

ARPA-E DIFFERENTIATE Award DE-AR0001215: **Final Report**

Performance Period: May 2019 to May 2022

Investigators: Baskar Ganapathysubramanian (PI, ISU)
Soumik Sarkar (ISU) Adarsh Krishnamurthy (ISU)
Chinmay Hegde (NYU) Zhenan Bao (Stanford)
Ross Larsen (NREL) Bryon Larson (NREL)

The information, data, or work presented herein was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number [DE-AR0001215]. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

*This Report contains no Protected Data.

Contents

1	Executive Summary	2
2	Summary of Milestones	3
3	Supporting Data and Additional Information	4
4	Publications Summary	16
5	Budget Summary	17

1 Executive Summary

This document describes progress in the ARPA-E DIFFERENTIATE project titled “*Context-Aware Learning for Inverse Design in Photovoltaics*” during the period of May 2019 to May 2022. This project is being performed at Iowa State University, New York University, Stanford University, and National Renewable Energy Laboratory. The project aims to develop a new machine learning (ML) framework to significantly accelerate the design of organic microstructures for improved organic photovoltaic performance.

In this project, we had developed an inverse design framework using Deep Learning called InvNets for generating microstructures with desired physics-driven properties. As a preliminary product, in Milestone 3, we demonstrated how InvNets show 20% improvement in the performance of the microstructures and over 100X speedup in the performance compared to traditional processes for physics-driven inverse design. Later, in Milestone 6, we demonstrated that InvNets work for more complex physics properties, specifically, generating microstructures for organic photovoltaic cells with desired current-voltage characteristics.

Further, in Milestone 4, we explored the idea of using physics-aware surrogates for obtaining solutions of partial differential equations(PDE) called as DiffNets(now called as NeuFENets to avoid ambiguity of names). The connection between both frameworks is that DiffNet surrogates form the physics-aware surrogate in the InvNet framework.

Finally in Milestone 8, we extend our framework for other physics domains. Specifically, we explore building geometry-aware NeuFENets by developing physics surrogates that exploit ideas from traditional immersed boundary finite element methods.

With these updates, we are able to achieve all the Milestones. More details can be found below.

2 Summary of Milestones

M8.1 Applications to solve other PDE based inverse designs

Milestone Description

Demonstrated applicability of InvNets to solve PDE based inverse designs in other disciplines.

Milestone Update

In this milestone, we extend our PDE solver to different physics on complex geometries to show applicability of our framework, NeuFENet, on different domains. We have completed this milestone in this quarter and have successfully completed all the technical milestones of this project.

M8.2 Release of software implementations

Milestone Description

Released open-source software implementations of tools/algorithms

Milestone Update

We have open-sourced all the softwares that came as a product of this project. The list is provided below.

M8.3 Final iteration of U.S. Commercialization and Formal Reporting of the Software Developed

Milestone Description

Final iteration of U.S. Commercialization and Formal Reporting of the Software Developed

Milestone Update

We applied T2M plus-up, which should satisfy the requirements of the commercialization of the software we have developed.

3 Supporting Data and Additional Information

Status of Ongoing Tasks

Task 8.1: Applications to other PDEs and domains

We focus on the problem of training a parametric Neural PDE solver that can account for a large parametrized shape input. This has implications for a variety of shape-design problems. Our approach borrows from techniques in classical PDE solver technology that *immerse* the parametrized shape into a regular bounding box, rather than trying to create a boundary-fitted mesh [27, 35]. This concept translates particularly well to the convolution-heavy neural networks (like our NeuFENet) that are usually deployed using hypercubes (2D rectangles, 3D cuboids).

Motivation Most physical phenomena are modeled using a set of governing partial differential equations (PDEs). Numerical methods—finite difference methods (FDM), finite element methods (FEM), and spectral methods—for solving PDEs discretize the physical domain (into cells, elements, etc.) and *approximate* the solution over this discretized domain using select families of basis functions [7, 14, 32]. A significant part of PDE solver technology involves solving PDEs on complex, irregular domains (for instance, flow across aerofoils in aeronautics, or in patient-specific organ geometries in medical diagnostics). This is a major challenge, as articulated in the NASA CFD 2030 [30] vision (“Mesh generation and adaptivity continue to be significant bottlenecks”). In particular, capturing the complex geometry, as well as rigorously accounting for the non-trivial boundary conditions on these complex geometries has driven careers in mesh generation and PDE solver technology.

Motivated by these challenges, this paper addresses the following problem, “Can we design a neural PDE solver that can produce field solutions across arbitrary geometries?”. We do so by utilizing an analytical approach developed in the computational mechanics community—the immersed boundary method (IBM) [20, 26]. In such approaches, the irregular geometry is ‘immersed’ in a regular grid, thus allowing standard meshes to model irregular geometries. We extend this powerful approach to neural PDE solvers, which enables the creation of PDE solvers that can produce field solutions for a distribution of irregular geometries. We show that such an approach allows the natural incorporation of different boundary conditions over complex geometries; as a side benefit, our approach also allows us to compute *a priori* error estimates using a combination of techniques from neural net generalization and finite element analysis.

Neural PDE Solvers Since neural networks are powerful nonlinear function approximators, there has been a growing interest in using neural networks to solve PDEs [23, 11, 29, 37, 19, 9, 5, 17, 25, 24, 16, 3, 1, 34]. Unlike numerical methods, many of these methods do not require a

mesh. But a common challenge faced by most neural PDE solvers is the imposition of boundary conditions. In the physics-informed neural network (PINN) family of methods, both the Dirichlet and Neumann conditions are imposed approximately. Furthermore, among collocation-based neural solvers, the satisfaction of certain regularity conditions becomes non-trivial to impose [31]. In addition, the solution of the PDE on irregularly shaped domains becomes challenging using PINN-based approaches.

