Available online at www.sciencedirect.com
ScienceDirect

Comput. Methods Appl. Mech. Engrg. 389 (2022) 114424

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains

Hengjie Wang[a,1], Robert Planas[b,1], Aparna Chandramowlishwaran[b], Ramin Bostanabad[b,*]

[a] *Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA*
[b] *University of California-Irvine, Irvine, CA, 92697, USA*

Received 23 June 2021; received in revised form 11 October 2021; accepted 29 November 2021
Available online xxxx

Abstract

Physics-informed neural networks (PINNs) are increasingly employed to replace/augment traditional numerical methods in solving partial differential equations (PDEs). While state-of-the-art PINNs have many attractive features, they approximate a specific realization of a PDE system and hence are problem-specific. That is, the model needs to be re-trained each time the boundary conditions (BCs) and domain shape/size change. This limitation prohibits the application of PINNs to realistic or large-scale engineering problems especially since the costs and efforts associated with their training are considerable.

We introduce a transferable framework for solving boundary value problems (BVPs) via deep neural networks which can be trained once and used forever for various unseen domains and BCs. We first introduce *genomic flow network* (GFNet), a neural network that can infer the solution of a BVP across arbitrary BCs on a small square domain called *genome*. Then, we propose *mosaic flow* (MF) predictor, a novel iterative algorithm that assembles the GFNet's inferences for BVPs on large domains with unseen sizes/shapes and BCs while preserving the spatial regularity of the solution. We demonstrate that our framework can estimate the solution of Laplace and Navier–Stokes equations in domains of unseen shapes and BCs that are, respectively, 1200 and 12 times larger than the training domains. Since our framework eliminates the need to re-train models for unseen domains and BCs, it demonstrates up to 3 orders-of-magnitude speedups compared to the state-of-the-art.
© 2021 Elsevier B.V. All rights reserved.

Keywords: Neural networks; Transferable deep learning; Scientific machine learning; PDEs; Navier–Stokes equations

1. Introduction

Partial differential equations (PDEs) are ubiquitously used in sciences and engineering to model physical phenomena. Two notable PDEs that have far-reaching applications are the Navier–Stokes (NS) equations which model the motion of viscous fluids [1] and Laplace equation which extensively appears in fluid dynamics, electrostatics, and steady-state heat transfer [2]. Solving such PDEs on large domains with arbitrary initial and boundary conditions (ICs and BCs) relies on numerical methods such as finite element (FE) [3] and finite difference (FD) [4]. While these methods are extremely powerful, they are computationally expensive. Because of these high costs, researchers face the challenge of drawing conclusions using a limited number of time-consuming studies. For

* Corresponding author.
E-mail address: Raminb@uci.edu (R. Bostanabad).
1 Equal Contribution.

https://doi.org/10.1016/j.cma.2021.114424
0045-7825/© 2021 Elsevier B.V. All rights reserved.

instance, NASA's CFD vision 2030 [5] estimates that achieving a 24-hour turnaround time on a wall-modeled large eddy simulation of a full-wing at Reynolds $Re = 10^7$ requires 180 PFlops/s which is only achievable on the most powerful supercomputer today. Such high costs make it infeasible to conduct compute-intensive studies such as uncertainty quantification (UQ) or airfoil shape optimization. To address this challenge, particularly in large-scale and inverse problems, simulations are augmented with inexpensive surrogates.

Although a wide range of surrogates such as Gaussian processes (GPs) [6–8] and trees [9,10] are available, most recent applications of surrogate modeling employ deep neural networks (DNNs) [11–21]. The increasing use of DNNs in emulation is largely because they have extremely high learning capacity which enables them to distill highly nonlinear and hidden features and relations from a training dataset without explicit instructions. In addition, DNNs are highly scalable and versatile. For instance, while DNNs can naturally build regressors/interpolators from large and high-dimensional data, GPs rely on numerical methods such as low-rank matrix approximation [8] to handle datasets with more than ~ 5000 samples [22,23]. Or, unlike trees, DNNs can easily handle both classification and regression/interpolation problems with high accuracy [13].

A major driver in using DNNs for surrogating PDE-governed systems is automatic differentiation (AD) [24] which allows the easy, systematic embedding of a system's governing equations in a network's training stage to build physics informed neural networks (PINNs) [25]. This inductive bias not only reduces the reliance on training data obtained from traditional solvers, but also increases the adherence of the DNN to the physics of the problem and dispenses with discretization errors (since AD is exact [26]). To demonstrate such a physics-informed training process, consider the $2D$ boundary value problem (BVP):

$$\begin{aligned} \nabla^2 u(\boldsymbol{x}) &= 0, \qquad \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{1}$$

where $\boldsymbol{x} = [x, y]$, ∇^2 is the Laplacian operator $(\partial^2/\partial x^2 + \partial^2/\partial y^2)$, Ω denotes the domain, and $g(\boldsymbol{x})$ is the boundary value function. When training a DNN that approximates $u(\boldsymbol{x})$, the loss function can be designed as:

$$\begin{aligned} L(\boldsymbol{\theta}) &= L_1(\boldsymbol{\theta}) + \alpha L_2(\boldsymbol{\theta}) + \beta L_3(\boldsymbol{\theta}), \\ L_1(\boldsymbol{\theta}) &= \frac{1}{N_1}\sum_{i=1}^{N_1}\left(u(\boldsymbol{x}_i) - \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\theta})\right)^2, \\ L_2(\boldsymbol{\theta}) &= \frac{1}{N_2}\sum_{j=1}^{N_2}\left(\nabla^2\mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\theta})\right)^2, \\ L_3(\boldsymbol{\theta}) &= |\boldsymbol{\theta}|^2 \end{aligned} \tag{2}$$

where $\boldsymbol{\theta}$ are the parameters of the network (weights and biases), $\mathcal{N}$ is the DNN approximation, $L_1(\boldsymbol{\theta})$ is the network's error in predicting the available data on the solution (these data can be on $\partial\Omega$ or in Ω), $L_2(\boldsymbol{\theta})$ is the residual error which enforces $\mathcal{N}$ to satisfy the Laplace equation on some randomly selected points (aka collocation points) in Ω, $L_3(\boldsymbol{\theta})$ denotes Tikhonov regularization that prevents overfitting, and α and β are constants that control the contributions of, respectively, $L_2(\boldsymbol{\theta})$ and $L_3(\boldsymbol{\theta})$ to $L(\boldsymbol{\theta})$. $L_2(\boldsymbol{\theta})$ in Eq. (2) involves partial derivatives which can be analytically calculated via AD.

Training a DNN that surrogates the solution of PDEs can be a time-consuming and challenging task primarily because (1) obtaining training data via high-fidelity simulations is expensive, (2) designing an efficient network architecture is an iterative process, (3) optimizing the network's parameters (such as weights and biases) is demanding, (4) calculating the residual loss is computationally costly because each time a differential operator is applied to the network, it almost doubles the number of computations associated with backpropagation [13,27], and (5) choosing the optimizer's parameters (such as batch size and learning rate) as well as balancing the contributions of different loss components to the overall loss (e.g., α and β in Eq. (2)) are iterative processes. These challenges are exacerbated when training DNNs that need many parameters to accurately approximate the solution of complex PDEs (such as the NS equations) in large domains.

We argue that the benefits of DNNs that approximate PDE-governed systems outweigh the high costs and challenges associated with their training if the DNNs are *transferable*. Unfortunately, existing methods [28–32] fail in this regard, i.e., there is currently no mechanism for estimating the solution of PDEs with a *pre-trained* DNN. For example, a PINN that approximates $u(\boldsymbol{x})$ in Eq. (1) would be largely useless if Ω and $g(\boldsymbol{x})$ change.

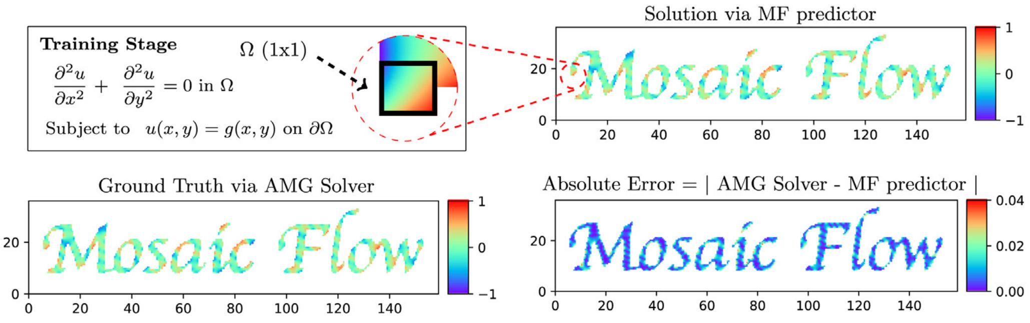


Fig. 1. (Color Online) Transferable learning of a PDE: We train our network in a small domain of size 1×1 for a wide range of boundary conditions. We then use the network to solve the Laplace equation in a domain that is $1200\times$ larger. Prediction time is almost 3 hours on a single NVIDIA V100 GPU.

To bridge this gap, we develop a novel framework that builds a transferable DNN surrogate that solves PDEs in unseen domains with arbitrary BCs (see Fig. 1). We present our idea in the context of flow prediction (i.e., surrogating the solution of the NS equations) but note that the framework is readily applicable to other PDE types such as the Laplace and Poisson equations. The source codes and pre-trained models of our framework can be downloaded at https://gitlab.com/UCI--CFD/mosaicflow.git.

For on-the-fly prediction of the flow in an unseen large domain with unseen BCs, we first decompose the domain into small subdomains called *genomes*. Then, we predict the flow in each genome with a pre-trained DNN, called *genomic flow network* (GFNet), such that the iterative assembly of these genome-wise predictions approximates the flow in the large domain. We denote this assembled flow as a *mosaic*. More specifically, this paper makes the following contributions:

- We introduce GFNet which can solve a BVP with arbitrary BCs on a small square domain called *genome*. We design GFNet's architecture based on the characteristics of the PDE and show that such a design significantly improves accuracy. For instance, our GFNet is twice more accurate than the conventional fully connected (FC) architecture for solving the Laplace equation (Section 4.1). For learning the NS equations, we further enforce the input BC in GFNet and reduce the generalization error by nearly 70% compared to DeepONet [33] and Fourier Neural Operator (FNO) [31] (Section 4.3).
- We develop *mosaic flow* (MF) predictor, a novel iterative algorithm based on the *continuous Schwarz alternating algorithm* [34] that iteratively assembles GFNet's inferences to predict the solution of the PDE system for unseen domains spanned by genomes with unseen BCs. We show that MF predictor can scale up GFNet's inferences to domains $\sim 1200\times$ and $12\times$ larger in the case of, respectively, Laplace and NS equations (Section 4). Moreover, MF predictor is independent of the neural network model used as GFNet. We compare our model with DeepONet [33] and FNO [31] and demonstrate that it reduces the error by nearly 6 times when employed by MF predictor to predict unseen flow features.
- With GFNet and MF predictor, we eliminate the need to re-train a DNN for unseen domains or BCs. This transferability delivers 1–3 orders of magnitude speedup compared to the start-of-the-art PINN for solving the Laplace and NS equations while achieving comparable or better accuracy (Section 4).
- Our GFNets are not tailored to specific applications and hence can be used by others. This reusability saves energy and benefits researchers with limited access to GPUs.

