x}, q)$.

If N is the number of points used to discretize the background mesh, then $\boldsymbol{\chi} \in \mathbb{R}^N$ and $\mathbf{U} \in \mathbb{R}^{N \times n_{dof}}$. So, numerically, we have $G_{ibn}(\boldsymbol{\chi}; \theta) : \mathbb{R}^N \times \mathbb{R}^{|\theta|} \to \mathbb{R}^{N \times n_{dof}}$, where n_{dof} is the number of unknowns in the PDE (e.g., $n_{dof} = 1$ for Poisson's equation, and $n_{dof} = (d + 1)$ for Navier–Stokes equations). The network architecture used to encode the geometry information and predict the corresponding field solution is a standard UNet [49].

4.2. Training loss calculation

A key novelty of our method lies in formulating an appropriate loss function based on the PDE in the computational domain (Ω), while enforcing the Dirichlet boundary conditions exactly on the irregular geometries. But before presenting the loss function (defined in Section 4.2.4), we need to discuss the key ingredients, namely, the computation of χ^h, the application of boundary conditions and the calculation of the PDE residual.



(a) *Pointcloud* $\{P\}$ *with background mesh* Ω_B **(b)** *Representation of* $\Gamma_o(q)$ *(and* $\Omega_o(q)$*).* **(c)** χ^h *indicating the immersed object.*

Fig. 3. Steps to generate the interior and exterior of the computational domain $\Omega(q)$ with respect to Ω_B and $\Omega_o(q)$: (a) The background domain Ω_B (and the mesh $\mathcal{K}^h$), and the object boundary $\Gamma_o(q)$ represented by a pointcloud $\{P\}$ (red), (b) the object domain $\Omega_o(q)$ and the object boundary $\Gamma_o(q)$ represented by $\{P\}$, (c) χ^h is calculated for all points in $\mathcal{K}^h$ with respect to the input $\{P\}$ using Eqs. (11) and (12). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.2.1. Calculation of the occupancy function using generalized winding number

To determine if a point $\underline{x}$ in Ω_B falls inside or outside of Γ_o, we compute the winding number corresponding to $\underline{x}$. The general form of the winding number counts the number of loops of the boundary around any point $\underline{x}$ and is calculated as the surface integral of the differential solid angle ($\int_{\Gamma_o} dS(\underline{x})$) [50]. The winding number ω as point $\underline{x}$ is defined as:

$$\omega(\underline{x}) = \sum_{i=1}^{n} a_i \frac{(\underline{p_i} - \underline{x}) \cdot \hat{\underline{n}}_i}{4\pi \|\underline{p_i} - \underline{x}\|^3} \tag{11}$$

where a_i is the Voronoi area of a given point and $\hat{\underline{n}}_i$ is the normal for a given point $\underline{p_i} \in \{P\}$ (recall that Ω_o is represented by $\{P\}$) [50]. The discrete occupancy function χ^h can now be obtained from ω using

$$\chi^h = 1 - \omega. \tag{12}$$

4.2.2. Imposition of boundary conditions on an immersed object

The occupancy function computed in Eq. (12) determines the interior and exterior of the computational domain Ω. The discrete occupancy function χ^h also results in an approximation $\tilde{\Gamma}_o$ of the boundary Γ_o (and also an approximation $\tilde{\Omega}_o$ of Ω_o) represented by the pointcloud $\{P\}$ (see Fig. 3(c)). A Dirichlet condition on Γ_o can now be applied on $\tilde{\Gamma}_o$ strongly. For example, in Fig. 3(c), a Dirichlet condition is applied to all the points in the white region. This method differs slightly from the classical immersed methods where the boundary conditions are applied approximately near the boundary curve, but the boundary curve Γ_o itself is generally not approximated discretely [5,6,27,28]. A similar approach can be taken in case of homogeneous Neumann condition that represents a zero-flux across a boundary. To mimic the zero-flux condition, the diffusivity parameter (i.e., ν) can be assigned zero on the exterior (i.e., inside Ω_o).

4.2.3. PDE residual

Here we discuss the calculation of the PDE residual for a particular realization of $q = q_*$. This residual will be used to calculate the final loss function over multiple realization from the training set (see Section 4.2.4). When q is fixed, the loss function is simply derived from the PDE using Galerkin formulation. For the abstract PDE in Eq. (1) (we assume that Neumann conditions are all homogeneous), the corresponding Galerkin formulation is to find $u^h \in V^h$ such that

$$\left(v^h, [\mathcal{N}(u^h) - f]\right)_{\Omega(q_*)} = 0, \quad \forall v^h \in V^h. \tag{13}$$

V^h is defined in Section 3.2. In general, if a unique u^h exists, then for any other function $w^h \in V^h$ ($w^h \neq u^h$) will make the right hand side of Eq. (13) nonzero. We call this the residual of Eq. (13), i.e.,

$$R = \left(v^h, [\mathcal{N}(w^h) - f]\right)_{\Omega(q_*)} \tag{14}$$

where, $w^h \in V^h$. Minimizing this residual would provide us with a unique solution.

Unfortunately, we cannot perform integrations over $\Omega(q_*)$ in a straightforward manner, but we *can* do so on Ω_B. On the other hand, Ω_B also contains $\Omega_o(q_*)$ but we do not want to perform integrations on $\Omega_o(q_*)$. We resolve this via the discrete occupancy function $\chi^h(q_*)$, which helps us perform integrations on Ω_B while at the same time enforcing no contribution to R from $\Omega_o(q_*)$.

Using χ, the discrete representation of the residual defined in Eq. (14) is written as:

$$R^h = \overset{\circledA}{\sum_{\chi_K(q_*)>0}} \left(v^h, [\mathcal{N}(w^h) - f]\right)_K, \tag{15}$$

where $\circ A$ signifies that the summation is to be understood in the sense of the so-called "assembly" process in FEM [1]. The set $\{\chi_K = 0\}$ is used to apply the boundary condition on $\Gamma_o(q_*)$. This computation is carried out on the background mesh Ω_B which is a regular (or structured) grid, therefore it can be vectorized (see Section 5).

4.2.4. Optimization problem

Eq. (15) calculates the residual for *one realization* of q. The total loss function over all realizations from Q is formulated as

$$\mathcal{J} = \mathbb{E}_{q\sim Q}[R^h(q)]^2. \tag{16}$$

In practical scenarios, we cannot evaluate the expectation in Eq. (16) because we may not have access to every possible event in Q. A more plausible scenario is to have a finite number of samples from Q available to us. In that case, we can replace the expectation in Eq. (16) by a sum over a finite number of samples drawn from Q.

Suppose we have a set of n_t samples $q_i \in Q_t \subset Q$, $i = 1, \ldots, n_t$. Define the "training sample set"

$$S_t = \left\{\Omega_o : \Omega_o = \Omega_o(q), q \in Q_t\right\}. \tag{17}$$

Then the loss function can be evaluated over the set S_t as

$$\mathcal{J}(\theta) = \frac{1}{n_t} \sum_{\Omega_o \in S_t} \left\| R^h_{\Omega_o}(\theta) \right\|_2^2 = \frac{1}{n_t} \sum_{i=1}^{n_t} \left\| R^h(q_i; \theta) \right\|_2^2. \tag{18}$$

Table 1
Comparison between IBN and other major methods. Notation: x = coordinate variable, u = solution function, q = random variable, $s(q)$ = stochastic input data (material property/object geometry etc.). Small letters (x, u, q, s) denote continuous variables whereas capital letters stand for the discrete counterpart.

	Non-parametric	*Parametric*		
	PINN	DeepONet	FNO	IBN
Mapping	$x \mapsto u$	$(x, s(q)) \mapsto u$	$\mathbf{S} \mapsto \mathbf{U}$	$\mathbf{S} \mapsto \mathbf{U}$
Spatial derivatives	Automatic diff. through NN	NA	NA	FEM-based
Training type	PDE-driven (data-free)	Data-driven	Data-driven	PDE-driven (data-free)
Loss function	PDE residual	MSE loss	MSE loss	(weighted) PDE residual
Training Dataset	Input: X, target: *none*	$(\mathbf{S}, \mathbf{U})$ pair Input: $\mathbf{S}$, target: $\mathbf{U}$	$(\mathbf{S}, \mathbf{U})$ pair Input: $\mathbf{S}$, target: $\mathbf{U}$	Input: $\mathbf{S}$, target: *none*

Then we can finally define the minimization problem as

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{J}(\theta). \tag{19}$$

which can be approximately solved using the stochastic gradient descent method.