Immersed Approach Classical numerical methods such as finite difference methods (FDM) and finite element methods (FEM) generally employ a grid or mesh to discretize the domain geometry and function space. While FDM is most commonly applied using a regular grid, FEM allows for unstructured grids, thus accommodating many complex geometries. Thus solving PDEs defined on complex geometries using FEM requires a mesh to be prepared before the analysis. This step, commonly known as the “mesh generation” step, is non-trivial and often expensive. Thus, both FDM and FEM-based methods face challenges for complex irregularly-shaped domains, but for different reasons. Immersed method [20, 18] is one way to overcome this challenge. The computational grid is simplified in immersed methods by considering a rectilinear axis-aligned grid. The irregularly-shaped geometry is considered “immersed” in this background mesh (see Fig. 3.1). Thus while some part of the background mesh forms the interior of the actual computational mesh, the rest of the background mesh is considered exterior, and thus not used in the computation.

In this work, we combine the generalization capability of deep neural networks to solve such PDEs defined on irregularly-shaped domains using ideas from immersed FEM. Our approach thus inherits the function approximation properties of an immersed finite element method. In recent years, immersed methods have been favored for parallel implementations since all computations are performed on a regular grid. This fact also applies to tensor-based operations in convolutional neural networks, which otherwise are unsuitable for dealing with 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. Our main contributions are as follows:

1. **[Framework]:** We present a parametric 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—Immersed Boundary Network (IBN)—that uses the geometric information as input to predict 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 a distribution of arbitrary shapes.
2. **[Error Analysis]:** We provide analysis of convergence and generalization error bounds of the proposed PDE-based loss function with the immersed approach.
3. **[Applications and Broader Impact]:** 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 and Navier-Stokes—and a host of irregular geometries. IBN opens up fast design exploration and topology optimization for various societally critical applications such as room ventilation for reduced disease risk, shape design for energy harvesters, and aerodynamic design of vehicles.

Mathematical Preliminaries

Consider a set $\Omega_B \subset \mathbb{R}^d$ ($d \in \{2, 3\}$) with a rectilinear boundary Γ_B ; and another set $\Omega_o \subset \Omega_B$ with an irregular boundary Γ_o . Without loss of generality, the d -dimensional unit interval $[0, 1]^d$

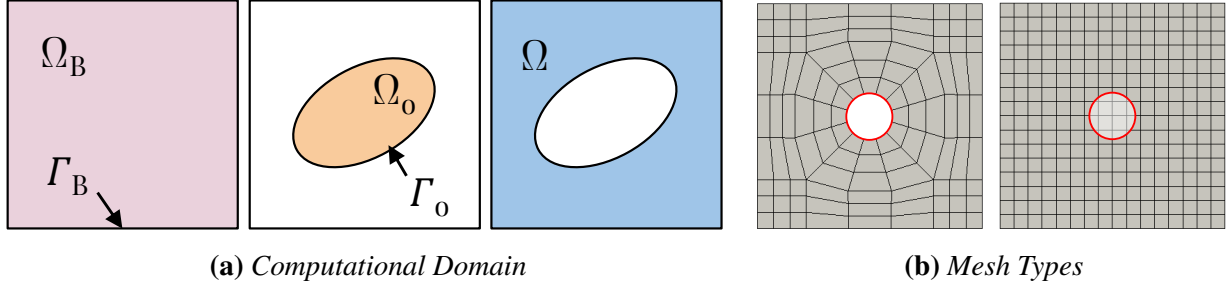


Figure 3.1: (a) Schematic of a typical domain for the background mesh Ω_B (with boundary Γ_B), the embedded/immersed object Ω_o (with boundary Γ_o) and the computational domain Ω . (b) A “body-fitted” mesh that discretely conforms to the object and an object embedded/immersed in a background mesh. This paper focuses on the immersed type meshes with different object geometries.

can be considered an example of Ω_B . Define $\Omega = \Omega_B/\Omega_o$ (see Figure 3.1a for an illustration in 2D Euclidean space). On this set Ω , we now consider an abstract PDE with Dirichlet boundary conditions given by:

$$\mathcal{N}(u) = f(\underline{x}) \text{ in } \Omega, \quad (3.1a)$$

$$u = g \text{ on } \Gamma_o \subset \partial\Omega. \quad (3.1b)$$

Here, u is an unknown function such that $u : \Omega \mapsto \mathbb{R}$, \mathcal{N} is a differential operator (possibly nonlinear) and $f(\underline{x})$ is a known function of the domain variable $\underline{x} \in \Omega$. Note that the overall boundary of Ω can be written as $\partial\Omega = \Gamma = \Gamma_B \cup \Gamma_o$. In this paper, we look at two concrete examples of Equation 3.1, namely the Poisson’s equation and the Navier-Stokes equations.

Poisson’s Equation: Poisson’s equation is frequently used to model steady-state mass/heat diffusion, electrostatics, surface reconstruction, etc. and is given by:

$$-\Delta u = f \text{ in } \Omega, \quad (3.2a)$$

$$u = g \text{ on } \Gamma, \quad (3.2b)$$

where $u : \Omega \rightarrow \mathbb{R}$ is a scalar function that, depending on the underlying physics, represents the mass density, temperature, electric potential, or the surface indicator function, respectively.

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

$$\underline{u} \cdot \nabla \underline{u} - \nu \Delta \underline{u} + \nabla p = \underline{f} \text{ in } \Omega \quad (3.3a)$$

$$\underline{u} = \underline{g} \text{ on } \Gamma \quad (3.3b)$$

where $\underline{u} : \Omega \rightarrow \mathbb{R}^d$ is a vector valued function that represents the velocity field, and $p : \Omega \rightarrow \mathbb{R}$ is a scalar representing the pressure in the fluid. The coefficient ν is a non-dimensional constant known as the viscosity of the fluid.