2. Related work

Existing approaches for building DNNs that solve PDEs employ a wide range of training mechanisms and network architectures. These choices largely depend on the application and the characteristics of the PDEs. The architectures are typically built with fully connected (FC), convolutional, or recurrent [35–37] layers. In many works a combination of these layers, with or without residual connections [38], are employed as well. For instance, networks similar to Resnet [38] and UNet [39], whose building blocks are convolutional neural networks (CNNs), are generally the backbone of super-resolution frameworks that aim to reconstruct a high-resolution solution from a coarse solution [29,40]. Such networks have also been trained in the Fourier space to learn a PDE operator using the modal space [31,41]. CNNs, which may also include FC layers, are extensively used as surrogates for predicting steady laminar flows around 2D and 3D objects [42], estimating 2D flows around specific airfoils [43,44], flow visualization [45], and many other applications [46–49].

Solving PDEs via CNN-based networks is data-efficient (due to parameter sharing and learning spatially invariant kernels [11,50–52]), but it requires developing a mechanism for interpolating the solution (and its gradients) on non-grid points [53]. One strategy to avoid such interpolation errors is to only employ FC layers. This strategy has been used for inverse estimation of PDE coefficients [25], inverse estimation of flow fields given observations on a passive scalar [27], predicting the Reynolds stress in Reynolds-averaged NS simulations [54,55], and solving Poisson equation and eigenvalue problems [56]. Networks based on FC layers have also been reformulated for variational learning (to improve predictions on the boundaries) [30,57], surrogating fractional PDEs (where AD cannot calculate residual errors) [58], emulating long time-dependent PDEs (where long temporal features render the training very difficult) [32], and learning operators that map functions to functions [33,59,60].

Regardless of the architecture, existing works lack *transferability*, i.e., they build models that are largely useless if the domain and BCs associated with the PDE system are modified. This important issue has been rarely addressed. DNNs that learn a PDE operator [33,61], can in theory handle varying BCs but they are not domain-agnostic and their applications thus far have been limited to simple PDEs over small, fixed domains. In [62], CNNs are used to solve the $2D$ Poisson equation on rectangular domains with different aspect ratios. In [63], the DNN is trained to solve the NS equations over backward-facing steps with varying heights. In [64], the DL model infers the lift and drag of various elliptic objects with different aspect ratios. In all these works, the geometry type (e.g., rectangle or ellipse) is fixed and only varies within a very limited range with one parameter. A few studies [43–45,49,65] have proposed more general techniques to embed geometry information into the training stage of a DNN such that it can predict the quantities of interests in unseen domains. However, the overall domain size is still fixed, grid data are required, and the variability of the objects is limited to specific applications.

Since the training costs of a PDE-solving DNN are generally high, parallel and distributed computations have been explored in their training [28]. However, the overall training costs are still high given the iterative and combinatorial process of tuning the architecture, optimization parameters, and loss components. To justify these high training costs, a DNN is expected to be transferable across multiple applications.

3. Transferable learning of PDEs

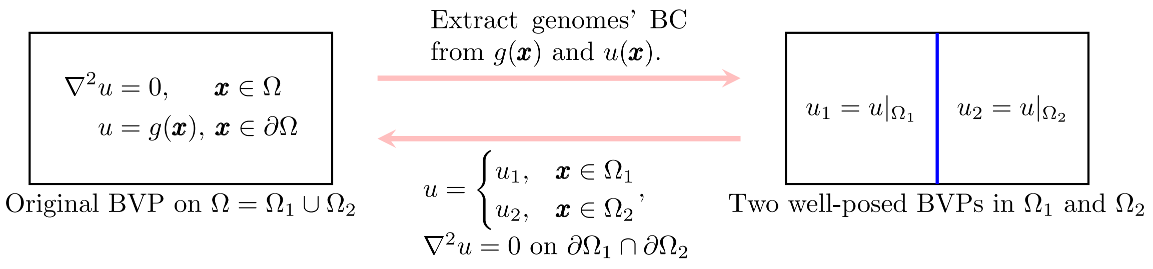
Our goal is to approximate the solution of the following homogeneous BVP in an *arbitrary* domain Ω for a wide range of $g(\boldsymbol{x})$:

$$\begin{aligned} H(u(\boldsymbol{x})) = 0, \quad & \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) = g(\boldsymbol{x}), \quad & \boldsymbol{x} \in \partial\Omega, \end{aligned} \tag{3}$$

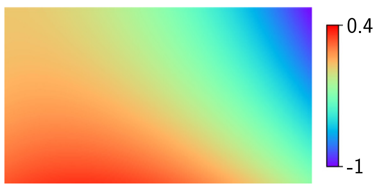
where $H(\cdot)$ denotes a partial differential operator composed of spatial derivatives. Henceforth, we drop the dependence of u to $\boldsymbol{x}$ for notational convenience. We highlight that while Eq. (3) is scalar, our approach is directly applicable to PDE systems such as the NS equations, see Section 4.2.

Building a *single* DNN that solves the BVP of Eq. (3) in *any* domain and under *any* BC is an extremely difficult (if not infeasible) learning task because considering arbitrarily large variations in domains and BCs in training drastically amplifies the challenges described in Section 1. In our framework, we solve this difficult task by decomposing it into two simpler (but highly coupled) tasks that correspond to (1) training a GFNet that solves the BVP in a genome (i.e., a small square) given any $g(\boldsymbol{x})$, and (2) building MF predictor that approximates the solution in any unseen large domain that can be spanned with genomes by stitching the genome-wise predictions made by the pre-trained GFNet. The advantage of our approach is that predicting the solution of the BVP in an unseen domain only relies on task 2, i.e., there is no need to train or re-train a DNN anymore.

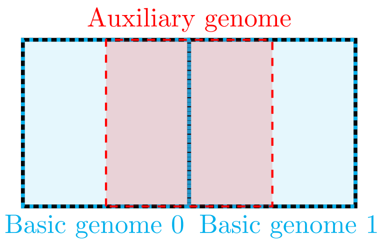
Our task decomposition strategy is motivated by curriculum learning [66] which was originally developed for animal training [67] where difficult tasks are divided into simpler tasks which are easier to learn and collectively make up the original task. In Section 3.1 we demonstrate that this approach is also applicable to solving BVPs. Then, we elaborate on how to build a GFNet and an MF predictor in Sections 3.2 and 3.3, respectively.


Extract genomes' BC from $g(\boldsymbol{x})$ and $u(\boldsymbol{x})$.
$\nabla^2 u = 0, \quad \boldsymbol{x} \in \Omega$
$u = g(\boldsymbol{x}), \ \boldsymbol{x} \in \partial\Omega$
Original BVP on $\Omega = \Omega_1 \cup \Omega_2$
$u = \begin{cases} u_1, & \boldsymbol{x} \in \Omega_1 \\ u_2, & \boldsymbol{x} \in \Omega_2 \end{cases}$,
$\nabla^2 u = 0$ on $\partial\Omega_1 \cap \partial\Omega_2$
$u_1 = u|_{\Omega_1}$
$u_2 = u|_{\Omega_2}$
Two well-posed BVPs in Ω_1 and Ω_2

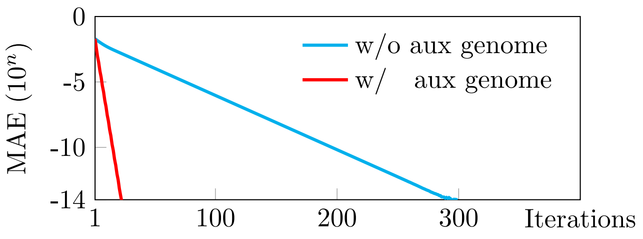
(a) Interchangeability between a BVP and its genomes' BVPs



(b) Direct solution via AMG



(c) Domain decomposition



(d) Convergence history in log-scale

Fig. 2. (Color Online) Decomposition of a BVP: The Laplace equation is solved in $\Omega = [0, 2] \times [0, 1]$ with three methods: a direct and two iterative approaches. The iterative approach that leverages an auxiliary genome converges to the direct solution 12× faster.

3.1. BVP decomposition: Why Mosaic works?

The BVP in Eq. (3) is well-posed if the solution, u, is unique. For such a BVP, u on any arbitrary sub-domain inside Ω also forms a well-posed BVP. Fig. 2(a) illustrates this relation for the Laplace equation where the domain Ω is split into two genomes, namely, Ω_1 and Ω_2. The boundaries of these genomes are denoted by $\partial\Omega_1$ and $\partial\Omega_2$ which share the border $\partial\Omega_1 \cap \partial\Omega_2$ in the middle of Ω (marked blue in the figure). Once the Laplace equation is solved in Ω, u will be available on $\partial\Omega_1 \cap \partial\Omega_2$. We can now solve two BVPs: one in Ω_1 and the other in Ω_2 to get the solutions u_1 and u_2, respectively. The above well-posedness relation indicates $u_1 = u|_{\Omega_1}$ and $u_2 = u|_{\Omega_2}$.

Conversely, if we can find u_1 and u_2 while ensuring that the PDE is satisfied on the shared border, i.e., $\nabla^2 u = 0$ on $\partial\Omega_1 \cap \partial\Omega_2$, then we will have u since $u = u_1 \cup u_2$. Indeed, to find u_1 and u_2 we need to solve two BVPs (one in Ω_1 and the other in Ω_2) both of which require u on $\partial\Omega_1 \cap \partial\Omega_2$. Since u is unknown, we propose an iterative strategy to find it on the shared border: we first randomly assign values to u on $\partial\Omega_1 \cap \partial\Omega_2$ and solve two BVPs in Ω_1 and Ω_2 to find u_1 and u_2, respectively. Then, we use u_1 and u_2 to update our initial guess and repeat this process until a convergence criterion is met (e.g., until values on $\partial\Omega_1 \cap \partial\Omega_2$ negligibly change across consecutive iterations). As illustrated below, we can dramatically accelerate the convergence speed of this simple iterative approach via the *continuous Schwarz alternating algorithm* (aka Schwarz method) [34] which greatly increases the pace of information propagation from $\partial\Omega$ to Ω using an *auxiliary* genome that overlaps with Ω_1 and Ω_2 (see Fig. 2(c)).