Note that the computation of the loss function Eq. (18) requires two different numerical integration. One is the spatial quadrature rule used for the evaluation of each $R^h(q_i; \theta)$; and the second is the quadrature for integration over the probability space (which is represented as a summation in Eq. (18)).

4.3. Discussion on the method

We provide here a discussion reflecting on the choice of the mapping type and loss function in IBN, placing it in relation with other well known state-of-the-arts methods. A summary is presented in Table 1.

4.3.1. Mapping type

It is evident that the choice of mapping (G_{ibn} defined in Eq. (10), see also Eq. (9)) has the following traits:

- G_{ibn} is an operator.
- G_{ibn} maps (a distribution of) input data[4] ($\Omega_o(q)$) to the (distribution of) solutions (u^h).
- Both the input and output are *spatially discrete* functions (i.e., there is an underlying spatial grid/mesh).
- G_{ibn} is designed to work for a problem of stochastic nature.

These characteristics conceptually place IBN in close proximity to FNO (and its variants) and PhyGeoNet. IBN also shares the operator nature of DeepONet; however, they differ in one respect, which is that DeepONet aims to create a map between discrete input functions and *continuous* output function (as opposed to discrete output function in case of IBN).

On the other hand, IBN differs from methods such as PINNs, which are designed for a *single instance* of a PDE, i.e., such methods need to be retrained for every realization of q. A second major difference is that PINN-like methods aim to provide the *function mapping* $\underline{x} \mapsto u^h$, whereas IBN provides a much more complex map of $\chi^h(q) \mapsto u^h$ (subject to an underlying grid).

4.3.2. Loss function

The loss function (Eq. (18)) has the following traits:

- $\mathcal{J}$ is based on the PDE (i.e., the governing physical law)
- $\mathcal{J}$ uses a finite number of samples of $\Omega_o(q),\ q \in Q_t$.
- $\mathcal{J}$ does *not* use any ground-truth data (e.g., a "true" value for u^h)

These properties make IBN a "data-free physics-based operator". It is "data-free" in the sense that the calculation of $\mathcal{J}$ does not make use of any ground-truth data. The loss function *does* use samples taken from S_t for the purpose of integration over the training set, but does not require any "true" value of the corresponding solutions. Therefore, the training is data-free (or "*output data-free*") and is completely governed by the PDE residual.

Thus, conceptually, the IBN loss function is inspired by PINN-type loss functions, but adapted for operator learning. And as a result, it shares similarities with most such "data-free"/physics-based neural methods. On the other hand, IBN differs from data-driven methods such as FNO, DeepONet (when they are not augmented with physics based loss functions).

5. Implementation details

5.1. Network architecture

The neural network architecture G is a standard UNet architecture. The UNet is composed of convolution and transpose convolution operations with specifically placed skip connections to facilitate optimization and expressivity. In total, the neural networks used contained 4.2 million parameters for both 2D and 3D cases. All cases were optimized with the Adam optimizer and a learning rate of 3e−4. An in-depth analysis of network architectures and other hyperparameters tuning was left for future work.

5.2. Boundary conditions

Using χ^h, we can accurately apply the boundary conditions to arbitrary complex domains. The nodes assigned $\chi = 0$ are understood to be inside the object (outside the computational domain) and $\chi = 1$ vice versa. If a DBC is imposed on Γ_o, then the values of the solution field are enforced to match the DBC in Ω_o. If a zero-NBC is applied on Γ_o, then the values of ν field is enforced to be zero on Ω_o (also see Sections 4.2.2 and 7.2.3). Nodes assigned $\chi = 1$ are understood to belong to the computational domain and are used for the PDE residual calculations (Eq. (15)). This is accomplished with a `torch.where()` function. The outer boundaries (i.e., Γ_B) are similarly assigned their respective Dirichlet boundary conditions.

5.3. Differentiation and integration

The computation of the loss function $\mathcal{J}$ in Eq. (18) requires an integration of the term $v^h[\mathcal{N}(w^h) - f]$ over Ω. And the evaluation of $\mathcal{N}(w^h)$, in turn, requires some differentiation (recall that $\mathcal{N}(\cdot)$ is a differential operator; also compare Eqs. (2) and (3)). But, this differentiation is not to be confused with the differentiation through the neural network (i.e., with respect to θ). Rather, these are spatial derivatives, and their calculation is done using the basis functions $\{\mathcal{B}_i(\underline{x})\}_{i=1}^N$ and therefore, the neural network is not responsible for these computations.

Finally, the integration is performed numerically using Gaussian quadratures. For a given mesh $\mathcal{K}^h$, the basis functions $\{\mathcal{B}_i\}$, as well

[4] Here "input data" means the data specified for the PDE (such as material property, initial/boundary conditions etc.) and not the training/testing data.

as the quadrature points, are known and are completely deterministic. This allows us to define the spatial gradients of the predicted field solution by simply evaluating each element with the first or second-order derivative of the basis function originally used to evaluate the predicted field solution. Since we use the FEM, we perform the integration in each discrete element and then perform a summation over the finite set of elements to obtain the total integral.

5.4. Differentiable winding number computation

The winding number for a given point cloud $\mathcal{P}$ at a given query point $\underline{x}$ is computed using Eq. (11). In order to evaluate the winding number at all the nodal locations, we perform all the pairwise distance computations between every nodal location and every point in the point cloud and perform a sum reduction of the pairwise distances as per the above equation. Using `pytorch`, we can achieve this with a simple broadcasting operation. This way, all the operations are accelerated using the GPUs.

6. Error analysis

In this section, we summarize theoretical results for the convergence behavior and the generalization error for IBN. The analysis presented below pertains to a *single test inference*, i.e., the total error incurred on a single test input $\chi^h(\hat{q})$, where $\hat{q}$ may or may not belong to Q.

For a given object geometry $\Omega_o(\hat{q})$ (boundary $\Gamma_o(\hat{q})$), we denote u as the original optimum solution to the PDE and u^h be the optimal solution at discretization level h; these are functions evaluated at any point $\underline{x} \in \Omega$. From classical FEM analysis, we will bound the discretization error $\|u^h - u\|$ as a function of h. Now, the field predicted by the network $G_{ibn}(\{P\}, \theta)$ is typically an inaccurate version of u_h, and we can express the error e_G as follows:

$$\|e_G^k\| = \|u_{\theta_k} - u\| = \|u_{\theta_k} - u_{\theta^*} + u_{\theta^*} - u^h + u^h - u\| \tag{20}$$

where, u_{θ_k} is the field solutions corresponding to the network parameters θ_k at the kth iteration of the optimization and u_{θ^*} is the theoretical optimum (i.e., a limit point) of u_{θ_k}. Note that the exact solution u is only defined inside the computational domain Ω, and is undefined inside the object Ω_o. Therefore, we redefine u by continuously extending it to Ω_o as

$$u \overset{\text{def}}{=} \Omega_\chi u + \Omega_{o_\chi} \sum_{i=1}^{N_D} \mathcal{B}_i(\underline{x}) g_i, \tag{21}$$

where Ω_χ and Ω_{o_χ} are the indicator functions for Ω and Ω_o, and N_D is the number of nodes in Ω_o where the Dirichlet condition of $u = g$ is applied.

We analyze the quadratic form of $\|e_G^k\|$, i.e., $\|e_G^k\|^2$; in the stochastic setting, the relevant quantity is the second moment, i.e., $\mathbb{E}[\|e_G^k\|^2]$. By applying the fundamental inequality $\|a+b+c\|^2 \leq 3(\|a\|^2 + \|b\|^2 + \|c\|^2)$, we can obtain

$$\|e_G^k\|^2 \leq 3\Big(\underbrace{\|u_{\theta_k} - u_{\theta^*}\|^2}_{\text{Optimization error}} + \underbrace{\|u_{\theta^*} - u^h\|^2}_{\text{Modeling error}} + \underbrace{\|u^h - u\|^2}_{\text{Discretization error}} \Big). \tag{22}$$

Thus the generalization error is bounded above by three different terms, namely, the *optimization error*, *modeling error*, and *discretization error*.