PDEs such as Equation 3.2 and Equation 3.3 are generally non-trivial to solve analytically except with liberal simplifications. Thus, in engineering contexts, such equations are generally solved numerically using methods such as the finite element method (FEM), finite difference method

(FDM), finite volume method (FVM), or the spectral methods, among many others. The FDM-like methods rely on rectilinear grids; thus, they are unsuitable for problems posed on irregular domains (without significant additional developments). But this limitation can be resolved by using the immersed boundary method ([20, 18]), where the irregular domain is “immersed” in a rectilinear domain with axis-aligned grid (see Figure 3.1b (right)). On the other hand, integral methods such as FEM and FVM allow for unstructured meshes on non-rectilinear geometries. From a practitioner’s perspective, for using FEM or FVM, a computational mesh needs to be generated before starting the computation. This mesh represents the discretization of the function space. Traditionally, the most popular method for this purpose is to generate a “body-fitted” mesh that conforms to the geometry of the domain (see Figure 3.1b). But good meshes are difficult to create automatically (and in parallel) [30] compared to axis-aligned structured meshes (Figure 3.1b (right)). Thus, there has been a growing interest in using immersed methods for FEM/FVM to analyze PDEs posed on non-trivial domains.

Immersed Finite Element Method: In this paper, we formulate our neural PDE solver based on an immersed FEM discretization. In terms of the notations developed above, we discretize Ω_B using an axis-aligned grid (Figure 3.1b (right)) 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, and the boundary conditions on Γ_o are applied approximately (in a weak manner) on the grid points that fall near this curve. The details of how the boundary conditions are applied are discussed in Section 3.

Our Approach: Immersed Boundary Network (IBN)

Model: We will learn an *immersed boundary network* (IBN) that produces a (field) solution to a given PDE and also (rigorously) adheres to the boundary conditions imposed by a family of complex geometries.

Our IBN model is written as a parameterized function $G(x; \theta)$, where θ is a set of tunable network weights, and x represents the geometry (x can either be in the form of a point cloud or a set of NURBS control points representing the surface of a complex geometry Γ_o). Given such an input x , a trained network G should be able to predict both the interior/exterior of the computational domain and the solution to the PDE. Thus, the neural mapping is more precisely defined as $G(x; \theta) : \mathbb{R}^{n \times d} \times \mathbb{R}^{|\theta|} \rightarrow \mathbb{R}^{1+N \times n_{dof}}$, where n is the number of control points used to represent the geometry, d is the spatial dimension, N is the number of points used to discretize the output mesh and 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). Suppose U_θ is the discrete approximation of the solution u (see below for a precise definition of U) and χ a discrete representation of the *occupancy function* that indicates the interior/exterior of the mesh. Then, given a set of surface control points $\{P\}$, we have:

$$[\chi_\theta, U_\theta] = G(\{P\}; \theta), \quad (3.4)$$

where χ_θ and U_θ are represented as d -dimensional tensors.

Numerical Method: In this work, we use FEM to discretely approximate the field solution as well as its spatial derivatives. Let \mathcal{K}^h be a discretization of Ω_B into n_{el} finite elements K_i such that $\cup_{n_{el}} K_i = \Omega_B$. We then define a function space V^h as follows:

$$V^h = \{v^h \in V : v^h|_K \in P_m(K), K \in \mathcal{K}^h\}, \quad (3.5)$$

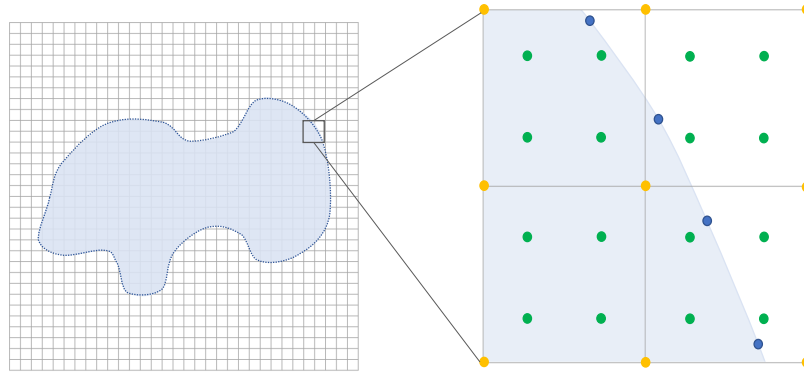
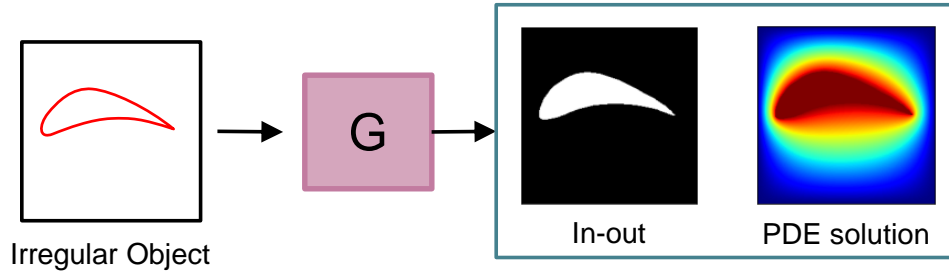


Figure 3.2: (a) The input to the network G is a representation of the object, possibly a point cloud or a NURBS curve/surface. A trained network can then predict the in-out information of the object and the field solution. (b) A visual representation of the irregular object intersecting the element. Each yellow dot denotes the pixels; green dots denote the Gauss points used to interpolate the value of the finite element; blue dots denote points from the object boundary. The shaded region is the area inside the boundary.

where $P_m(K)$ denotes the set of polynomial functions of degree m defined on K . Since we are dealing with an immersed finite element method, we assume that \mathcal{K}^h is composed of a rectilinear axis-aligned grid with N nodes. Now, suppose $\{\phi_i(\underline{x})\}_{i=1}^N$ is a suitable basis that span V^h . Then any function $u^h \in V^h$ can be written as

$$u^h(\underline{x}) = \sum_{i=1}^N \phi_i(\underline{x})U_i, \quad (3.6)$$

where U_i are the function values at the nodal points in the mesh \mathcal{K}^h . 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 $\{U_i\}$, $i = 1, 2, \dots$, (the vector representation). The two will be used interchangeably, assuming that the underlying basis functions $\phi(\underline{x})$ are known.

Training Loss Function: A key novelty of our method lies in the formulation of an appropriate PDE loss function for irregular geometries, and its associated numerics. At a high level, our

loss consists of two different parts: one addressing the boundary conditions imposed by (complex, irregular) geometries, and the other addressing the given PDE. Thus we can break down the overall loss function into two main components: (i) predicting the field u^h (PDE Loss), and (ii) predicting the occupancy function χ and exact imposition of boundary conditions (Geometry Loss).