We illustrate the Schwarz method by solving the BVP in Eq. (1) in a domain of size 2×1 (i.e., $\Omega = [0, 2]\times[0, 1]$) subject to the boundary condition $g(\boldsymbol{x}) = x^2 - y^2 + xy$. As illustrated in Fig. 2(b), we first use algebraic multi-grid

(AMG) [68] to directly find the ground truth, i.e., u in Ω. Then, we use AMG within both iterative approaches and show that while both the simple iterative approach and Schwarz method converge to the ground truth, the latter is significantly faster.

Since Ω is of size 2×1, we decompose the domain into two *basic* genomes of size 1×1 that share a single border, see Fig. 2(c). In the first iterative strategy, we initialize the shared border with zero values for u, solve for u_1 and u_2 using AMG, and exchange data across the border. This iterative process converges in 300 iterations when the mean absolute error (MAE) compared to the direct solution drops below 1e-14.

In the Schwarz method, after finding u_1 and u_2, we update u on the shared border by solving the Laplace equation in an auxiliary genome whose BCs on its two vertical borders are inside Ω_1 and Ω_2, see Fig. 2(c). This approach introduces a stronger and more effective coupling across the basic genomes which, in turn, accelerates the rate of information propagation from $\partial\Omega$ to the shared border. The higher rate of information propagation increases the convergence rate by almost 12 times compared to the first iterative strategy, see Fig. 2(d).

For the general elliptic PDE,

$$\begin{aligned} H(u(\boldsymbol{x})) &= \nabla \cdot (\alpha \nabla u) - f = 0, \quad \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega, \end{aligned} \tag{4}$$

where $\alpha(\boldsymbol{x})$ is a variable coefficient in space and $f(\boldsymbol{x})$ denotes the source term, the iterative solution by Schwarz method converges to the PDE's exact solution [69,70]. To illustrate this, we formulate the approximate solution after solving BVP in the ith genome (Ω_i) as follows [71]

$$u^{k+\frac{i}{N_g}}(\boldsymbol{x}) = u^{k+\frac{i-1}{N_g}}(\boldsymbol{x}) + P_i(u^*(\boldsymbol{x}) - u^{k+\frac{i-1}{N_g}}(\boldsymbol{x})), \quad \boldsymbol{x} \in \Omega \tag{5}$$

where N_g denotes the total number of basic and auxiliary genomes, u^* is the exact solution for the given BC, and $u^{k+\frac{i}{N_g}}$ is the solution after solving BVP in the ith genome at the kth iteration. We define one iteration as updating all genomes exactly once. In Eq. (5), P_i is a projection operator mapping the error $u^* - u^{k+(i-1)/N_g}$ to a function that has zero values outside Ω_i. We refer readers to [71] for more details about P_i's proof of existence and properties. Applying Eq. (5) to all the genomes in one iteration yields the *error propagation map* [71],

$$|u^* - u^{k+1}| = |\prod_{i}^{N_g}(I - P_i)|\ |(u^* - u^k)|, \tag{6}$$

where the coefficient $E = |\prod_i^{N_g}(I - P_i)|$ prescribes how fast the error decays. It can be proved that $E < 1$ for general elliptic PDEs and this is independent of the numerical method (e.g., FE, FD, etc.) used for solving the genome-wise BVPs [69,70].

Our framework is akin to the Schwarz method, except for the following two aspects. First, instead of using a numerical method we always use a single pre-trained GFNet to predict the solution in either the basic genomes or the auxiliary genomes. Following Eq. (6), if GFNet's generalization error approaches 0 (see Section 4.3 for the definition of different error types), the prediction of our framework converges to the elliptic BVP's exact solution. Second, there is no mechanism in the original Schwarz method to define the proper arrangement of the overlapping sub-domains. In MF predictor, we distinguish between basic and auxiliary genomes and propose general guidance for placing genomes and ordering their updates (see Section 3.3).

3.2. Genomic Flow Network (GFNet)

The iterative nature of MF predictor requires a GFNet to be able to estimate the solution anywhere inside a genome for a wide range of BCs. Hence, the inputs of GFNet are:

$$\text{GFNet inputs:} \quad \underbrace{g(\boldsymbol{x}_1^{bc}),\ g(\boldsymbol{x}_2^{bc}),\ \cdots,\ g(\boldsymbol{x}_{N_{bc}}^{bc})}_{\boldsymbol{g} \text{ of size } N_{var} \times N_{bc}},\ \underbrace{\boldsymbol{x}}_{N_{dim}}$$

where N_{bc}, N_{var} and N_{dim} denote the number of points on the genome boundary, variables, and spatial dimensions, respectively. We consider $2D$ problems in this work so $N_{dim} = 2$. In scalar PDEs such as the Laplace equation, $N_{var} = 1$ while $N_{var} = 3$ for PDEs such as the incompressible NS equations studied in Section 4.2.

In the above, $\boldsymbol{g}$ represents a discretized version of $g(\boldsymbol{x})$ at the genome's boundaries and $\boldsymbol{x}$ indicates the coordinates of the point where GFNet predicts the solution, i.e., $u(\boldsymbol{x})$ which has N_{var} components. The loss function of GFNet is adopted from Eq. (2) where the size of the training data (e.g., N_1 and N_2) and coefficients (e.g., α and β) are chosen based on the BVP. For instance, our studies indicate that while residual loss improves the accuracy of GFNet when learning the Laplace equation, it reduces the accuracy in the case of the NS equations, see Section 4 for the details and our reasoning. Additionally, when residual errors improve the accuracy of GFNet, we introduce a mechanism to adaptively (instead of randomly) choose the location of collocation points in the genome.

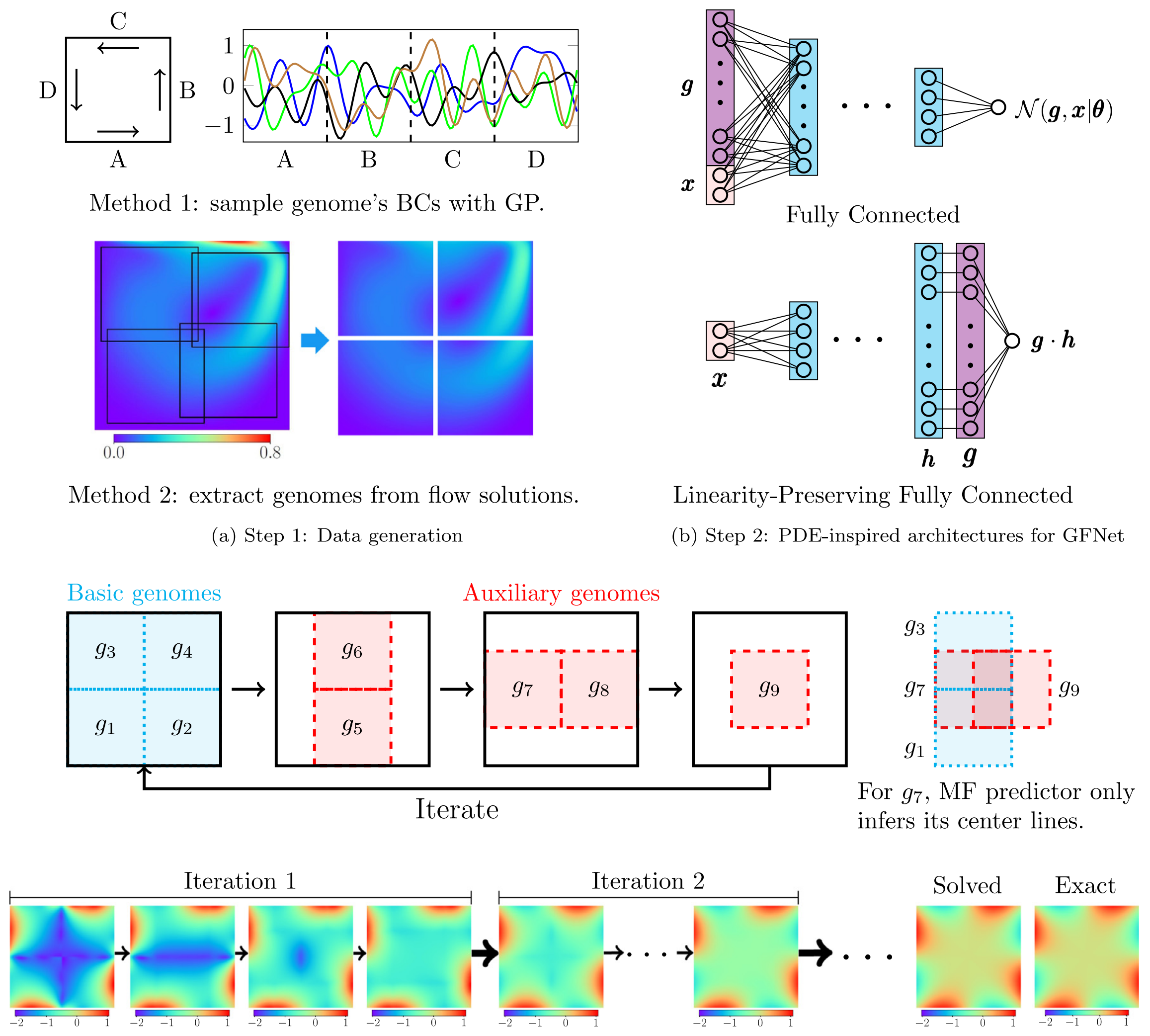
We design the architecture of GFNet based on the PDE. In linear PDEs such as the Laplace equation, $u(\boldsymbol{x})$ is essentially a linear combination of the discretized boundary function values, $\boldsymbol{g}$ (note that the solution can still be non-linear if $g(\boldsymbol{x})$ is a linear function). To preserve the linearity, we propose a novel *linearity-preserving fully connected* (LPFC) architecture shown in Fig. 3(b). In LPFC, we pass the coordinates of the input point $\boldsymbol{x}$ through multiple fully connected hidden layers and set the last hidden layer's outputs, $\boldsymbol{h}$, to have the same size as $\boldsymbol{g}$. The output of GFNet is a linear combination of $\boldsymbol{g}$ with $\boldsymbol{h}$, i.e., $\boldsymbol{g} \cdot \boldsymbol{h}$. With this configuration, $\boldsymbol{h}$ leverages the spatial correlations in Ω and $\boldsymbol{g} \cdot \boldsymbol{h}$ enforces it to conform to the linearity of the PDE. For non-linear PDEs such as the NS equations, we adopt a *fully connected* (FC) architecture where $\boldsymbol{g}$ and $\boldsymbol{x}$ are concatenated into one input vector and passed through all the FC hidden layers, see Fig. 3(b). A detailed evaluation of LPFC and FC architectures is presented in Section 4.