The discretization error is the error incurred due to the spatial discretization scheme, in this case the mesh size h. The modeling error is the error incurred due to the choice of neural network architecture. This is completely dependent on the approximation properties of the neural network. Both these errors remains static through out the optimization process. Finally, the optimization error is the error introduced by the optimization algorithm.

Among these three components, the discretization error is the most straight-forward to characterize. The following lemma follows standard results found in the finite element method literature.

Lemma 1. *Assume that the basis functions $\mathcal{B}_i(\underline{x})$ (see Eq. (7)) are chosen such that they are at least continuously differentiable locally over a mesh. Then*

$$\|u^h - u\|^2 \leq Ch^{2\alpha}, \tag{23}$$

where $C > 0$ and α is the order of continuous derivative. Typically, $\alpha \geq 1$.

The modeling error does not have a straight-forward relationship with the network architecture. But using approximation results for the Barron spaces [51], we can write

$$\|u_{\theta^*} - u^h\| < \epsilon, \tag{24}$$

where it is assumed that the underlying neural network G_{ibn} can be represented by a two-layer neural network G_B^m with m neurons and ReLU activation function. As $m \to \infty$, we have $\epsilon \to 0$.

Next, we analyze the optimization error. We begin with the following assumption.

Assumption 2. There exists a constant $L > 0$ such that for all $\underline{x}, \underline{y} \in \mathbb{R}^d$, $\|U_i(\underline{x}) - U_i(\underline{y})\| \leq L\|\underline{x} - \underline{y}\|$, for all $i \in \{1, 2, \ldots, N\}$.

Assumption 2 implies that the prediction function provided by IBN is Lipschitz continuous [52], which signifies the robustness of the predictions obtained from deep neural networks under perturbations of the network parameters. This is a standard assumption made during the analysis of neural network generalization.

Assumption 2 allow us to establish the relationship between $\|u_{\theta_k} - u_{\theta^*}\|$ and $\|\theta_k - \theta^*\|$. Using Eq. (7), we can write:

$$u_{\theta_k} := u^h(\mathbf{x}; \theta_k) = \sum_{i=1}^{N} \mathcal{B}_i(\mathbf{x}) U_i(\theta_k), \tag{25a}$$

$$u_{\theta^*} := u^h(\mathbf{x}; \theta^*) = \sum_{i=1}^{N} \mathcal{B}_i(\mathbf{x}) U_i(\theta^*) \tag{25b}$$

Subtracting one from the other, we have

$$\begin{aligned} \|u_{\theta_k} - u_{\theta^*}\| &= \|\sum_{i=1}^{N} \mathcal{B}_i(\mathbf{x}) U_i(\theta_k) - \sum_{i=1}^{N} \mathcal{B}_i(\mathbf{x}) U_i(\theta^*)\| \\ &\leq \sum_{i=1}^{N} \|\mathcal{B}_i(\mathbf{x})\| \; \|U_i(\theta_k) - U_i(\theta^*)\| \\ &\leq L \sum_{i=1}^{N} \|\mathcal{B}_i(\mathbf{x})\| \; \|\theta_k - \theta^*\|. \end{aligned} \tag{26}$$

where the first inequality is obtained using both the triangle inequality and the Cauchy–Schwarz inequality, whereas the second inequality is obtained using Assumption 2.

Notice that the loss function in Eq. (18) is strongly convex. To characterize the first main result, we present two well-known lemmas as follows.

Lemma 3. *If a continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly convex, for all $x, y \in \mathbb{R}^d$, then*

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \mu \|x - y\|^2. \tag{27}$$

Lemma 4. *If a continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly convex and β-smooth, for all $x, y \in \mathbb{R}^d$, then*

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \frac{\mu\beta}{\mu + \beta} \|x - y\|^2 + \frac{1}{\mu + \beta} \|\nabla f(x) - \nabla f(y)\|^2. \tag{28}$$

With these two lemma in hand, we are now ready to present a key auxiliary lemma.

Lemma 5. *Suppose that $\mathcal{J}$ is μ-strongly convex and β-smooth. By applying the gradient descent algorithm with a constant step size $\eta_k = \eta = \frac{2}{\mu+\beta}$, the iterates $\{\theta_k\}$ satisfy the following relationship*

$$\|\theta_K - \theta^*\|^2 \leq \left(1 - \frac{2}{\kappa + 1}\right)^{2K} \|\theta_0 - \theta^*\|^2, \tag{29}$$

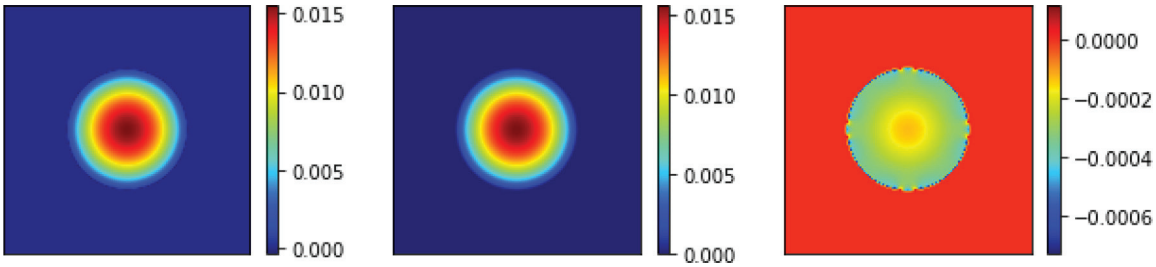


Fig. 4. Contours of u^h, u and $u^h - u$ with linear basis ($\alpha = 1$) for the Poisson's equation solved on a circular disk of radius $R = 0.25$.

Table 2
A summary of all the datasets used in this work.

Dataset name	Dimension	Type	Example images	Used in	Used for
A	2D	various objects (unknown parameterization)	Fig. 6	Case-1, 2	Training
B	2D	NACA airfoils, (parameterized)	Fig. 7, 8	Case-1, 2	Testing
C	2D	NURBS curves (parameterized)	Fig. 7, 8	Case-1, 2	Testing
D	3D	arbitrary images (unknown parameterization)	Fig. 11	Case-3	Training, Testing

where $\kappa = \frac{\beta}{\mu} \geq 1$ *and* K *is the number of epochs.*

Proof. Based on the gradient descent update, we have

$$\begin{aligned}
&\|\theta_k - \theta^*\|^2 \\
&= \|\theta_{k-1} - \eta\nabla\mathcal{J}(\theta_{k-1}) - \theta^*\|^2 \\
&= \|\theta_{k-1} - \theta^*\|^2 - 2\eta\langle\nabla\mathcal{J}(\theta_{k-1}), \theta_{k-1} - \theta^*\rangle + \eta^2\|\nabla\mathcal{J}(\theta_{k-1})\|^2 \\
&\leq \left(1 - 2\frac{\mu\beta\eta}{\mu+\beta}\right)\|\theta_{k-1} - \theta^*\|^2 + \left(\eta^2 - \frac{2\gamma}{\mu+\beta}\right)\|\nabla\mathcal{J}(\theta_{k-1})\|^2 \\
&= \left(1 - \frac{2}{\kappa+1}\right)^2\|\theta_{k-1} - \theta^*\|^2 \\
&\leq \left(1 - \frac{2}{\kappa+1}\right)^{2k}\|\theta_0 - \theta^*\|^2.
\end{aligned} \tag{30}$$

The first inequality follows from the substitution of the step size and Lemma 4. While the third equality is due to $\eta^2 - \frac{2\gamma}{\mu+\beta} = 0$. The desirable result is obtained by changing k to K. □