PDE Loss: The PDE component of the loss function is standard and follows directly from the Galerkin formulation of the given PDE. For the abstract PDE in Equation 3.1, the corresponding Galerkin formulation (with boundary conditions weakly satisfied) is to find $u^h \in V^h$ such that

$$(v^h, [\mathcal{N}(u^h) - f])_{\Omega} + \lambda (v^h, [u^h - g^h])_{\Gamma} = 0, \quad \forall v^h \in V^h. \quad (3.7)$$

In general, if an 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 Equation 3.7 nonzero. We call this the residual of Equation 3.7, i.e.,

$$R = (v^h, [\mathcal{N}(w^h) - f])_{\Omega} + \lambda (v^h, [w^h - g^h])_{\Gamma}, \quad (3.8)$$

where, $w^h \in V^h$. Minimizing this residual should provide us with a unique solution. Unfortunately, we cannot perform integrations over Ω in a straightforward manner, but we *can* do so on Ω_B . On the other hand, Ω_B also contains Ω_o but we do not want to perform integrations on Ω_o . We resolve this via the occupancy function χ^h , which helps us perform integrations on Ω_B while at the same time enforcing no contribution from Ω_o to R .

Geometric Loss: This corresponds to exact imposition of boundary conditions on the surface points $p_i \in \{P\} \subset \Gamma_o$ and correct prediction of the occupancy function χ . For given surface points $\{P\}$, we can further extend the definition of $u^h(x)$ on the boundary points as $u_i = \sum_{j=1}^N \phi_j(p) U_j \forall p_i \in \{P\}$. This interpolation performed on the point cloud (separate from the Gauss points, see Figure 3.2(b)) helps us exactly impose the boundary conditions.

Prediction of the occupancy function χ can be obtained using two approaches : (i) computing Winding numbers (ii) solving Eikonal equation as another unknown.

Generalized Winding Number: This is the general form of the winding number which counts the number of loops of the boundary around any point q . The generalized winding number can be calculated as the surface integral of the differential solid angle ($\int_{\Gamma_o} dS(q)$) [2]. Given a boundary representation of an object, we can use the generalized winding number to determine in-out information since any point inside the object boundary will have a winding number 1. The in-out occupancy χ_w is calculated as:

$$\chi_w(\mathbf{q}) = \sum_{i=1}^m a_i \frac{(\mathbf{p}_i - \mathbf{q}) \cdot \hat{\mathbf{n}}_i}{4\pi \|\mathbf{p}_i - \mathbf{q}\|^3} \quad (3.9)$$

where a_i is the Voronoi area of a given point and n_i is the normal for a given point p_i [2]. Using χ_w , we can obtain all nodes that are inside the geometry ($u^h \in \Omega_o$) by using χ_w as a masking function.

Eikonal Equation: Signed distance fields (ϕ) denote the distance of the closest point on the surface Γ_o . The sign of the distance fields can be used as occupancy information ($\chi_e = \text{sign}(\phi)$). The signed distance field is obtained by solving the viscosity stabilized version of the Eikonal equation ($(1 + \tau) \|\nabla \phi\| - \tau (\nabla \phi) = 1$, $\tau \in [0, 0.5]$) [15]. We demonstrate the usage of both χ_e and χ_w for obtaining the mask and weakly applying the boundary conditions.

Combining the PDE Loss and the Geometric Loss, the complete loss function is:

$$\mathcal{J}(w^h) = \underbrace{\|R(w^h)\|_2}_{\text{PDE loss}} + \lambda_1 \underbrace{\sum_{i=1}^n |w^h(x_i) - g_i|^2}_{\text{boundary condition on object}} + \lambda_2 \underbrace{\sum_{j \in \{j|\chi > 0\}} |w^h - g_{\text{in}}|^2}_{\text{exterior points}}, \quad (3.10)$$

where g_i and g_{in} represent suitable functions defined in Γ_o and Ω_o respectively. We assume that g_i conforms to the boundary Γ_o . i.e. the point cloud is not noisy and the boundary conditions are applied accurately. Thus we can then define the minimization problem statement as

$$u^h = \arg \min_{w^h \in V^h} \mathcal{J}(w^h). \quad (3.11)$$

which can be approximately solved using several gradient descent steps. To summarize, the first term in Equation 3.10 represents the loss function derived from the PDE, the second term attempts to “weakly” set the boundary points to their respective prescribed values, and finally, the third term attempts to set the object interior points to a fixed value conforming to the object boundary. Appropriate values for λ_1 and λ_2 are generally problem dependent, but for our experiments we take $\lambda_1 = \lambda_2 = O(1)$. However, in case of a high gradient in the solution values near the boundaries, these coefficients scale $\propto \frac{1}{h}$.

Error Analysis

In this section, we summarize theoretical results for the convergence behavior and the generalization error for IBN. Due to tight space constraints, all detailed proofs are relegated to the supplementary material. A summary of the theoretical results is presented in Table 3.1.

For a given geometry Γ_o , 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 $G(x, \theta)$ is typically an inaccurate version of u_h , and we can express the error e_G as follows:

$$\|e_G^k\| = \|u_{\theta_k} - u\| = \|u_{\theta_k} - u_{\theta^*} + u_{\theta^*} - u^h + u^h - u\| \quad (3.12)$$

Table 3.1: Summary of different generalization bounds.

Setting	Property	Step Size	Rate
Batch	μ -strong convexity	Constant	$\mathcal{O}\left(N^2 \rho_1^K + \frac{1}{(NK)^\alpha}\right)$
Stochastic	μ -PL condition	Constant	$\mathcal{O}\left(N^2 \rho_2^K + N^2 \sigma^2 + \frac{1}{(NK)^\alpha}\right)$
Stochastic	μ -PL condition	Diminishing	$\mathcal{O}\left(\frac{N^2}{K} + \frac{1}{(NK)^\alpha}\right)$

μ : the strongly convex constant; N : the number of basis functions; $\rho_1, \rho_2 \in (0, 1)$; K : the number of epochs; α : the polynomial order of FEM basis functions; PL: Polyak-Lojasiewicz; σ : second moment constant.

where, u_{θ_k} is the field solutions corresponding to the network parameters θ_k at the k^{th} iteration of the optimization and u_{θ^*} is the theoretical optimum (i.e., a limit point) of u_{θ_k} .