Since a GFNet takes $\boldsymbol{g}$ as an input, a wide range of well-posed (and ideally negligibly correlated) BCs must be used in training. For linear PDEs such as the Laplace equation, the BVP has a unique solution as long as the boundary function $g(\boldsymbol{x})$ is smooth, i.e., its derivatives in $H(\cdot)$ exist. Hence, we use GPs to generate a smooth function in $1D$ and then wrap that function around the perimeter of the genome, see Fig. 3(a). GPs are stochastic processes that place a prior distribution on functions. By judiciously selecting the hyperparameters of their kernel, we can generate a wide range of smooth functions that are minimally correlated. For highly nonlinear PDEs such as the NS equations, the BCs must satisfy mass, momentum, and energy conservation laws which makes it highly challenging to create well-posed BCs via a GP. Hence, we first solve the PDEs with realistic BCs on a large domain and then extract solutions for small genomes embedded in the domain. This strategy is demonstrated in Fig. 3(a) where the flow in a lid-driven cavity is swept by a genome while recording the flow on the boundary as well as inside the genome (the cavity's domain spans $[0, 1] \times [0, 1]$ while the genome has a size of 0.5×0.5). As argued in Section 3.1, this procedure generates valid data since the flow on each genome's boundary constitutes the BCs for the BVP on that genome.

Note that the genomes in training and prediction have the same shape because GFNet does not learn the effect of the genome's shape. We can address this limitation by including the boundary's geometrical information as input but this is beyond the scope of this work.

3.3. *Mosaic flow predictor*

As detailed in Section 3.1 we can (1) convert a BVP in a large domain to multiple BVPs in genomes that cover that domain and then solve these BVPs iteratively, and (2) employ auxiliary genomes to significantly accelerate the convergence. Based on the above two attributes, MF predictor decomposes an unseen domain into two sets of genomes, see Fig. 3(c). The first set consists of basic genomes that cover the entire domain and can overlap with each other. The BCs on the border of basic genomes are either known (if the borders lie on the large domain's boundary) or must be inferred via MF predictor. The second set consists of auxiliary genomes that overlap with the basic genomes to accelerate the propagation of boundary information into the domain, i.e., to speed up the iterative process of estimating the unknown BCs of the basic genomes. We distribute these two sets of genomes based on the following general guidelines. First, we arrange the basic genomes to cover the entire domain with as little overlap as possible ($g1$ through $g4$ in Fig. 3(c)). Then, we place auxiliary genomes on the vertical and horizontal shared borders of the basic genomes ($g5$ through $g8$ in Fig. 3(c)). Finally, we place an additional auxiliary genome on every corner/region where four basic genomes connect/overlap ($g9$ in Fig. 3(c)). We have developed this spatial genome distribution to balance the pace of information transfer (which directly affects convergence speed) and implementation efforts. As we demonstrate in Section 4.2.3, for highly nonlinear PDEs such as the NS equations, more auxiliary genomes are required to facilitate information propagation. In such cases, following the above procedure, we place the additional auxiliary genomes on the shared borders of other auxiliary genomes. We will explore more advanced techniques in our future works.


C
D
B
A
1
0
−1
A B C D
Method 1: sample genome's BCs with GP.
0.0
0.8
Method 2: extract genomes from flow solutions.
$\boldsymbol{g}$
$\boldsymbol{x}$
$\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}|\boldsymbol{\theta})$
Fully Connected
$\boldsymbol{x}$
$\boldsymbol{g} \cdot \boldsymbol{h}$
$\boldsymbol{h}$ $\boldsymbol{g}$
Linearity-Preserving Fully Connected
Basic genomes
Auxiliary genomes
g_3 g_4 g_1 g_2
g_6 g_5
g_7 g_8
g_9
Iterate
g_3 g_7 g_1 g_9
For g_7, MF predictor only infers its center lines.
Iteration 1
Iteration 2
Solved
Exact
−2 −1 0 1

(a) Step 1: Data generation

(b) Step 2: PDE-inspired architectures for GFNet

(c) Step 3: Iterative nature of MF predictor

Fig. 3. (Color Online) Primary steps of our framework for transferable PDE learning: Steps 1 and 2 are done only once and their cost is fully amortized in step 3 which can be applied to unseen domains. (*a*) Training data are generated by either solving a BVP whose BCs are sampled from a GP, or by sweeping the solution of the BVP in a large domain with a genome (the plot in method 2 corresponds to the velocity magnitude in a lid-driven cavity simulation). (*b*) A GFNet takes the discretized boundary function $\boldsymbol{g}$ and $\boldsymbol{x}$ as inputs and outputs the solution at $\boldsymbol{x}$. We use LPFC and FC architectures for linear and non-linear PDEs, respectively. (*c*) Mosaic flow (MF) predictor estimates the solution in an unseen domain by systematically coupling the predictions of a GFNet on basic and auxiliary genomes. Note that basic genomes can overlap with each other and auxiliary genomes can be placed anywhere inside the domain (we have avoided these scenarios to improve clarity).

MF predictor finds the solution in the genomes set by set using a pre-trained GFNet and iterates until convergence. In one iteration, MF predictor updates all the genomes once in the order that the genomes are grouped and placed. That is, it first predicts for all the basic genomes, then infers auxiliary genomes covering the basic genomes' horizontal and vertical borders, and at last updates the genomes overlapping the basic genomes' corners. The rationale behind this order is that, updating genomes with more boundary information (adjacent or close to boundary) before the ones with less information can maximize the pace at which the information is propagated to the domain's interior. For instance, the basic genomes are inferred first as they cover the entire domain boundary while the auxiliary genomes overlapping the basic genomes' corners are inferred at last because they are typically immersed inside the domain without direct contact to the boundary. The basic genomes' borders are initialized either with the domain's BC if they lie on the domain's boundary or with 0 if they lie between basic genomes . The predictions in basic genomes provide the BCs for auxiliary genomes and then the predictions in auxiliary genomes update the unknown BCs of the basic genomes. The iterations stop when the change in the inferred BCs of basic genomes is smaller than the user-defined tolerance ϵ.

Fig. 3(c) illustrates the spatial distribution of basic and auxiliary genomes and the iterative nature of MF predictor for solving the Laplace equation in $\Omega = [0, 2] \times [0, 2]$ with genomes of size 1×1. In this figure, there are 4 basic genomes ($g1$-$g4$) that, without any overlap, cover Ω. The BCs are only known on two sides of the basic genomes and hence the MF predictor aims to predict the unknown BCs on the other two sides. There are also 5 auxiliary genomes ($g5$-$g9$) that accelerate the pace of inferring the unknown BCs of the basic genomes. In one iteration, the genomes are updated in the following order: $g1$-$g4 \rightarrow g5, g6 \rightarrow g7, g8 \rightarrow g9$. (Within this order, non-overlapping genomes can be updated concurrently, e.g., $g1$ through $g4$ or $g5$ and $g6$. We plan to exploit this parallelism in our future work.) In this example, MF predictor converges to the exact solution in 18 iterations.

In contrast to conventional numerical methods and PINN, MF predictor does *not* solve for all the grid/collocation points in the entire domain. Instead, it only infers the solution on the borders of basic and auxiliary genomes. For instance, for genome g_7 in Fig. 3(c), MF predictor uses the pre-trained GFNet to predict the solution on its vertical and horizontal center lines. These predictions are then used for inferring the solutions in genomes g_9 (in the current iteration) as well as g_1 and g_3 (in the next iteration).

4. Results and discussions

We evaluate GFNet and MF predictor by solving the linear Laplace equation and the non-linear incompressible NS equations in Sections 4.1 and 4.2, respectively. In each section, we first present the details on data generation, GFNet architecture, accuracy, and performance (the latter refers to computational costs). Then, we compare GFNet and MF predictor against the state-of-the-art PINNs. In Section 4.3 we do an in-depth error analysis of our framework and test its accuracy when the Fourier neural operator (FNO) [31] and deep operator networks (DeepONet) [33] are used as GFNets and in Section 4.4 we demonstrate the advantage of enforcing the network architecture to satisfy the applied BCs.

We use Tensorflow [72] and ADAM [73] optimizer. We implement GFNet with TensorFlow 2 (2.2.0) whereas the PINN related sources [25,74] used for comparison are built with TensorFlow 1 (1.8.0). In Section 4.3, we implement FNO with Pytorch 1.9.0 and DeepONet with Tensorflow 2 (2.4.0). For our models, we initialize the learning rate, η, by 5e-4 and reduce it by 20% when the validation loss decreases by less than 0.01% across 200 consecutive epochs. We use a $9 : 1$ ratio for splitting the data into training and validation samples and terminate the training when $\eta =$ 1e-7. We measure accuracy via mean absolute error (MAE) and mean absolute residual (MAR) metrics calculated as:

$$\text{MAE: } \sum_{i=1}^{N_g} |\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}_i|\boldsymbol{\theta}) - u(\boldsymbol{x}_i)|/N_g \qquad \text{MAR: } \sum_{i=1}^{N_g} |H\left(\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}_i|\boldsymbol{\theta})\right)|/N_g.$$

All the simulations are done on an NVIDIA Quadro RTX 8000 GPU.

4.1. Laplace equation

Laplace equation is an elliptic BVP that ubiquitously appears in fluid dynamics and heat transfer [2], see Eq. (1). In this section, we learn the $2D$ Laplacian operator via a GFNet in a unit square domain, i.e., the genome covers $[0, 1] \times [0, 1]$. Then, we use MF predictor to solve the Laplace equation in domains that are up to 1200 times larger. We compare the accuracy and performance of GFNet and MF predictor against PINN [25] and XPINN [28].