Theorem 6. *Suppose that* $\mathcal{J}$ *is* μ*-strongly convex and* β*-smooth. Let Assumption 2 hold. By applying the gradient descent algorithm with a constant step size* $\eta_k = \eta = \frac{2}{\mu+\beta}$*, the generalization error* $\|e_G^K\|^2$ *satisfies the following relationship after* K *epochs,*

$$\|e_G^K\|^2 \leq C_1\epsilon^2 + C_2h^{2\alpha} + C_3\left(\frac{\kappa-1}{\kappa+1}\right)^{2K}\|\theta_0 - \theta^*\|^2, \tag{31}$$

where $C_3 = 3L^2N^2\hat{B}^2$*;* N *is defined in Assumption 2,* μ *is the strong convexity constant,* β *is the smoothness constant,* L *is the Lipschitz continuity constant,* $\hat{B}$ *is the upper bound of the basis function* $\mathcal{B}_i(\underline{x})$*,* $\kappa = \frac{\beta}{\mu}$ *is the condition number, and* θ_0 *and* θ^* *are respectively the initialization and optimum of* θ*.*

Proof. The desired result can be obtained by combining results from Eqs. (26), (29), and (23). □

Theorem 6 implies that the generalization error of IBN is upper bounded by two terms, with the first term related to the optimization error and the second term the static error due to the discretization. Additionally, in an asymptotic manner, we have that

$$\lim_{K\to\infty}\|e_G^K\|^2 \leq 3Ch^{2\alpha}, \tag{32}$$

Table 3
Comparison of L2 norm of errors incurred by the IBN method and PINN, respectively, when compared with a baseline numerical solver. The errors and training times are listed for a select number of cases where the object is an airfoil. The first column denotes the name of the airfoil in the NACA [53] (or equivalent) specification.

Geometry	$\|\|u_{PINN} - u_{FEM}\|\|$	$\|\|u_{IBN} - u_{FEM}\|\|$	Training time PINN (s)	Training time IBN (s)
2032C	0.1638	0.0158	3667.83	329.35
A18	0.1745	0.0156	3893.29	286.41
NACA 0012	0.1483	0.0158	3890.78	245.64
NACA 0010	0.1536	0.0157	2031.78	279.98
NACA 0021	0.1471	0.0160	3666.13	244.36

which suggests that the generalization error will ultimately be dominated by the discretization error determined by the resolution h and the order of the FEM basis function α (here $\alpha = 1$), both of which rely on the definition of basis functions $\{\mathcal{B}_i(\underline{x})\}_{i=1}^N$. If h is chosen such that $h = \mathcal{O}(\frac{1}{\sqrt{KN}})$ (justified by [54]), the following corollary can be obtained:

Corollary 7. *Suppose that* $h = \mathcal{O}(\frac{1}{\sqrt{KN}})$*. Given all conditions and parameters from Theorem 6, the generalization error achieves an overall sublinear convergence rate*

$$\|e_G^K\|^2 \leq \mathcal{O}\left(N^2\rho_1^K + \frac{1}{(NK)^\alpha}\right). \tag{33}$$

where ρ_1 *is a constant smaller than 1.*

In summary, our upper bound on the generalization error can be separated into two terms, both of which converge to zero as the number of training epochs K become large—provided the discretization is also chosen inversely with K. Qualitatively, the above bound provides an initial estimate to set the discretization of the mesh, h, based on available computation time (measured in the number of epochs K) and the desired error level e_G. From a practical perspective, the bound allows the estimation of resource requirements for a desired discretization and error level.

7. Experimental results

In this section, we provide results from IBN. We highlight the single instance field solutions for Poisson's and Navier–Stokes equations and

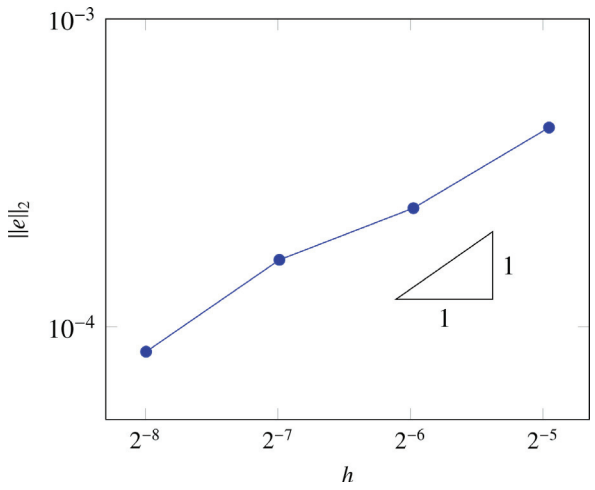


Fig. 5. Error convergence with linear basis functions for the Poisson's equation solved on a circular disk.

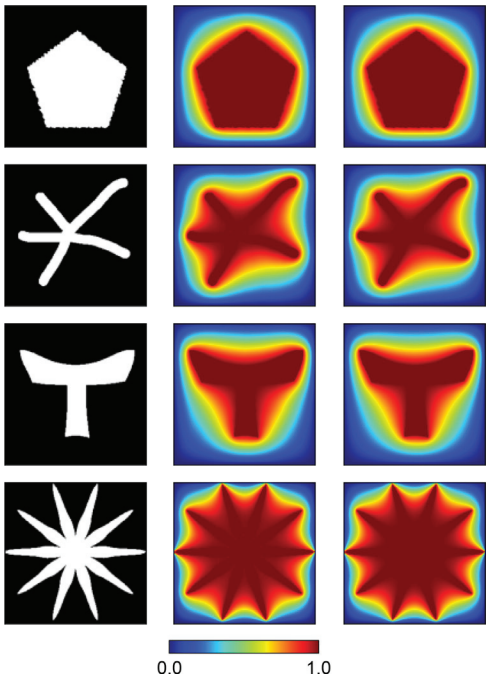


Fig. 6. IBN trained on dataset 'A'. The left shows the heat source in white, the middle shows the IBN-predicted temperature distribution, and the right shows the fully converged numerical solution.

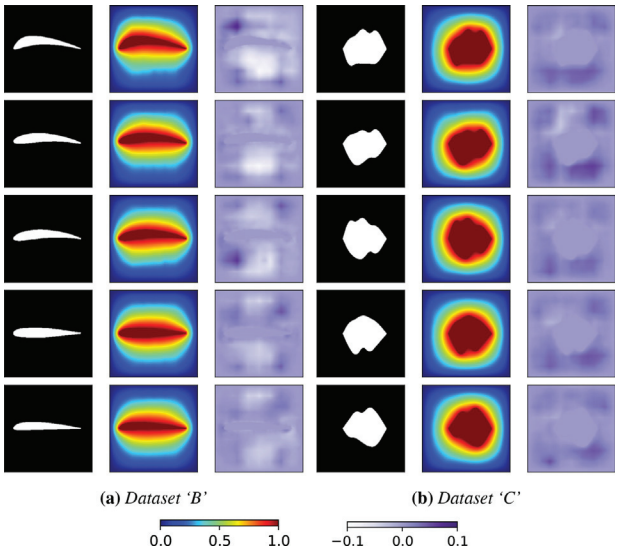


Fig. 7. Selected test examples (Case-1): IBN is trained on dataset A, and tested on (a) Dataset B and (b) Dataset C. Each set comprises of three columns: left columns shows the heat source in white, the middle shows the IBN-predicted solution, and the right column shows errors with respect to a solution obtained from the classical finite element method.

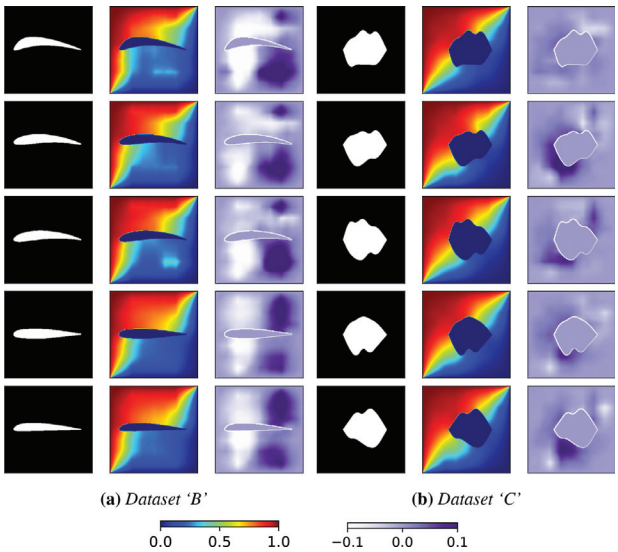


Fig. 8. Selected test examples (Case-2): IBN is trained on dataset A, and tested on (a) Dataset B and (b) Dataset C. Each set comprises of three columns: left columns shows the heat source in white, the middle shows the IBN-predicted solution, and the right column shows errors with respect to a solution obtained from the classical finite element method.

parametric field solutions to Poisson's equation for multiple families of shapes. As previously mentioned, the IBN framework can map geometry representation to the field solution of a given PDE. While the geometry representation could be an unstructured point cloud or other structured representations such as NURBS control points, the PDE field solution is represented as a uniform grid. We use a single Nvidia Titan RTX GPU in all the experiments shown below. Additionally, we provide experimental results which support our error analysis outlined in the previous section.