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

$$\|e_G^k\|^2 \leq 3 \left(\underbrace{\|u_{\theta_k} - u_{\theta^*}\|^2}_{\text{Optimization error}} + \underbrace{\|u_{\theta^*} - u^h\|^2}_{\text{Modeling error}} + \underbrace{\|u^h - u\|^2}_{\text{Discretization error}} \right). \quad (3.13)$$

Thus the generalization error is bounded above by three different terms, which are respectively the *optimization error*, *modeling error*, and *discretization error*. The discretization error is only dependent on the choice of the discretization and the modeling error is based on the choice of the network architecture and hence both the errors remains static through out the optimization.

To facilitate the understanding of how $\|e_G^k\|^2$ converges and to simplify the analysis, we will next combine the optimization error and modeling error, while following standard FEM analysis to bound the discretization error. We make the following assumptions.

Assumption 1. There exists a constant $c > 0$ such that $\|u_{\theta^*} - u^h\| \leq c\|u_{\theta_k} - u_{\theta^*}\|, \forall k \geq 0$.

Assumption 1 implies that the modeling error can be upper bounded by the optimization error up to a constant. This should be possible since we can always find some constant c to ensure that the above assumption holds. Therefore, we now obtain

$$\|e_G^k\|^2 \leq 3[(1 + c^2)\|u_{\theta_k} - u_{\theta^*}\|^2 + \|u^h - u\|^2]. \quad (3.14)$$

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

Assumption 2 implies that the prediction function provided by IBN is Lipschitz continuous [33], 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.

We first investigate the convergence rate when \mathcal{J} is *strongly convex and in a batch setting*. Notice that all terms in Equation 3.10, including the boundary loss (which is L_2 loss of the imposed boundary conditions) and the PDE residual loss (from variational arguments) are strongly convex.

Lemma 3. Assume that the basis function $\phi_i(\mathbf{x})$ are chosen such that they are at least continuously differentiable locally over a mesh. Then the following relationship holds true:

$$\|u^h - u\|^2 \leq Ch^{2\alpha}, \quad (3.15)$$

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

Theorem 4. Suppose that \mathcal{J} is μ -strongly convex and β -smooth. Let Assumptions 1 and 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,

$$\|e_G^K\|^2 \leq 3(1 + c^2)L^2N^2\hat{\phi}^2 \left(1 - \frac{2}{\kappa + 1}\right)^{2K} \|\theta_0 - \theta^*\|^2 + 3Ch^{2\alpha}. \quad (3.16)$$

Theorem 4 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

$$\lim_{K \rightarrow \infty} \|e_G^K\|^2 \leq 3Ch^{2\alpha}, \quad (3.17)$$

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 $\{\phi_i(\mathbf{x})\}_{i=1}^N$. If h is chosen such that $h = \mathcal{O}(\frac{1}{\sqrt{KN}})$ (justified by [12]), the following corollary can be obtained:

Corollary 5. *Suppose that $h = \mathcal{O}(\frac{1}{\sqrt{KN}})$. Given all conditions and parameters from Theorem 4, the generalization error achieves an overall sublinear convergence rate*

$$\|e_G^K\|^2 \leq \mathcal{O}\left(N^2 \rho_1^K + \frac{1}{(NK)^\alpha}\right). \quad (3.18)$$

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 a thumb rule 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 estimation of resource requirement for a desired discretization and error level.

Experimental Results

In this section, we provide results from our proposed immersed neural approach and a brief overview of the In-Out processes used. We highlight single instance field solutions for Poisson’s and Navier-Stokes equations and the field solutions to Poisson’s equation for a family of randomly generated shapes bounded by NURBS curves. As previously mentioned, the IBN framework can map sparse point cloud data to the field solution of a given PDE. The field solution to the PDE is represented as a uniform grid and may be thought of as a typical 2D image. The point clouds were composed of 1,000 (x, y) points, with each value lying between 0 and 1, and all field solutions were represented by a 128×128 pixel image. Our neural network used to map the point cloud to the image domain was composed of a 6-layer MLP followed by 8 layers of convolution and transpose convolution operations. All experiments were carried out on a single Nvidia Tesla P40. Each network used the Adam optimizer with a learning rate of $3e-4$.

During experimentation, we demonstrated two different methods of generating masks to impose the boundary conditions on our predicted field solution during the Finite Element loss computation. The first was with a differentiable winding number method. The second leverages the Finite Element framework to solve the Eikonal equation. The field solution to the Eikonal equation is a signed distance field. Once a signed distance field is obtained for the irregular boundary, we use a differentiable conditional function to create an accurate binary mask from the signed distance field.

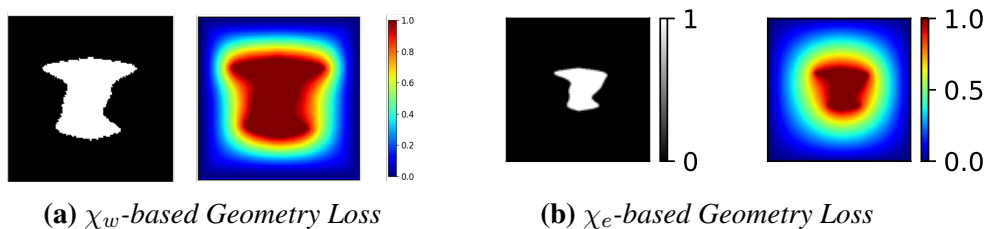


Figure 3.3: (a): *Left: The in-out information is obtained by the winding number calculation. Right: Poisson’s equation is solved with an immersed object represented using a 2D point cloud.* (b) *The in-out information is obtained by solving the Eikonal equation in addition to the solution to the Poisson’s equation using the IBN framework.*

Poisson’s Equation with χ_w -based Geometry Loss: The canonical Poisson’s equation was introduced in Equation 3.2. In this section we apply IBN to solve the Poisson’s equation with $f = 0$ and the boundary conditions defined as: $u = 1$, on Γ_o and $u = 0$, on Γ_B . This case mimics a steady heat transfer problem where Ω_o represents a heat source (of infinite capacity) and the outer boundary Γ_B represents a sink (of infinite capacity). The solution to Equation 3.2 along with the above f and the boundary conditions is the temperature field under those conditions. This case essentially refers to a large family of different problems where Γ_B can be any closed surface such that $\Omega_o \subset \Omega_B$. As discussed in Section 3, we consider the cases where such source object shapes are available as point cloud data, and our goal is to devise a fast neural method to use the point cloud data directly in analysis without having to go through a mesh generation process. Figure 3.3a shows the results of the winding number and the corresponding solution to the PDE.