4.1.1. Training data

As described in Section 3.2, we use GPs to generate BCs. In particular, we use Sobol sequence [75] to sample the hyperparameters of the infinitely differentiable Gaussian kernel of a $1D$ GP. Then, we draw a sample function (i.e., a $1D$ curve) from each GP and wrap it around the genome, see Fig. 3a. Finally, for each BC, we numerically solve the Laplace equation via PyAMG [68] which relies on FD to provide the solution on a uniform grid that consists of 32×32 cells and 128 boundary points, see Fig. 4(a).

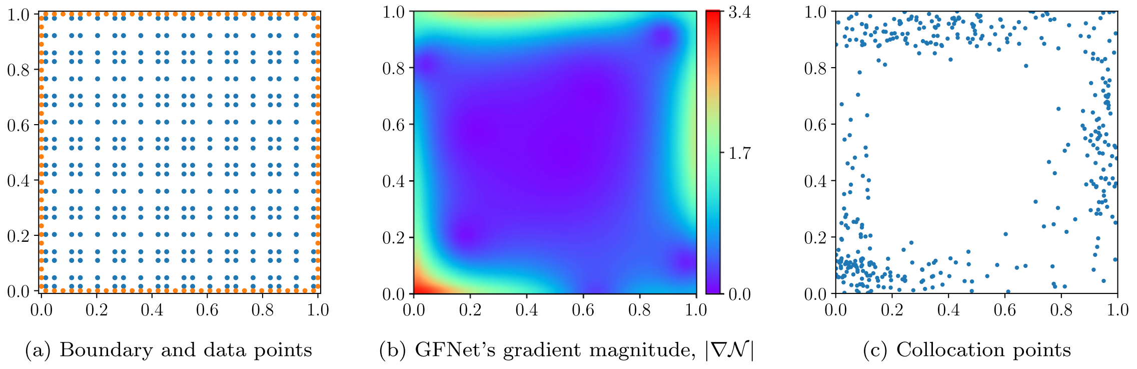


Fig. 4. (Color Online) Distribution of training data in Ω: 128 boundary points, 400 data points, and 400 collocation points. The locations of boundary and data points are fixed during training while those of the collocation points adaptively change based on $|\nabla\mathcal{N}|$.

4.1.2. Architecture and training of GFNet

As defined in Section 3.2, GFNet's inputs include $\boldsymbol{x}$ and the discretized boundary value function $\boldsymbol{g}$ which in this case has $N_{bc} = 128$ points. Since Laplace equation has one scalar variable (i.e., $N_{var} = 1$), GFNet has a total of $N_{bc} \times N_{var} + N_{dim} = 130$ inputs. The output of GFNet is a single value $\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}|\boldsymbol{\theta})$ that approximates $u(\boldsymbol{x})$.

We choose tanh as the activation function and evaluate both FC and LPFC architectures introduced in Section 3.2. The loss function of GFNet is defined in Eq. (2) and uses $\beta = 0$ and $\alpha =$1e-3. When minimizing this loss, three data types are used in each epoch: boundary points, data points, and collocation points. While the location of boundary and data points are determined by PyAMG (and hence fixed), we adaptively choose the location of collocation points during training based on the spatial gradients that GFNet estimates inside the genome. These gradients are estimated via AD and their magnitude, $|\nabla\mathcal{N}|$, controls the spatial distribution of the collocation points by increasing the point density in regions where $|\nabla\mathcal{N}|$ is large. Fig. 4(b) demonstrates the spatial distribution of these three data types in one randomly selected epoch during training. In this figure, boundary and data points are on a grid (since PyAGM generates them based on FD) while collocation points concentrate near the genome's border where the solution tends to change sharply.

4.1.3. Accuracy for unseen BCs

We train the FC and LPFC architectures using 2000 samples where each sample consists of 128 boundary, 100 data, and 400 collocation points. The FC architecture consists of 8 hidden layers of sizes $128^2 \otimes 96^2 \otimes 64^2 \otimes 32^2$ which make a cone as shown in Fig. 3(b). The notation $k^i \otimes l^j$ denotes i FC layers of size k followed by j FC layers of size l. We invert the order of these hidden layers for LPFC so that both architectures have the same number of neurons. The resulting GFNets are tested on 400 unseen samples generated by PyAMG where the MAE and MAR are evaluated at all the 32×32 grid cells.

The first two rows in Table 1 compare the accuracy of the two architectures. We observe that LPFC learns the Laplacian operator more accurately than FC by one order of magnitude. This superior accuracy is because FC applies a non-linear activation function (tanh) to $\boldsymbol{g}$ which conflicts with the linearity of the Laplace equation. On the contrary, LPFC applies the non-linear activation function only to $\boldsymbol{x}$ and retains the linearity of the Laplacian operator. Therefore, we use the LPFC architecture in the remainder of this section.

We further improve GFNet's accuracy by increasing its depth and the size of training data. (To reduce training costs, we do not use all the 32×32 points where PyAMG provides the solution.) In particular, we use 14 layers of size $32^2 \otimes 64^2 \otimes 96^5 \otimes 128^5$ and 8000 samples to reduce the MAE and MAR of LPFC by 1.3 and 1.8 times, respectively. Moreover, LPFC consistently outperforms FC in every case and exhibits 2 times higher accuracy with 8000 samples. Further increasing the number of samples or hidden layers did not noticeably affect accuracy. Therefore, we select LPFC architecture and use the 14-layer GFNet trained with 18 000 samples as the final model in MF predictor. We also test single and double-precision computations but no significant difference in accuracy is observed for either GFNet (see Table 1) or MF predictor (see Fig. 5). Since the former is 17% faster, we choose the model trained with single-precision.

Table 1
Accuracy of GFNet on learning the Laplacian operator: LPFC architecture learns the Laplacian operator more accurately than FC. The training column includes the number of layers, samples, data/collocation points, and the mean absolute error (MAE). The test column reports the MAE and the mean absolute residual (MAR) which are calculated by evaluating GFNet's predictions against PyAMG [68].

Network	Precision	Training				Test on unseen BCs	
		Layers	Samples	Points	MAE	MAE	MAR
FC	Single	8	2000	100\400	1.33e−3	2.64e−3	1.72e−1
LPFC	Single	8	2000	100\400	3.88e−4	5.42e−4	2.33e−2
FC	Single	14	4000	400\400	5.57e−4	1.30e−3	7.12e−2
LPFC	Single	14	4000	400\400	2.55e−4	4.33e−4	2.05e−2
FC	Single	14	8000	400\400	4.06e−4	8.79e−3	4.17e−2
LPFC	Single	14	8000	400\400	2.37e−4	4.10e−4	1.30e−2
FC	Double	14	8000	400\400	3.86e−4	8.12e−4	3.84e−2
LPFC	Double	14	8000	400\400	2.39e−4	4.07e−4	1.51e−2
LPFC	Single	14	18000	400\400	1.59e−4	4.10e−4	1.46e−2
LPFC	Double	14	18000	400\400	1.59e−4	4.10e−4	1.46e−2

4.1.4. Accuracy on unseen domains subject to unseen BCs

We first evaluate MF predictor on square domains of area $A > 1$ with two boundary functions $g_1(s) = \sin(2\pi s/\sqrt{A})$ and $g_2(s) = \sin(2\pi s)$ where s parameterizes the boundary. We gradually increase A and execute MF predictor in both single and double-precision. The accuracy of MF predictor is measured by the MAE between the converged solution and the ground truth generated with PyAMG. Fig. 5 summarizes the results.

At $A = 1$, the error is solely due to GFNet (i.e., inferring the solution for an unseen BC). For both $g_1(s)$ and $g_2(s)$, the MAE spikes at $A = 2\times 2$ as it starts to include additional error accumulated during the iterative process in MF predictor. However, as A increases, the influence of BC on the inner region of the domain reduces, and therefore MF predictor can more accurately predict the solution. As a result, MAE decays and plateaus once $A > 4 \times 4$.

The converged MAEs for $g_1(s)$ and $g_2(s)$ are different. For $g_1(s)$, the boundary function oscillates less across the domain boundary as A increases which simplifies the genome-wise BVP compared to $A = 1$. Therefore, at $A = 8 \times 8$, even with the additional accumulated error due to MF predictor, the MAE is less than that at $A = 1$. However, for $g_2(s)$ the oscillation frequency does not depend on A and hence the MAE at $A = 8 \times 8$ exceeds that at $A = 1$ and is also 4 times larger than the corresponding MAE for $g_1(s)$.

We also demonstrate the transferability and scalability of MF predictor by solving the Laplace equation in a domain that resembles "Mosaic Flow" and is 1222.5× larger than the training domain (i.e., a genome), see Fig. 1. The prescribed BC is $g(\boldsymbol{x}) = \sin(2\pi(x/6 + y/5))$ and 2020 genomes (basic and auxiliary) are used in MF predictor which converges in 9821 seconds with an MAE of 2.61e−3.

In this study, we only consider square genomes for GFNet. Accordingly, MF predictor can accurately assemble GFNet's inference for domains spanned by genomes, i.e., with rectilinear boundaries. However, for non-rectilinear domains such as the above calligraphy shape, we need to adopt a *mosaic* representation of the domain by decomposing the curved boundary to small squares, which results in the zigzag-shaped boundary in Fig. 1. We will investigate how to accurately consider non-square genomes in our future work.

4.1.5. Comparison with PINN and XPINN

We compare the accuracy and computational costs of GFNet and MF predictor against a state-of-the-art PINN which has the feed-forward architecture of $60 \otimes 60 \otimes 60 \otimes 60$. To maximize the accuracy of this PINN, we train it with a two-stage optimization process [76] that starts with 40 000 iterations using ADAM [73] and is followed by a second-order optimization method based on L-BFGS [77]. We use $20000 \times A$ collocation points and $32 \times 4 \times \sqrt{A}$ boundary points to consider the effect of A. Contrary to PINN, L-BFGS is not used in fine-tuning GFNet.