7.1. Dataset

As discussed in Section 4.2.4, in actual computation task, we typically have access to S_t that contains a collection of different object shapes. Note that unlike typical (supervised) machine learning methods our datasets contain only the input shapes, and do not contain the resulting field solution (i.e ground truth values for training). In the sequel, we illustrate our approach with results over 2D and 3D object geometries. These datasets are discussed below.

In 2D, we consider three different families of complex geometries: the training dataset (A) consists of 1464 irregularly shaped objects with an (unknown) underlying parameterization (Fig. 6). We use the trained model on two test datasets: one of them (B) consists of a parametrizable family of 100 NACA airfoils [53] (Fig. 7, 8), and the other (C) consists of 100 objects parameterized by NURBS curves (Fig. 7, 8). The proposed IBN is trained with the training dataset (A) and tested on datasets (B) and (C). In 3D, we utilize another dataset that is comprised of topology

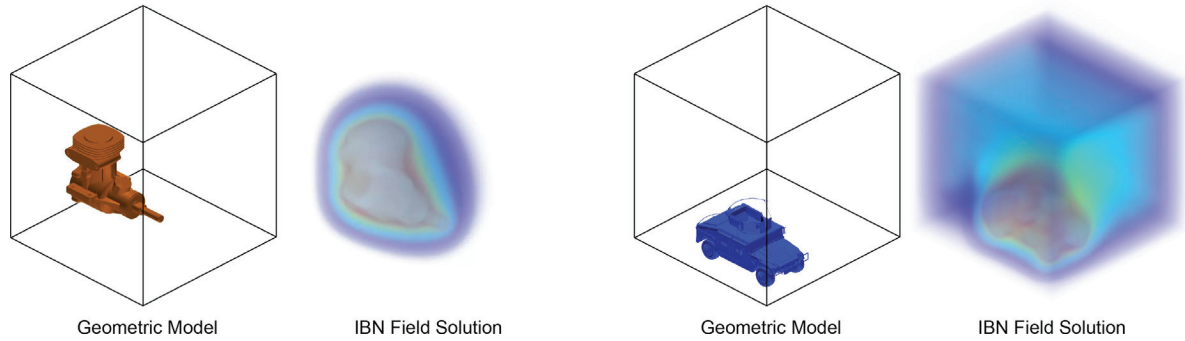


Fig. 9. Geometric model with the domain (left) and the field solution to Poisson's equation using the IBN framework (right) for the Engine and the Humvee models.

optimized structural shapes (size 128^3); we denote this set as Dataset 'D' (see Fig. 11). A summary of all these datasets is given in Table 2.

7.2. Solution to Poisson's equation in 2D and 3D

We perform a convergence analysis and comparisons of the 2D Poisson's equation first. We then demonstrate our method with results for Poisson's equation in 2D and 3D. Collectively, there are three different groups of results; the 2D parametric case, the 3D parametric case, and the final group is two different single-instance solutions over extremely complex geometries in mega-voxel domains.

7.2.1. Convergence analysis

To check the accuracy of our method, we solve Poisson's equation Eq. (2) on a circular disk of radius $R = 0.25$ immersed in a unit square. The forcing function $f = 1$ on the disk and the boundary condition is specified as $u = 0$ at the perimeter of the disk. Therefore, $\Omega_o = \{r < R\}$ and $\Gamma_o = \{r = R\}$. The exact solution to this problem is given by $u(r) = \frac{1}{4}\left(R^2 - r^2\right)$, where $r = \sqrt{x^2 + y^2}$ is the radial position of a point on the disk.

We discretize the unit square domain using an $N \times N$ grid, and the object boundary Γ_o is represented using a point cloud. We optimize the loss function mentioned in Section 4 to obtain the solution. Fig. 4 shows the contours of the discrete solution u^h, the exact solution u and the error $e = \|u^h - u\|$, for $N = 128$. Fig. 5 shows the L^2-norm of the error with mesh size h in a log–log plot. We see first-order convergence with increasing resolution, which matches established theory from IBM analysis ([55] (Lemma 37.2) and [56]).

7.2.2. Comparisons

We compare IBN to standard FE-based result for Poisson's equation in 2D and compare it with one of the state-of-the-art neural methods, PINNs [8] or Physics Informed Neural Networks. Table 3 shows comparison of IBN to numerical solution obtained using traditional approaches [29–32] on five common airfoil shapes taken from the 2D test set 'B'. Similarly, we show these comparisons for PINNs as well. Our approach is consistently more accurate than PINNs in predicting field solutions. Also, the training time for PINNs is much higher than the training time for IBN, with our approach almost 10× faster. Since PINNs can deal with only one geometry at a time, we show similar training behavior, but our approach can generalize to more than one geometry.

7.2.3. 2D results of parametric shapes

We train IBN over the training dataset (A) and evaluate the trained model on the unseen test datasets (B) and (C), respectively.

Case 1—Dirichlet Boundary Condition on the object: The boundary conditions in this case are given by:

$$u = 1 \quad \text{on} \quad \Gamma_o, \tag{34a}$$

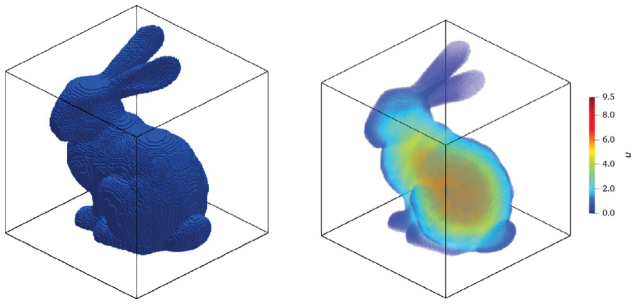


Fig. 10. (Left) the "Stanford bunny" [57] placed within the background domain, (right) a non-parametric Poisson equation is solved on the bunny using IBN. The forcing $f = 500$, and the boundary condition is given by $u = 0$ on the surface of the bunny. This is an example where the solution is sought *inside* the object rather than outside.

$$u = 0 \quad \text{on} \quad \Gamma_B, \tag{34b}$$

where $\Gamma_B = \{x = 0\} \cup \{x = 1\} \cup \{y = 0\} \cup \{y = 1\}$. This case mimics a scenario where the object (Ω_o) acts as a source (of heat or mass); and the Poisson problem describes the steady-state diffusion of heat or mass from the source to the outer boundary. As mentioned above, this problem is trained on dataset A, and the trained model is then tested on 100 samples from datasets B and C each. Fig. 7(a)–7(b) show some anecdotal examples from datasets B and C. The query results are compared against classical finite element methods, and the errors are reported in Table 4 (denoted as 'Case-1').

Case 2—Neumann Boundary Condition on the object: The boundary conditions in this case are given by:

$$\nu \underline{\nabla} u \cdot \underline{\hat{n}} = 0 \quad \text{on} \quad \Gamma_o, \tag{35a}$$

$$u = 1 \quad \text{on} \quad \{x = 0\} \cup \{y = 1\} \tag{35b}$$

$$u = 0 \quad \text{on} \quad \{x = 1\} \cup \{y = 0\}. \tag{35c}$$

This case mimics a scenario where the left and top (outer) boundary act as the source, whereas the right and bottom boundaries act as sink. The object Γ_o acts as an insulator. Thus, in steady state, a flux is established between the source and the sink and the Poisson problem describes the steady-state diffusion of heat or mass from the source to the outer boundary. To apply the zero-Neumann condition, we assign a zero-diffusivity to the object, i.e., we set $\nu = 0$ in Ω_o. This ensures that the solution does not flow to the object, thereby ensuring the zero-flux condition. We note here that the non-zero flux condition is more involved and is not treated in this work.