Poisson’s Equation with χ_e -based Geometry Loss: In this case, along with the Poisson’s Equation, we solve Eikonal equation $(1 + \tau)\|\nabla\phi\| - \tau(\nabla\phi) = 1$, $\tau \in [0, 0.5]$ with $\phi = 0$ at Γ_o . For this case, we simultaneously solve for both the PDEs and show the results in Figure 3.3b.

Convergence Analysis: To check the accuracy of our method, we solve the Poisson’s equation Equation 3.2 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 $u(r) = \frac{1}{4}(R^2 - r^2)$, where $r = \sqrt{x^2 + y^2}$ is the radial position of a point on the disk.

To solve this, we discretize the unit square domain using an $N \times N$ grid. And we use a point cloud to denote the object boundary Γ_o . And then, we optimize the loss function mentioned in Section 3 to obtain the solution. The results are presented in Figure 3.4. Figure 3.4a shows the contours of the discrete solution u^h , the exact solution u and the error $e = u^h - u$, for $N = 128$. Figure 3.4b shows the L^2 -norm of the error alongside the mesh size h on a log-log plot. We see a first-order convergence with increasing resolution, which matches established theory from immersed boundary analysis ([4](Lemma 37.2) and [28]).

Navier-Stokes Equation: We also validate our framework against a canonical flow past an airfoil using the steady Navier-Stokes equations (NSE) that were introduced in Equation 3.3. The flow field output from IBN shows the expected wake structure (at Reynolds number 40), is symmetric

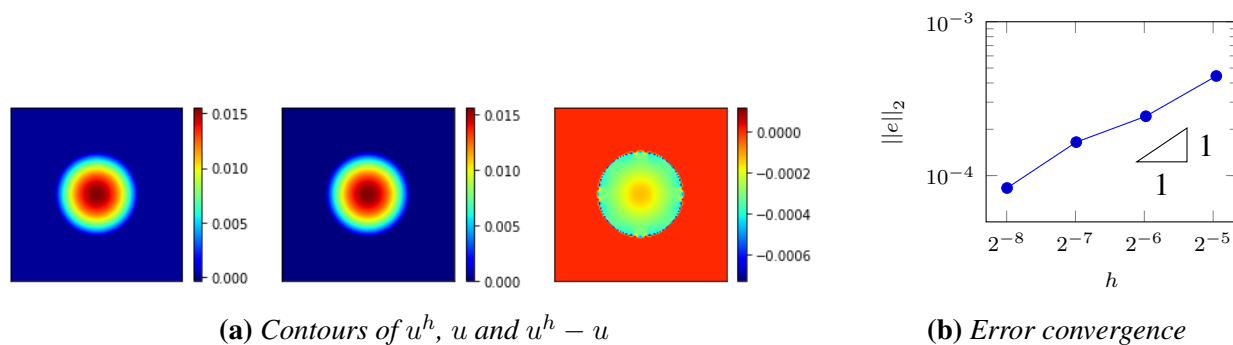


Figure 3.4: Error convergence with linear basis ($\alpha = 1$) for the Poisson’s equation solved on a circular disk of radius $R = 0.25$.

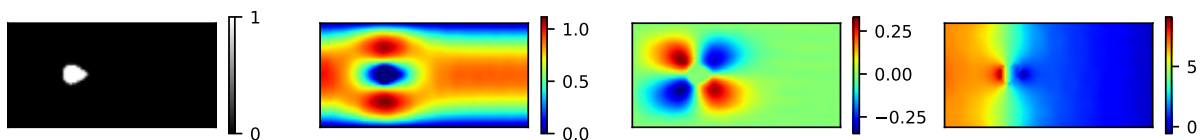


Figure 3.5: Immersed method with object mask applied to solve the Navier-Stokes equation for a steady flow past a NACA 0012 aerofoil. (left) domain/boundary mask (middle left) x velocity, (middle right) y velocity, and (right) pressure.

about the mid-plane, shows the stagnant pressure point on the upwind side of the immersed object, and satisfies the imposed no-slip condition.

The boundary conditions for this problem are:

$$x = 0 : u_x = 1 - \left(\frac{2y}{H} - 1 \right)^2, \quad u_y = 0 \quad (3.19a)$$

$$y = 0 : u_x = 0, u_y = 0 \quad (3.19b)$$

$$y = H : u_x = 0, u_y = 0 \quad (3.19c)$$

$$(x, y) \in \Gamma_o : u_x = 0, u_y = 0. \quad (3.19d)$$

Figure 3.5 shows the x velocity, y velocity, and the pressure solution for the Navier Stokes equation using IBN for an aerofoil.

Parametric Steady State Heat Equation: We revisit Poisson’s equation for a family of objects characterized by boundary Γ_o . Specifically, we consider the object boundary to be taken from a set of parameterized curves. The parametrization here is a NURBS-based one. This is equivalent to having access to the point cloud representing the object. We show results for two distinct distributions of objects. The first example consists of irregularly shaped objects without any obvious parametrization, while the second example consists of a parametrizable family of one thousand NACA airfoils. Figure 3.6 shows some anecdotal example predictions of such a trained network. We note that each example consists of a *single trained IBN*, which is then queried for a set of unseen object boundaries. IBN is able to accurately predict the field solution under non-trivial geometries, as a comparison with a high-resolution FEM solver confirms (columns 3, 6, 9, 12).