The evaluation results are illustrated in Fig. 5. PINN and GFNet achieve a similar accuracy for a single genome. For the simpler BC, $g_1(s) = \sin(2\pi x/\sqrt{A})$, PINN achieves smaller MAEs compared to MF predictor for $A > 1$. This higher accuracy is because (1) the parameters of PINN are fine-tuned with L-BFGS while GFNet is trained via Adam (GFNet cannot be trained via L-BFGS since it has too many parameters), and (2) PINN is specifically trained on this BC (and cannot be used for any other BC) while MF predictor assembles the predictions from a GFNet that

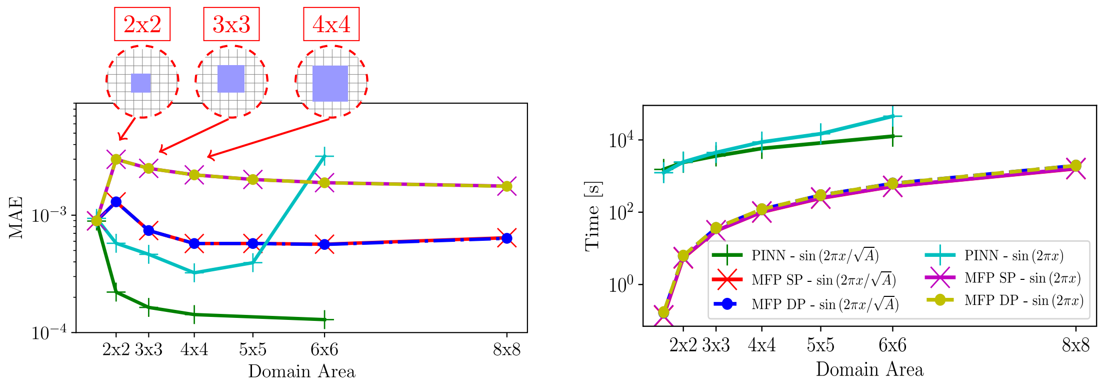


Fig. 5. (Color Online) Accuracy and performance of MF predictor and PINN for the Laplace equation with two different BCs: For the boundary condition $g_1(s) = \sin(2\pi s/\sqrt{A})$, PINN achieves better accuracy than MF predictor. However, for the more complex BC $g_2(s) = \sin(2\pi s)$, the accuracy of PINN drops, especially in larger domains. Note that while PINN is re-trained for every new BC, MF predictor has never seen these BCs. In terms of computational costs (sum of training and inference times), PINN is several orders of magnitude slower than MF predictor (we were unable to train PINN on the 8×8 domain due to very high costs). In addition, MF predictor with single (SP) and double-precision (DP) yields near identical results.

Table 2
XPINN vs. MF predictor: Both methods approximate the Laplace operator in a square domain of size 4×4.

Method	MAE
MF predictor	5.82e−4
XPINN	2.12e−2

has never seen this BC. For the more complex BC, $g_2(s) = \sin(2\pi x)$, PINN fails to scale up to large A; indicating that its architecture is domain-specific and must be optimized anew. Unlike PINN, MF predictor scales up quite robustly to large domains with unseen BCs without the need to re-train or fine-tune its GFNet. Remarkably, MF predictor is between 1–3 orders-of-magnitude faster than PINN depending on the domain size, which underscores its potential for scalable inference, especially on large and complex domains (for instance, we were unable to train a PINN on an 8×8 domain due to the extremely high training costs).

Finally, we test MF predictor against XPINN [28] which is an extension of PINN that aims to address PINN's scalability issues. XPINN divides a domain into some sub-domains and trains a PINN on each one while ensuring continuity across the sub-domains. In our comparison, we choose a square domain of size 4×4 subject to the BC: $g_1(s) = \sin(2\pi x/\sqrt{A})$ and divide it into 4 square sub-domains of equal sizes. The 4 PINNs used in XPINN have an architecture of $20 \otimes 20 \otimes 20 \otimes 20$ and are trained using Adam for 5000 epochs with 16000 collocation points and 128 boundary points. Table 2 enumerates the accuracy of XPINN and MF predictor and shows that our approach is two orders of magnitudes more accurate.

4.2. Navier–Stokes equations

The $2D$ steady incompressible NS equations are:

$$\begin{aligned} &H_1 : \partial_x u + \partial_y v = 0, \\ &H_2 : u\partial_x u + v\partial_y u + \partial_x p/\rho - \nu\nabla^2 u = 0, \\ &H_3 : u\partial_x v + v\partial_y v + \partial_y p/\rho - \nu\nabla^2 v = 0, \end{aligned} \tag{7}$$

where $u(\boldsymbol{x})$, $v(\boldsymbol{x})$, and $p(\boldsymbol{x})$ denote velocity components and pressure, respectively. We solve Eq. (7) in a lid-driven cavity problem where the top lid moves with an arbitrary horizontal velocity, i.e., $u = g(x)$, $v = 0$, and the remaining boundaries are treated as static viscous walls where $u = v = 0$. We fix $\rho = 1.0$ and $\nu = 0.002$ and aim to predict $u(\boldsymbol{x})$, $v(\boldsymbol{x})$, and $p(\boldsymbol{x})$ in unseen domains that have unseen BCs on the top lid and are larger than a genome. Our imposed lid velocity profiles can be highly nonlinear functions and collectively generate Reynolds numbers in the $(0, 500)$ range. The corresponding flow fields are also more complex than cases where the top lid moves with a uniform velocity.

The convergence theory in Section 3.1 does not strictly apply to the NS equations since the non-linear terms in H_2 and H_3 do not conform to the formulation of the elliptic systems in Eq. (4). Nonetheless, the incompressible NS equations can still be approximately viewed as elliptic because the information propagates in the same way as it does in elliptic systems where any fluctuation at one point affects the entire domain. Several studies have successfully employed Schwarz method with other numerical methods in solving the NS equations and present accurate numerical results [78–81]. In the following sections, we will demonstrate that MF predictor with GFNet can also learn the NS equations quite accurately and infer flow features that have not been seen in training.

4.2.1. Training data

We sample the velocity profile of the top lid using GPs and solve the NS equations with OpenFOAM [82] on a uniform grid of 129×129 vertices. Then, for each solution field obtained via OpenFOAM, we sweep it with a genome of size 0.5×0.5 and record the values on the boundary and inside the genome, see Fig. 3(a). This procedure results in highly correlated training samples that adversely affect GFNet's training. Hence, we (1) discard a sample if the Euclidean distance of its BC to another sample's BC, i.e., $|\boldsymbol{g}_i - \boldsymbol{g}_j|$, is small, and (2) include cavities of size 1×2 and 2×1. Following this procedure, 2634 samples are generated.

4.2.2. Architecture and training of GFNet

For each sample, we uniformly extract 128 boundary points from the grid vertices located on the boundary. GFNet's inputs include the velocities (u, v) at boundary points and $\boldsymbol{x}$, i.e., a total of $N_{var} \times N_{bc} + N_{dim} = 128 \times 2 + 2 = 258$ inputs. Following [76], we exclude the BC on p from the inputs and instead solve it as an internal variable. The GFNet has three outputs $\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}|\boldsymbol{\theta}) = (\hat{u}, \hat{v}, \hat{p})$ which approximate the solution (u, v, p) at $\boldsymbol{x}$. The loss function of GFNet is adopted from Eq. (2) as follows:

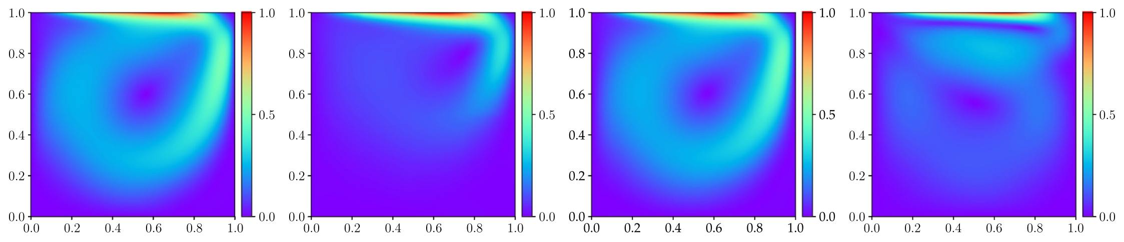
$$\begin{aligned} L(\boldsymbol{\theta}) = &\frac{1}{N_0}\sum^{N_0}\left(\gamma_1(\hat{u}-u)^2 + \gamma_2(\hat{v}-v)^2 + \gamma_3(\hat{p}-p)^2\right) \\ &+ \frac{1}{N_1}\sum^{N_1}\left(\alpha_1 H_1^2 + \alpha_2 H_2^2 + \alpha_3 H_3^2\right) + \beta|\boldsymbol{\theta}|^2. \end{aligned} \tag{8}$$

where the first summation minimizes the error on predictions, the second summation penalizes the residuals, and the last term represents the Tikhonov regularization. $\boldsymbol{\gamma} = [\gamma_1, \gamma_2, \gamma_3]$, $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \alpha_3]$, and β balance the contributions to the overall loss. When minimizing the PDE residuals, we cannot adapt the collocation points based on one variable's spatial gradients in the same way we do for learning the Laplace equation (see Section 4.1.2). This is because the point distribution based on one variable's gradient may not align with the distributions based on the other two variables' gradients. We could use the norm of the gradient vector but that will bias the training to favor learning one variable better in expense of losing accuracy for the other variables. To avoid this issue, we increase the number of collocation points by 3 times compared to solving the Laplace equations (1200 over 400, see Tables 1 and 3) and randomly initialize their positions with a uniform spatial distribution. These collocation points are fixed during training. We use the FC architecture due to the non-linear nature of NS equations and select the layer sizes as $256^3 \otimes 128^3 \otimes 96^3 \otimes 64^3 \otimes 32^2 \otimes 3$. All except the last hidden layer use tanh as the activation function. The depth and size of our GFNet are considerably larger than the networks in related works [25,74,76,83] because we (1) use BCs (which are high-dimensional) as inputs, and (2) find that a deep network with Tikhonov regularization generalizes better than shallow networks that exclude regularization. To find the optimal values of $\boldsymbol{\gamma}$, $\boldsymbol{\alpha}$, and β, we set $\gamma_1 = \gamma_2$ and $\alpha_1 = \alpha_2 = \alpha_3 = \alpha$ since u and v are equally important and the three equations are highly coupled, i.e., no reason to penalize one equation's residual more than the others. We alter β between 1e-13 and 1e-9 and for each value of β, test a few combinations of γ_1, γ_3 and α. The most accurate results obtained are presented in Table 3.