Once again, this problem is optimized on the training dataset A, and tested on datasets B and C. Some anecdotal examples are presented in Fig. 8(a)–8(b). The query results are compared against classical finite element methods, and the errors are reported in Table 4 (denoted as 'Case-2').

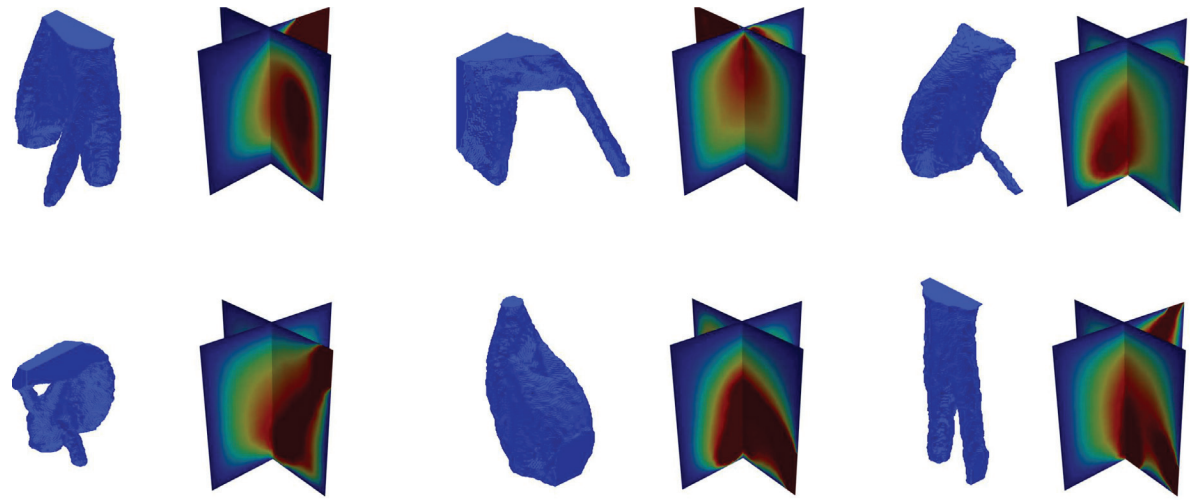


Fig. 11. A family of 3D shapes and their corresponding field solutions. The predicted solutions match the conventionally computed fields.

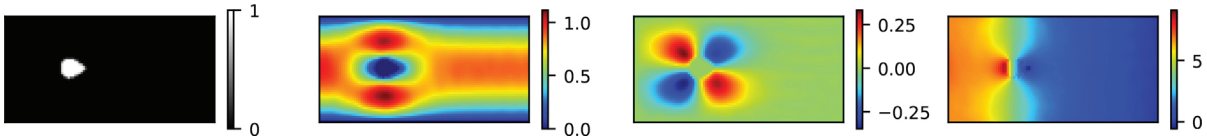


Fig. 12. Immersed method with object mask applied to solve the Navier–Stokes equation for a steady flow past a NACA 0012 aerofoil. (left) domain/boundary mask (middle left) x velocity, (middle right) y velocity, and (right) pressure.

Table 4
Mean of errors calculated on the test datasets B, C, and D.

	$\mathbb{E}(\eta)$		
	Dataset-B	Dataset-C	Dataset-D
Case-1	4.38×10^{-2}	3.14×10^{-2}	–
Case-2	8.13×10^{-2}	3.23×10^{-2}	–
Case-3	–	–	9.28×10^{-2}

Note: $\eta = \frac{\|u_{IBN}-u_{FEM}\|}{\|u_{FEM}\|}$.

7.2.4. 3D results of non-parametric shapes

We demonstrate the capability of the IBN framework to scale to mega-voxel domains (number of voxels = 256^3) with single-instance examples. We provide results for two different cases—an engine, and a Humvee model, which are both publicly available (Fig. 9). This case is a 3D analogue of 'Case-1' in 2D (discussed above), i.e., the object acts as a constant source ($u = 1$ on Γ_o), the outer boundary as a sink ($u = 0$ on Γ_B), and a heat flow occurs in Ω. On the other hand, Fig. 10 presents a similar case, *except* that the Poisson equation is solved *inside* the object, i.e., the computational domain is now Ω_o itself ($f = 1$ in Ω_o, and $u = 0$ on Γ_o). This shows that IBN can be used to solve PDEs both inside and outside the irregular object.

7.2.5. 3D results of parametric shapes

We will refer to this case as 'Case-3'. This case is essentially a 3D analogue of 'Case-1' ($u = 1$ on Γ_o, and $u = 0$ on Γ_B). As outlined Section 7.1, dataset 'D' is family of complex 3D shapes obtained through topology optimization [58]. These shapes are defined as 128^3 binary voxel grids with unique structural characteristics. In Fig. 11, the mesh representation of the heat source is shown next to the predicted field solution of Poisson's equation. The query results are compared against classical finite element methods, and the errors are reported in Table 4. In line with the 2D results presented, the IBN framework is capable of scaling to larger and equally complex families of geometries.

7.3. Navier–Stokes equation

We also validate our framework against a canonical flow past an airfoil using the steady Navier–Stokes equations (NSE) introduced in Eq. (3a). The boundary conditions for this problem are:

$$x = 0: \quad u_x = 1 - \left(\frac{2y}{H} - 1\right)^2, \quad u_y = 0 \tag{36a}$$

$$y = 0: \quad u_x = 0, u_y = 0 \tag{36b}$$

$$y = H: \quad u_x = 0, u_y = 0 \tag{36c}$$

$$(x, y) \in \Gamma_o: \quad u_x = 0, u_y = 0. \tag{36d}$$

The flow field output from IBN shows the expected wake structure (at Reynolds number 40), is symmetric about the mid-plane, shows the stagnant pressure point on the upwind side of the immersed object, and satisfies the imposed no-slip condition. Fig. 12 shows the x velocity, y velocity, and the pressure solution for the Navier Stokes equation using IBN for an aerofoil.

8. Limitations and associated future work

We identify certain limitations of our current work that also suggests possible avenues for future work.

The winding number computation has a large memory footprint and is susceptible to noise in the point cloud data. We are currently exploring algorithmic as well as mathematical approaches to overcome this limitation. Another limitation is discretizing any arbitrary geometry into its pixelated counterpart. This limitation is rather simple to alleviate using additional terms to the loss function along the recently developed shifted boundary method [7]. Our numerical exploration indicates that the current loss function performs well—i.e., generalizes well to unseen geometries—for self-adjoint operators (like Poisson) as opposed to non-self-adjoin operators (like Navier–Stokes). We are exploring several avenues to resolve this challenge, including designing loss functions and co-designing architectures.

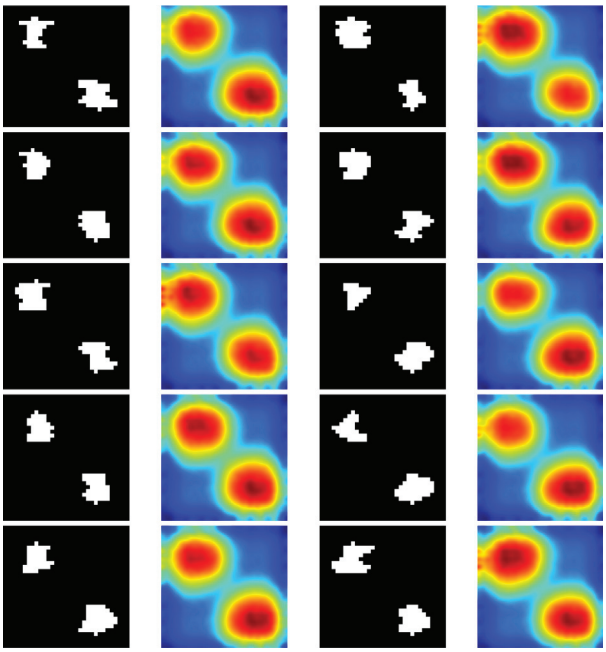


Fig. 13. Failure cases where the solution is not generalized to multiple geometries, especially in thin regions close to the edges of the domain.