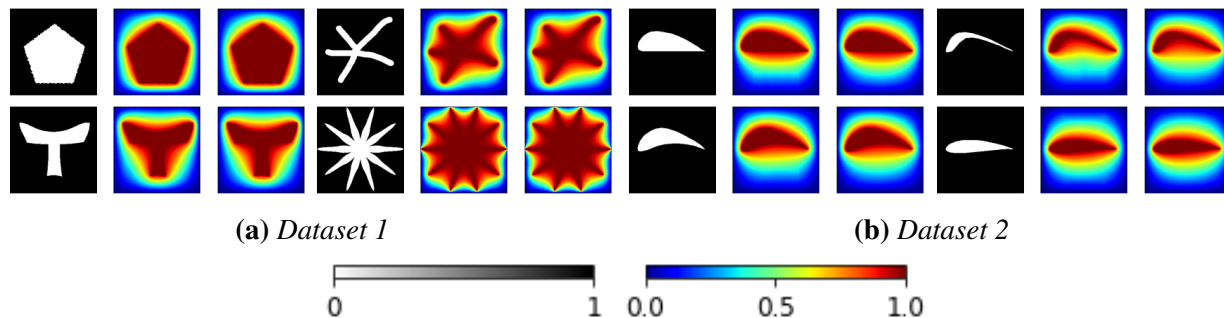


Figure 3.6: IBN applied on two different datasets. Within each panel: the left shows the heat source in white, the middle shows the IBN-predicted temperature distribution, and the right shows the fully converged numerical solution.

Broader Impacts

We have developed a neural PDE solver that can handle irregularly shaped domains by building on well-established finite element and immersed boundary methods. Our neural PDE solver, coined IBN, demonstrates the ability to predict field solutions for irregular boundaries immersed in the target domain. We highlight two specific PDE cases, Poisson 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. IBN opens up fast design exploration and topology optimization for various societally critical applications such as room ventilation for reduced disease risk, shape design for energy harvesters, and aerodynamic design of vehicles.

Task 8.2: Release of Software

We have released the following softwares during the execution of project:

1. <https://github.com/adityabalu/DiffNet>
2. <https://github.com/rocketmlhq/sciml>
3. <https://github.com/idealab-isu/NURBSDiff>
4. <https://github.com/idealab-isu/DSA>
5. <https://github.com/ameya005/Phys-GAN>
6. <https://doi.org/10.24433/CO.1250930.v1>
7. <https://doi.org/10.24433/CO.3688453.v1>
8. <https://doi.org/10.5281/zenodo.4552777>
9. <https://doi.org/10.5281/zenodo.2580293>

4 Publications Summary

- Anjana Deva Prasad, Aditya Balu, Harshil Shah, Soumik Sarkar, Chinmay Hegde, and Adarsh Krishnamurthy. Nurbs-diff: A differentiable programming module for nurbs. *Computer-Aided Design*, page 103199, 2022
- Biswajit Khara, Aditya Balu, Ameya Joshi, Soumik Sarkar, Chinmay Hegde, Adarsh Krishnamurthy, and Baskar Ganapathysubramanian. NeuFENet: Neural finite element solutions with theoretical bounds for parametric pdes. *arXiv preprint arXiv:2110.01601*, 2021
- Chih-Hsuan Yang, Balaji Sesha Sarath Pokuri, Sangeeth Balakrishnan, Chinmay Hedge, Soumik Sarkar, and Baskar Ganapathysubramanian. Multi-fidelity machine learning models for structure-property mapping, 2022. ArXiv
- Xian Yeow Lee, Aditya Balu, Balaji Sesha Sarath Pokuri, Adarsh Krishnamurthy, Soumik Sarkar, and Baskar Ganapathysubramanian. Inverse design of microstructures via generative networks for organic solar cells. In *AAAI 2022 Workshop on AI for Design and Manufacturing (ADAM)*, 2021
- Ethan Herron, Anushrut Jignasu, Jaydeep Rade, Xian Yeow Lee, Aditya Balu, Adarsh Krishnamurthy, and Soumik Sarkar. Fast unsupervised generative design for structural topology optimization. In *AAAI 2022 Workshop on AI for Design and Manufacturing (ADAM)*, 2021
- Jaydeep Rade, Aditya Balu, Ethan Herron, Anushrut Jignasu, Sergio Botelho, Santi Advani, Soumik Sarkar, Baskar Ganapathysubramanian, and Adarsh Krishnamurthy. Multigrid distributed deep cnns for structural topology optimization. In *AAAI 2022 Workshop on AI for Design and Manufacturing (ADAM)*, 2021
- Ameya Joshi, Gauri Jagatap, and Chinmay Hegde. Adversarial token attacks on vision transformers. *arXiv preprint arXiv:2110.04337*, 2021

5 Budget Summary

For the DOE ARPA-E award DE-AR0001215, managed by PI – Baskar Ganapathysubramanian:
The spending rate for the quarter January-March, 2022 averaged around \$100,000 per month with a total expenditure of \$1.5M till date from the beginning of the project.