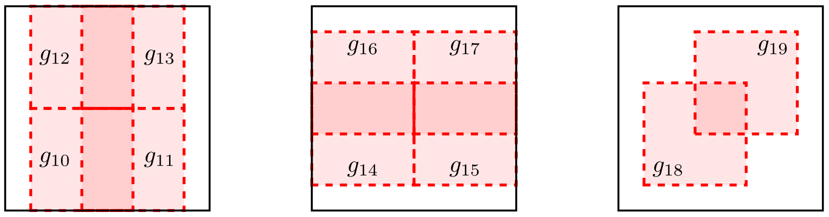
For data-only GFNet, i.e., $\alpha = 0$, reducing pressure's contribution to gradient descent improves the overall accuracy. We use $\gamma_3 = 0.5$ since further decreasing it to $\gamma_3 = 0.2$ negligibly improves the predictions for velocities while increasing the error on pressure by 27%. Next, we increase the number of samples and minimize PDE residuals in GFNet's training. Using collocation points and α = 1e-3 reduces the accuracy by more than 50%

Table 3
GFNet's accuracy in learning the NS equations: The first and second values in column one indicate the number of data and collocation points, respectively. γ_1, γ_3, α, and β are the coefficients in Eq. (8). The three validation error and loss values correspond to $u(\boldsymbol{x})$, $v(\boldsymbol{x})$, and $p(\boldsymbol{x})$ which denote the velocity components and pressure, respectively.

Points	γ_1, γ_3		α	β	Validation MAE			Validation PDE loss		
961/-	1.0	1.0	0	1e−11	1.02e−03	1.01e−03	6.49e−04	–	–	–
961/-	1.0	0.5	0	1e−10	6.81e−04	6.43e−04	5.51e−04	–	–	–
961/-	1.0	0.2	0	1e−10	6.42e−04	6.30e−04	7.05e−04	–	–	–
576/1200	1.0	0.5	1e−3	0	1.48e−03	1.24e−03	8.79e−04	8.30e−04	2.58e−03	4.51e−04
576/1200	1.0	0.5	1e−2	1e−10	1.88e−03	1.56e−03	9.84e−04	3.20e−04	4.57e−04	1.94e−04



(a) Ground truth (b) MF predictor , 9 genomes (c) MF predictor , 19 genomes (d) PINN



(e) The arrangement of 10 additional auxiliary genomes

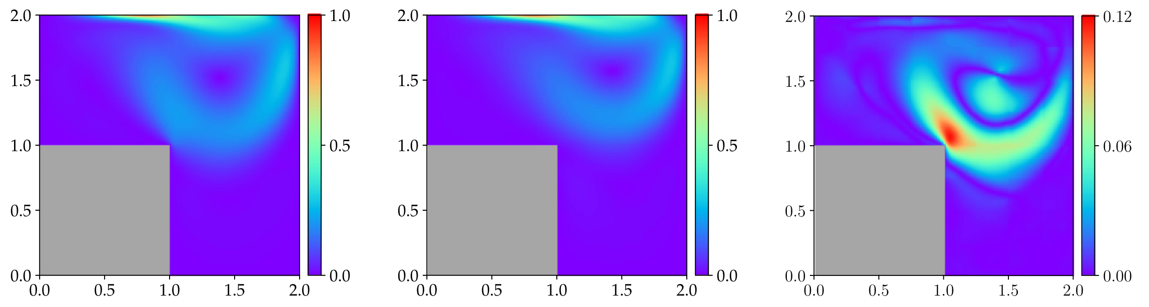
Fig. 6. (Color Online) Velocity magnitudes $\sqrt{u^2+v^2}$ for flow in a square lid-driven cavity with unseen BC: (*a*) The ground truth is simulated by OpenFOAM [82]. (*b*) MF predictor with 9 genomes fails to accurately predict the velocities. (*c*) MF predictor with 19 genomes accurately predicts the unseen conditions and achieves an MAEs of 5.68e−3 and 4.96e−3 for velocities (u, v), respectively. (*d*) PINN fails to resolve the flow and achieves MAEs 1.52e−1 and 1.39e−1 for velocities (u, v). (*e*) In addition to the 9 genomes (g_{1-9}) arranged as shown in Fig. 3(c), we add 10 more auxiliary genomes (g_{10-19}) to improve MF predictor's accuracy in resolving the unseen flow.

and further increasing α exacerbates the inaccuracies. The reasons that minimizing PDE residuals reduces accuracy are twofold. Firstly, as pointed out in [74,84,85], training physics-informed models is equivalent to solving a stiff system of ordinary differential equations which requires more advanced training techniques such as adaptive weight or adaptive activation functions. Leveraging these techniques can potentially increase the accuracy of our GFNet and will be pursued in our future works. Secondly, Eq. (8) uses AD to compute the derivatives analytically and regularizes the differential form of the NS equations while OpenFOAM solves the integral form of the NS equation. Note that the velocity and pressure from OpenFOAM are accurate but their derivatives differ from the ones obtained by analytical differentiation and dissatisfy the PDE residual in Eq. (8). For instance, OpenFOAM reports an average residual of 9e-10 for H_1 using the velocities shown in Fig. 6(a). However, the same velocities result in an average residual of 3e-4 if FD is used to calculate the derivatives in Eq. (7). As a result, the data loss and PDE residual contradict each other during gradient descent and decrease the overall accuracy of GFNet.

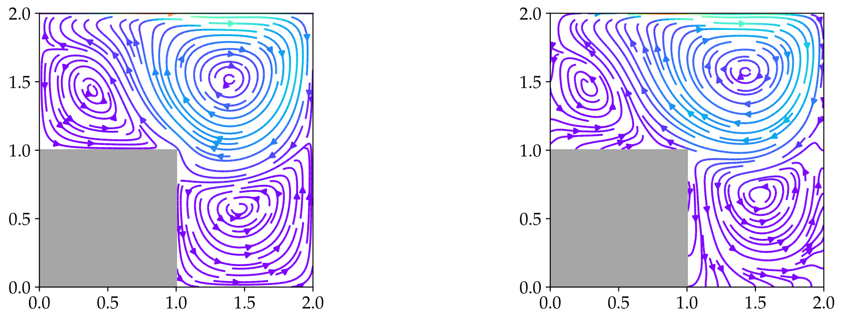
For the lid-driven cavity problem, we are interested in inferring the velocity accurately across unseen BCs and unseen domains. Therefore, we choose the model with the lowest MAE in velocity which is obtained with $\gamma_3 = 0.5$ and $\alpha = 0$.

4.2.3. Accuracy for unseen domains subject to unseen BCs and comparison with PINN

We evaluate the accuracy of MF predictor in a domain of size $[0, 1] \times [0, 1]$ which is four times larger than the genome used in training the GFNet. The unseen BC is again generated via a GP and determines the velocity profile



(a) $\sqrt{u^2+v^2}$ ground truth (b) $\sqrt{u^2+v^2}$ by MF predictor (c) Error in $\sqrt{u^2+v^2}$



(d) Streamlines ground truth (e) Streamlines by MF predictor

Fig. 7. (Color Online) Velocity magnitude $\sqrt{u^2+v^2}$ and streamlines for the flow in a step-shaped lid-driven cavity: Compared to the ground truth simulated by OpenFOAM [82], MF predictor achieves MAEs of 1.35e−2 and 1.24e−2 for velocity components u and v, respectively. The streamline plots (colored by velocity magnitude) reveal that the step induces a complex flow pattern that consists of three inter-related vortices and MF predictor successfully captures all these vortex structures while the GFNet has never seen such scenarios during training.

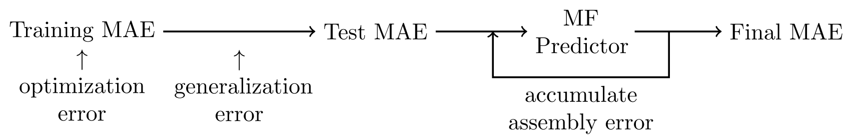


Fig. 8. Error decomposition: Our framework has two primary error sources. The genomic error is introduced in GFNet's prediction for unseen BCs while the assembly error is accumulated from the iterative procedure in MF predictor. The genomic error is divided into optimization error, generalization error, and approximation error [86–88]. The optimization error is estimated by the training MAE. The generalization error is measured by subtracting test MAE and training MAE. Given the large size of our neural networks, we assume the approximation error is negligible. The assembly error is evaluated by subtracting GFNet's test MAE from the final MAE achieved by MF predictor.

of the top lid. We first use MF predictor with 9 genomes (4 basic and 5 auxiliary) arranged as shown in Fig. 3(c) which is also used throughout Section 4.1. However, as illustrated in Fig. 6(b), using only 9 genomes does not provide sufficient accuracy and the MAE for velocity components u and v are 7.15e−2 and 6.82e−2, respectively. This is due to the highly non-linear nature of the NS equations which complicates the propagation of the boundary information. To address this issue, we include more auxiliary genomes (that overlap with the original 9 genomes) to facilitate the information propagation, see Fig. 6(e). As shown in Fig. 6(c), using 19 genomes significantly improves the accuracy and reduces the MAE of velocity components u and v to 5.68e−3 and 4.96e−3, respectively.

To compare accuracy and performance with the state-of-the-art, we construct the PINN designed in [74] to solve the lid-driven cavity problem with $u = 1$, $v = 0$ on the top lid (we were able to reproduce the accurate results reported in [74]). We re-train the model for 40 000 epochs with Adam using the architecture $50\otimes50\otimes50\otimes50\otimes50$, 4×129 boundary points, and 10 000 collocation points. As shown in Fig. 6(d), PINN is unable to resolve the flow field and the MAE for velocities (u, v) are as large as 1.52e−1 and 1.39e−1. This indicates that the architecture of PINN only applies to the specific BC used in [74], i.e., changing the BC requires not only re-training the PINN but also re-designing its architecture. Regarding computational costs, MF predictor with 19 genomes converges to an accurate solution in 117 seconds while the PINN's training takes 8846 seconds.

We further evaluate MF predictor by predicting the flow in a step-shaped lid-driven cavity that has an unseen velocity profile set to the top lid, see Fig. 7. This unseen cavity is 12 times larger than our genomes and is decomposed into 12 basic and 21 auxiliary genomes. Fig. 7 compares the results from MF predictor and the ground truth simulated by OpenFOAM [82] using the same grid resolution as in data generation. MF predictor converges in 313 seconds which is almost 3 times more than the cost for the square cavity in 6(d). This increase in inference time is consistent with the 3:1 area ratio between the two domains.

The MAE for velocities (u, v) are 1.35e−2 and 1.24e−2, respectively, which are larger than the MAE for the square cavity by 1 order of magnitude. As shown in Figs. 7(c) and 7(d), the increased error primarily comes from complex flow patterns that have never been seen by GFNet during its training: the flow over the step corner and the large vortex interleaved with two neighboring vortices on its left and bottom. Nonetheless, MF predictor still captures all three vortex structures in the cavity as shown by the streamlines in Fig. 7(e). We also observe some unphysical flows in the near-wall regions where the streamlines should be parallel with the wall. This error is caused by the inaccuracies of GFNet in predicting the flow on the boundaries, which will be further discussed in Sections 4.3 and 4.4.