Our numerical experiments in Section 7.2.3 show that IBN performs better when a Dirichlet boundary condition is specified on the object boundary Γ_o, and relatively worse when Neumann boundary conditions are specified. Also, applying non-zero Neumann boundary conditions on the irregular object, especially in a parametric setting, is non-trivial, and we leave it for future work.

We also note that the unseen datasets B and C are anecdotal, and the same network may perform poorly on some other test datasets. Two issues come into play here: first, the neural mapping will depend on how good the training dataset is, and second, the nature of the problem may dictate the actual optimization process (e.g., Neumann conditions on pixelated geometry may be susceptible to more errors). We empirically demonstrate a failure case of IBN (Fig. 13), where the network is trained on dataset B and tested on some unseen samples. These unseen samples contain two disjoint objects (recall that the training dataset only contains a single object). The inference results exhibit a violation of the maximum principle [59], and thus, are incorrect.

Finally, this work can be seen as a stepping stone toward a more generalized and efficient neural solvers that can hopefully replace traditional solvers in future. Faster convergence of the training problem, the interplay between the type of the equation and the related design of neural networks, and the incorporation of more advanced ideas from numerical analysis—these are some of the most fundamental problems that still require a substantial attention in future.

9. Conclusions

We have developed a neural PDE solver (termed IBN) that can handle irregularly shaped domains by building on well-established finite element methods and ideas from the immersed boundary methods. IBN can be described as an operator that can take the geometry indicator function (for an irregular or complex domain) as input and predict field solutions over that domain. In addition, IBN is a "data-free" (or "output data-free"); only input training examples are required in the training process, but no target solution values are needed. We highlight two specific PDE cases, Poisson's and Navier–Stokes, which show promising results. Alongside the empirical results, we have included theoretical results for the error bounds of the optimization process of our finite element-based loss function. Since IBN does not require data pairs for training, it can benefit applications where generating target solutions (i.e., data pairs) is expensive. In applications where a large number of expensive inferences are required, e.g., fast design exploration, topology optimization, etc., IBN may provide an alternative.

CRediT authorship contribution statement

Biswajit Khara: Writing – original draft, Visualization, Validation, Software, Formal analysis, Conceptualization. **Ethan Herron:** Writing – original draft, Visualization, Software, Formal analysis, Data curation. **Aditya Balu:** Writing – review & editing, Supervision, Methodology, Conceptualization. **Dhruv Gamdha:** Visualization, Validation. **Chih-Hsuan Yang:** Visualization, Software. **Kumar Saurabh:** Writing – review & editing, Software. **Anushrut Jignasu:** Visualization. **Zhanhong Jiang:** Writing – original draft, Formal analysis. **Soumik Sarkar:** Writing – review & editing, Supervision, Conceptualization. **Chinmay Hegde:** Writing – review & editing, Supervision, Formal analysis, Conceptualization. **Baskar Ganapathysubramanian:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **Adarsh Krishnamurthy:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported in part by the National Science Foundation under grants CCF-2005804, LEAP-HI-2053760, OAC-1750865, DMREF-2323715/2323716, CPS-FRONTIER-1954556, and USDA-NIFA-2021-67021-35329. We would like to thank NVIDIA Corp. for their donation of GPUs for academic research.

Appendix A. Creation of dataset C

The third dataset ('C') is a composition of point clouds and corresponding normals, which outline Non-Uniform Rational B-Splines (NURBS) curves to represent the boundaries of irregular domains. NURBS curves are presented by a set of control points, with each control point being described by cartesian coordinates, in this case only (x,y). For this dataset, we selected points along the x-axis, which were uniformly spaced from 0 to 1.

The corresponding coordinates along the y-axis were randomly sampled from a uniform distribution with a minimum value of 0.2 and a maximum value of 0.8. To attain the point cloud on the boundary defined by the NURBS curve, we utilized NURBS-Python [60], a geometric modeling library. NURBS-Python provides a point cloud on the boundary of the NURBS curve and the normals, unique vectors for each point in the point cloud pointing in the orthogonal direction with respect to the boundary. Additionally, the area for each point is required for calculating the winding number. In this work, we assume each point has a uniform area, which we maintain for each irregular boundary in the entire dataset.

Appendix B. Network architecture

The primary architecture of G_{ibn} (see Eq. (10)) in this paper is given by U-Net [49,61]. U-Nets have been known to be effective for applications such as semantic segmentation and image reconstruction. The architecture of the network is shown in Fig. B.14. First, a block of convolution and instance normalization is applied. Then, the output is saved for later use via skip connection. This intermediate output is then down-sampled at a lower resolution for a subsequent block of convolution, instance normalization layers. This process is repeated two times. Now, the upsampling starts where the saved outputs of similar dimensions are concatenated with the output of upsampling for creating the skip connections, followed by a convolution layer. LeakyReLU activation was used for all the intermediate layers, with Sigmoid activation for the final layer. The exact architecture can be found in the accompanying code, which is openly available on GitHub. We would like to note that the IBN method is not specifically dependent on the U-Net architecture, and there is scope for exploring other architectures.

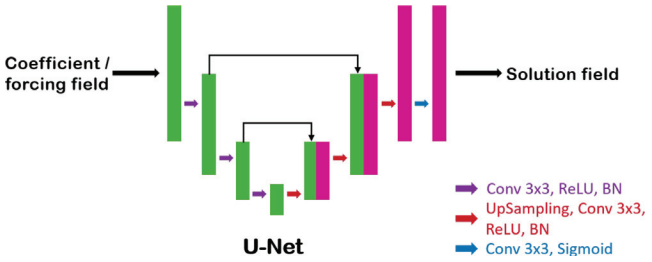


Fig. B.14. UNet architecture used for training IBN.