Bibliography

- [1] Aditya Balu, Sergio Botelho, Biswajit Khara, Vinay Rao, Chinmay Hegde, Soumik Sarkar, Santi Adavani, Adarsh Krishnamurthy, and Baskar Ganapathysubramanian. Distributed multigrid neural solvers on megavoxel domains, 2021. Supercomputing 21, To Appear.
- [2] Gavin Barill, Neil G Dickson, Ryan Schmidt, David IW Levin, and Alec Jacobson. Fast winding numbers for soups and clouds. *ACM Transactions on Graphics (TOG)*, 37(4):1–12, 2018.
- [3] Sergio Botelho, Ameya Joshi, Biswajit Khara, Soumik Sarkar, Chinmay Hegde, Santi Adavani, and Baskar Ganapathysubramanian. Deep generative models that solve PDEs: Distributed computing for training large data-free models, 2020. arXiv 2007.12792.
- [4] Alexandre Ern and Jean-Luc Guermond. *Finite Elements II: Galerkin Approximation, Elliptic and Mixed PDEs*, volume 73. Springer Nature, 2021.
- [5] Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34): 8505–8510, 2018.
- [6] Ethan Herron, Anushrut Jignasu, Jaydeep Rade, Xian Yeow Lee, Aditya Balu, Adarsh Krishnamurthy, and Soumik Sarkar. Fast unsupervised generative design for structural topology optimization. In *AAAI 2022 Workshop on AI for Design and Manufacturing (ADAM)*, 2021.
- [7] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.
- [8] Ameya Joshi, Gauri Jagatap, and Chinmay Hegde. Adversarial token attacks on vision transformers. *arXiv preprint arXiv:2110.04337*, 2021.
- [9] Sharmila Karumuri, Rohit Tripathy, Ilias Billionis, and Jitesh Panchal. Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks. *Journal of Computational Physics*, 404:109120, 2020.
- [10] Biswajit Khara, Aditya Balu, Ameya Joshi, Soumik Sarkar, Chinmay Hegde, Adarsh Krishnamurthy, and Baskar Ganapathysubramanian. NeuFENet: Neural finite element solutions with theoretical bounds for parametric pdes. *arXiv preprint arXiv:2110.01601*, 2021.

- [11] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.
- [12] Mats G Larson and Fredrik Bengzon. *The finite element method: theory, implementation, and applications*, volume 10. Springer Science & Business Media, 2013.
- [13] Xian Yeow Lee, Aditya Balu, Balaji Sessa Sarath Pokuri, Adarsh Krishnamurthy, Soumik Sarkar, and Baskar Ganapathysubramanian. Inverse design of microstructures via generative networks for organic solar cells. In *AAAI 2022 Workshop on AI for Design and Manufacturing (ADAM)*, 2021.
- [14] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.
- [15] Yaron Lipman. Phase transitions, distance functions, and implicit neural representations. *arXiv*, 2021. doi: 10.48550/ARXIV.2106.07689. URL <https://arxiv.org/abs/2106.07689>.
- [16] Lu Lu, Pengzhan Jin, and George Em Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.
- [17] Craig Michoski, Milos Milosavljevic, Todd Oliver, and David Hatch. Solving irregular and data-enriched differential equations using deep neural networks. *arXiv preprint arXiv:1905.04351*, 2019.
- [18] Rajat Mittal and Gianluca Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.
- [19] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.
- [20] Charles S Peskin. The immersed boundary method. *Acta numerica*, 11:479–517, 2002.
- [21] Anjana Deva Prasad, Aditya Balu, Harshil Shah, Soumik Sarkar, Chinmay Hegde, and Adarsh Krishnamurthy. Nurbs-diff: A differentiable programming module for nurbs. *Computer-Aided Design*, page 103199, 2022.
- [22] Jaydeep Rade, Aditya Balu, Ethan Herron, Anushrut Jignasu, Sergio Botelho, Santi Adavani, Soumik Sarkar, Baskar Ganapathysubramanian, and Adarsh Krishnamurthy. Multigrid distributed deep cnns for structural topology optimization. In *AAAI 2022 Workshop on AI for Design and Manufacturing (ADAM)*, 2021.
- [23] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

- [24] Amuthan A Ramabathiran and Prabhu Ramachandran. Spinn: Sparse, physics-based, and partially interpretable neural networks for pdes. *Journal of Computational Physics*, 445: 110600, 2021.
- [25] Esteban Samaniego, Cosmin Anitescu, Somdatta Goswami, Vien Minh Nguyen-Thanh, Hongwei Guo, Khader Hamdia, X Zhuang, and T Rabczuk. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering*, 362:112790, 2020.
- [26] Kumar Saurabh, Boshun Gao, Milinda Fernando, Songzhe Xu, Makrand A Khanwale, Biswajit Khara, Ming-Chen Hsu, Adarsh Krishnamurthy, Hari Sundar, and Baskar Ganapathysubramanian. Industrial scale large eddy simulations with adaptive octree meshes using immersed-geometric analysis. *Computers & Mathematics with Applications*, 97:28–44, 2021.
- [27] Kumar Saurabh, Masado Ishi, Milinda Fernando, Boshun Gao, Kendrick Tan, Ming-Chen Hsu, Adarsh Krishnamurthy, Hari Sundar, and Baskar Ganapathysubramanian. Scalable adaptive pde solvers in arbitrary domains, 2021. *Supercomputing 21*, To Appear.
- [28] Dominik Schillinger, Isaac Harari, Ming-Chen Hsu, David Kamensky, Stein KF Stoter, Yue Yu, and Ying Zhao. The non-symmetric nitsche method for the parameter-free imposition of weak boundary and coupling conditions in immersed finite elements. *Computer Methods in Applied Mechanics and Engineering*, 309:625–652, 2016.
- [29] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
- [30] Jeffrey Slotnick, Abdollah Khodadoust, Juan Alonso, David Darmofal, William Gropp, Elizabeth Lurie, and Dimitri Mavriplis. CFD vision 2030 study: A path to revolutionary computational aerosciences. In *54th AIAA Aerospace Sciences Meeting*, 2014.
- [31] N Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.
- [32] Lloyd N Trefethen. *Spectral methods in MATLAB*. SIAM, 2000.
- [33] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.
- [34] Nils Wandel, Michael Weinmann, Michael Neidlin, and Reinhard Klein. Spline-pinn: Approaching pdes without data using fast, physics-informed hermite-spline cnns. *arXiv preprint arXiv:2109.07143*, 2021.
- [35] Chenglong Wang, Fei Xu, Ming-Chen Hsu, and Adarsh Krishnamurthy. Rapid b-rep model preprocessing for immersed-geometric analysis using analytic surfaces. *Computer aided geometric design*, 52:190–204, 2017.

- [36] Chih-Hsuan Yang, Balaji Sesa Sarath Pokuri, Sangeeth Balakrishnan, Chinmay Hedge, Soumik Sarkar, and Baskar Ganapathysubramanian. Multi-fidelity machine learning models for structure-property mapping, 2022. ArXiv.
- [37] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.