References

[1] Hughes TJ. The finite element method: linear static and dynamic finite element analysis. Courier Corporation; 2012.
[2] LeVeque RJ. Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems. SIAM; 2007.
[3] Trefethen LN. Spectral methods in MATLAB. SIAM; 2000.
[4] Slotnick J, Khodadoust A, Alonso J, Darmofal D, Gropp W, Lurie E, et al. CFD vision 2030 study: A path to revolutionary computational aerosciences. In: 54th AIAA aerospace sciences meeting. 2014, p. 12.
[5] Peskin CS. The immersed boundary method. Acta Numer 2002;11:479–517.
[6] Mittal R, Iaccarino G. Immersed boundary methods. Annu Rev Fluid Mech 2005;37:239–61.
[7] Main A, Scovazzi G. The shifted boundary method for embedded domain computations. Part I: Poisson and Stokes problems. J Comput Phys 2018;372:972–95.
[8] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 2019;378:686–707.
[9] Lagaris IE, Likas A, Fotiadis DI. Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 1998;9(5):987–1000.
[10] Bartlett PL, Foster DJ, Telgarsky MJ. Spectrally-normalized margin bounds for neural networks. Adv Neural Inf Process Syst 2017;30.
[11] Babuška I, Rheinboldt WC. A-posteriori error estimates for the finite element method. Internat J Numer Methods Engrg 1978;12(10):1597–615.
[12] Kharazmi E, Zhang Z, Karniadakis GE. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. Comput Methods Appl Mech Engrg 2021;374:113547.
[13] Sirignano J, Spiliopoulos K. DGM: A deep learning algorithm for solving partial differential equations. J Comput Phys 2018;375:1339–64.
[14] Yang L, Zhang D, Karniadakis GE. Physics-informed generative adversarial networks for stochastic differential equations. SIAM J Sci Comput 2020;42(1):A292–317.
[15] Pang G, Lu L, Karniadakis GE. fPINNs: Fractional physics-informed neural networks. SIAM J Sci Comput 2019;41(4):A2603–26.
[16] Karumuri S, Tripathy R, Bilionis I, Panchal J. Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks. J Comput Phys 2020;404:109120.
[17] Han J, Jentzen A, Weinan E. Solving high-dimensional partial differential equations using deep learning. Proc Natl Acad Sci 2018;115(34):8505–10.
[18] Michoski C, Milosavljević M, Oliver T, Hatch DR. Solving differential equations using deep neural networks. Neurocomputing 2020;399:193–212.
[19] Samaniego E, Anitescu C, Goswami S, Nguyen-Thanh VM, Guo H, Hamdia K, et al. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. Comput Methods Appl Mech Engrg 2020;362:112790.
[20] Ramabathiran AA, Ramachandran P. SPINN: Sparse, physics-based, and partially interpretable neural networks for PDEs. J Comput Phys 2021;445:110600.
[21] Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat Mach Intell 2021;3(3):218–29.
[22] Botelho S, Joshi A, Khara B, Sarkar S, Hegde C, Adavani S, et al. Deep generative models that solve PDEs: Distributed computing for training large data-free models. In: 2020 IEEE/ACM workshop on machine learning in high performance computing environments (MLHPC) and workshop on artificial intelligence and machine learning for scientific applications (AI4S). IEEE Computer Society; 2020, p. 50–63.
[23] Balu A, Botelho S, Khara B, Rao V, Sarkar S, Hegde C, et al. Distributed multigrid neural solver on megavoxel domains. In: SC '21: proceedings of the international conference for high performance computing, networking, storage and analysis, vol. 49. 2021, p. 1–12.
[24] Wandel N, Weinmann M, Neidlin M, Klein R. Spline-PINN: Approaching PDEs without data using fast, physics-informed Hermite-spline CNNs. In: Proceedings of the AAAI conference on artificial intelligence, vol. 36. 2022, p. 8529–38.
[25] Lagari PL, Tsoukalas LH, Safarkhani S, Lagaris IE. Systematic construction of neural forms for solving partial differential equations inside rectangular domains, subject to initial, boundary and interface conditions. Int J Artif Intell Tools 2020;29(05):2050009.
[26] Sukumar N, Srivastava A. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. Comput Methods Appl Mech Engrg 2022;389:114333.
[27] Zhang L, Gerstenberger A, Wang X, Liu WK. Immersed finite element method. Comput Methods Appl Mech Engrg 2004;193(21–22):2051–67.
[28] Xu F, Schillinger D, Kamensky D, Varduhn V, Wang C, Hsu M-C. The tetrahedral finite cell method for fluids: Immersogeometric analysis of turbulent flow around complex geometries. Comput & Fluids 2016;141:135–54.
[29] Saurabh K, Gao B, Fernando M, Xu S, Khanwale MA, Khara B, et al. Industrial scale Large Eddy Simulations with adaptive octree meshes using immersogeometric analysis. Comput Math Appl 2021;97:28–44.
[30] Bangerth W, Hartmann R, Kanschat G. deal. II—a general-purpose object-oriented finite element library. ACM Trans Math Softw 2007;33(4):24–es.
[31] Griffith BE, Hornung RD, McQueen DM, Peskin CS. An adaptive, formally second order accurate version of the immersed boundary method. J Comput Phys 2007;223(1):10–49.
[32] Egan R, Guittet A, Temprano-Coleto F, Isaac T, Peaudecerf FJ, Landel JR, et al. Direct numerical simulation of incompressible flows on parallel octree grids. J Comput Phys 2021;428:110084.
[33] Saurabh K, Ishii M, Fernando M, Gao B, Tan K, Hsu M-C, et al. Scalable adaptive PDE solvers in arbitrary domains. In: Proceedings of the international conference for high performance computing, networking, storage and analysis. 2021, p. 1–15.
[34] Tan K, Gao B, Yang C-H, Johnson EL, Hsu M-C, Passalacqua A, et al. A computational framework for transmission risk assessment of aerosolized particles in classrooms. Eng Comput 2023;1–22.
[35] Gao H, Sun L, Wang J-X. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. J Comput Phys 2021;428:110079.
[36] McFall KS, Mahan JR. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. IEEE Trans Neural Netw 2009;20(8):1221–33.
[37] Sheng H, Yang C. PFNN: A penalty-free neural network method for solving a class of second-order boundary-value problems on complex geometries. J Comput Phys 2021;428:110085.
[38] Berg J, Nyström K. A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing 2018;317:28–41.
[39] Han X, Yang Y, Liu Y. Determining the defect locations and sizes in elastic plates by using the artificial neural network and boundary element method. Eng Anal Bound Elem 2022;139:232–45.
[40] Sun J, Liu Y, Yao Z, Zheng X. A data-driven multi-flaw detection strategy based on deep learning and boundary element method. Comput Mech 2023;71(3):517–42.
[41] Sun J, Liu Y, Wang Y, Yao Z, Zheng X. BINN: A deep learning approach for computational mechanics problems based on boundary integral equations. Comput Methods Appl Mech Engrg 2023;410:116012.
[42] Sahli Costabal F, Pezzuto S, Perdikaris P. Δ-PINNs: Physics-informed neural networks on complex geometries. Eng Appl Artif Intell 2024;127:107324.

[43] E. W, Yu B. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun Math Stat 2018;6(1):1–12.
[44] Wang S, Teng Y, Perdikaris P. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J Sci Comput 2021;43(5):A3055–81.
[45] Wang S, Yu X, Perdikaris P. When and why PINNs fail to train: A neural tangent kernel perspective. J Comput Phys 2022;449:110768.
[46] Krishnapriyan A, Gholami A, Zhe S, Kirby R, Mahoney MW. Characterizing possible failure modes in physics-informed neural networks. Adv Neural Inf Process Syst 2021;34:26548–60.
[47] Kamensky D, Hsu M-C, Schillinger D, Evans JA, Aggarwal A, Bazilevs Y, et al. An immersogeometric variational framework for fluid–structure interaction: Application to bioprosthetic heart valves. Comput Methods Appl Mech Engrg 2015;284:1005–53.
[48] Hsu M-C, Wang C, Xu F, Herrema AJ, Krishnamurthy A. Direct immersogeometric fluid flow analysis using B-rep CAD models. Comput Aided Geom Design 2016;43:143–58.
[49] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. In: International conference on medical image computing and computer-assisted intervention. Springer; 2015, p. 234–41.
[50] Barill G, Dickson NG, Schmidt R, Levin DI, Jacobson A. Fast winding numbers for soups and clouds. ACM Trans Graph 2018;37(4):1–12.
[51] Barron AR. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans Inf Theory 1993;39(3):930–45.
[52] Virmaux A, Scaman K. Lipschitz regularity of deep neural networks: analysis and efficient estimation. Adv Neural Inf Process Syst 2018;31.
[53] Allen B. NACA Airfoils. NASA; 2017, URL https://www.nasa.gov/image-feature/langley/100/naca-airfoils.
[54] Larson MG, Bengzon F. The finite element method: theory, implementation, and applications, vol. 10, Springer Science & Business Media; 2013.
[55] Ern A, Guermond J-L. Finite Elements II: Galerkin approximation, elliptic and mixed PDEs, vol. 73, Springer Nature; 2021.
[56] Schillinger D, Harari I, Hsu M-C, Kamensky D, Stoter SK, Yu Y, et al. The non-symmetric nitsche method for the parameter-free imposition of weak boundary and coupling conditions in immersed finite elements. Comput Methods Appl Mech Engrg 2016;309:625–52.
[57] Lindstrom P, Turk G. Fast and memory efficient polygonal simplification. In: Proceedings visualization'98 (cat. no. 98CB36276). IEEE; 1998, p. 279–86.
[58] Rade J, Balu A, Herron E, Pathak J, Ranade R, Sarkar S, et al. Algorithmically-consistent deep learning frameworks for structural topology optimization. Eng Appl Artif Intell 2021;106:104483.
[59] Evans LC. Partial differential equations. Graduate studies in mathematics 1998;19(4):7.
[60] Bingol OR, Krishnamurthy A. NURBS-Python: An open-source object-oriented NURBS modeling framework in Python. SoftwareX 2019;9:85–94.
[61] Çiçek Ö, Abdulkadir A, Lienkamp SS, Brox T, Ronneberger O. 3D U-Net: learning dense volumetric segmentation from sparse annotation. In: International conference on medical image computing and computer-assisted intervention. Springer; 2016, p. 424–32.