4.3. Detailed error analysis and comparison with DeepONet and Fourier Neural Operator (FNO)

We attribute the error of our framework to two sources: the *genomic error* which is introduced by GFNet's inference for a single genome with an unseen BC, and the *assembly error* which is accumulated during the iterative procedure in MF predictor. The genomic error can be further divided into three parts [86–88],

$$\begin{aligned} |\mathcal{N}(\boldsymbol{g}^*, \boldsymbol{x}|\boldsymbol{\theta}) - \boldsymbol{u}^*(\boldsymbol{x})| &= |\mathcal{N}(\boldsymbol{g}^*, \boldsymbol{x}|\boldsymbol{\theta}) - \hat{\boldsymbol{u}}(\boldsymbol{x}) + \hat{\boldsymbol{u}}(\boldsymbol{x}) - \boldsymbol{u}^*(\boldsymbol{x})| \\ &\le |\boldsymbol{\delta}(\bar{\boldsymbol{g}}, \boldsymbol{x})| + |\boldsymbol{\delta}(\boldsymbol{g}^*, \boldsymbol{x}) - \boldsymbol{\delta}(\bar{\boldsymbol{g}}, \boldsymbol{x})| + |\hat{\boldsymbol{u}}(\boldsymbol{x}) - \boldsymbol{u}^*(\boldsymbol{x})|, \end{aligned} \tag{9}$$

where $\boldsymbol{u}^*(\boldsymbol{x})$ is the exact solution to a BVP with an unseen BC ($\boldsymbol{g}^*$), $\hat{\boldsymbol{u}}(\boldsymbol{x})$ represents the closest approximation to $\boldsymbol{u}^*(\boldsymbol{x})$ that a given neural network can achieve, $\bar{\boldsymbol{g}}$ is a training sample that resembles $\hat{\boldsymbol{g}}$, and $\boldsymbol{\delta}(\boldsymbol{g}, \boldsymbol{x}) = \mathcal{N}(\boldsymbol{g}, \boldsymbol{x}|\boldsymbol{\theta}) - \hat{\boldsymbol{u}}(\boldsymbol{x})$ measures the difference between $\hat{\boldsymbol{u}}(\boldsymbol{x})$ and the approximation achieved by training the neural network. In Eq. (9), the first part represents the *optimization* error introduced during training [86,87] while the second part quantifies the neural network's *generalization* error [88] in predicting unseen inputs. The last part defines the *approximation* error [86,87] which appears when the target function $\boldsymbol{u}^*(\boldsymbol{x})$ is not covered by the neural network's functional space. According to the universal approximation theorem [89,90], the approximation error effectively diminishes by increasing the size of the neural network.

In practice, it is hard to compute the above errors rigorously since $\hat{\boldsymbol{u}}(\boldsymbol{x})$ is typically unknown. Nonetheless, we can define approximate metrics to reflect the error's magnitudes, see Fig. 8. Given the large size of our neural networks which have up to 3×10^6 parameters, we assume that the approximation error is small compared to the optimization and generalization errors which are estimated by, respectively, GFNet's training MAE and the subtraction of training MAE from test MAE. We compute the test MAE of GFNet by (1) decomposing an unseen domain into basic genomes, (2) extracting the basic genomes' BCs directly from the ground truth, and (3) inferring the solution in all the basic genomes using the extracted BCs, and (4) evaluating the MAE against the ground truth across all the genomes. Note that this test MAE is evaluated without using MF predictor and it solely reveals the genomic error in predicting flows with unseen BCs. At last, we approximate the assembly error by subtracting the GFNet's test MAE from the final MAE achieved by MF predictor.

In what follows we present a detailed error analysis of our framework in solving the Laplace and the NS equations with unseen domains and BCs. We note that *the transferability feature of framework is independent of the specific architecture used in GFNet*. That is, other state-of-the-art DNN architectures such as DeepONet [33] and the Fourier neural operator (FNO) [31] can also be used to learn the underlying PDE system in the genomic domain and, in turn, used in MF predictor. With this important note in mind, in this section we compare our GFNet's performance against DeepONet and FNO at the genomic as well as large scales to obtain a deeper understanding of the error sources.

Table 4
Accuracy of different operator learners: The residual is computed with both FD (with a step of 1/256) and AD. AD-based residual cannot be calculated for FNO due to its architecture and training are set up. Only LPFC best uses collocation points in its training.

Operator learner	Train MAE	Validation MAE	Residual with FD	Residual with AD
DeepONet	2.19e−04	2.34e−04	5.61e+2	2.81e+1
FNO	6.95e−05	7.58e−05	8.48e+2	–
LPFC	1.67e−04	1.66e−04	1.85e+4	5.27e+3
LPFC Best	1.59e−4	4.10e−4	4.03e−3	1.46e−2

Table 5
Effect of GFNet on the errors: The errors are defined in Section 4.3 (see also Fig. 8) and calculated for the "Mosaic Flow" logo in Fig. 1.

GFNet model	Optimization error	Generalization error	Assembly error	Final MAE
DeepONet	2.19e−04	6.87e−04	1.97e−03	2.87e−03
FNO	6.95e−05	5.81e−04	4.74e−03	5.39e−03
LPFC	1.67e−04	4.80e−04	1.98e−03	2.79e−03
LPFC Best	1.59e−4	8.04e−04	1.80e−03	2.61e−03

4.3.1. Laplace equation

We train and compare LPFC against DeepONet and FNO where in both cases we test different architectures and optimization hyper-parameters and select the optimum one. The optimized DeepONet is an unstacked DeepONet with a branch network consisting of 6 fully connected hidden layers of sizes 100^6 and a trunk network consisting of 6 fully connected hidden layers of sizes 100^6, all using swish activation functions (see [33] for more details on the architecture). The optimized FNO has an architecture with width 64 and 18 modes (see [91] for more details). The training procedure used in these two operator learners includes a total of 10 000 Laplace solutions from which 8000 are used in training while 2000 employed in validation. For each sample, we include 900 data points (placed on a regular grid), 124 boundary condition points, and no collocation points. The reason that collocation points are not used in training any of the models compared in this subsection is that FNO cannot be trained using AD because it predicts on a pre-selected grid which is fixed during training. Hence, the output of FNO is not single-valued and the AD-based residual of a single point is not properly defined. Thus, for fair comparison, we retrain the LPFC architecture with the same setup explained above. The accuracy of these three models are also evaluated against the best LPFC architecture of Section 4.1.4 (which uses AD-based residuals in training) to investigate the effect of collocation points on the performance.

The training results are summarized in Table 4 and indicate that in this experiment FNO achieve the best MAE results in both training and validation. We note that while our best LPFC architecture presents the worst validation error, it outperforms the other models in terms of the PDE residual. This result is a direct consequence of training it with collocation points but less labeled data points. Additionally, we attribute the better performance of LPFC over DeepONet to two different reasons: (1) LPFC can be considered as a simplified version of DeepONet and therefore is easier to train, and (2) LPFC preserves the linearity of the boundary conditions which is a key feature of the Laplace equation.

Next, we evaluate the performance of these models when used in MF predictor . We perform this test on our largest domain that represents the spelling of "Mosaic Flow", see Fig. 1. The results are summarized in Table 5 and indicate interesting features about the error sources. We observe that the training and validation errors shown in Table 4 slightly correlate with the generalization error, i.e., the model with lower training/validation error has smaller generalization error compared to other models. However, this trend is not observed in the assembly error, i.e., a lower generalization error does not result in a smaller assembly error. A closer look at Tables 4 and 5 indicates that the best assembly error is obtained for the model that achieves smallest residual error, i.e., the LPFC architecture which employs collocation points in its training. In fact, the largest assembly error is obtained by the FNO architecture whose FD-based residual diverges as the FD step size decreases. From these observations we conclude that when MF predictor is used in large domains the PDE residuals at the genome scale greatly contribute to the final MAE and hence the trained GFNet must achieve not only a small MAE in training/validation, but also a small PDE residual.

Table 6
Accuracy of different GFNets for learning the NS equations: We compare four different GFNet implementations that include the fully connected (FC) network, DeepONet, fourier neural operator (FNO), and FC with the enforcement of the input BC (FC+ BC). The generalization and assembly error are evaluated on the step-shaped cavity. Each error includes two entries for the two velocity components which are u and v, respectively. DeepONet and FNO achieve lower optimization errors but comparable/worse generalization and assembly errors. Enforcing the exact BC not only improves GFNet's training and generalization errors but also reduces the assembly error in MF predictor.

GFNet model	Optimization error		Generalization error		Assembly error		Final MAE	
	u	v	u	v	u	v	u	v
DeepONet	6.26e−4	5.20e−4	1.86e−3	2.00e−3	6.30e−2	6.98e−2	6.48e−2	7.19e−2
FNO	4.01e−4	3.87e−4	2.39e−3	1.76e−3	5.02e−2	2.74e−2	5.30e−2	3.31e−2
FC	6.74e−4	6.54e−4	1.87e−3	1.65e−3	1.10e−2	1.02e−2	1.35e−2	1.24e−2
FC+ BC	4.16e−4	3.55e−4	1.15e−3	1.20e−3	6.51e−3	8.44e−3	8.07e−3	1.00e−2

Table 7
Residual comparison of different models for learning the NS equations: We compare the PDE residuals of three GFNets where FC outperforms the other models by 1–2 orders of magnitudes. The calculations are performed on validation data from the training.

GFNet model	Residual with FD			Residual with AD		
	H_1	H_2	H_3	H_1	H_2	H_3
DeepONet	4.19e−02	6.65e+00	9.24e−02	1.79e−02	9.79e−02	2.29e−03
FNO	3.34e−01	5.73e−01	3.72e−01	–	–	–
FC	2.30e−03	5.96e−03	4.27e−04	7.62e−04	2.96e−04	1.37e−04
FC+BC	4.972e−01	1.179e−01	9.335e−03	1.049e−01	1.960e−03	2.820e−03

4.3.2. Navier–Stokes

We repeat the analyses of previous section for the NS equations. In particular, we build two GFNets using the DeepONet and FNO architectures using the training data introduced in Section 4.2.2 (2634 samples with 961 data points per sample and no collocation points). We test all GFNets within MF predictor to estimate the flow in the step-shaped cavity.

Rows one through three in Table 6 enumerate the breakdown of errors where FNO achieves the lowest optimization error while FC slightly outperforms the other two models in terms of the generalization error. Compared to the results in Table 5, the ratio between generalization and optimization errors has increased because (1) compared to the Laplace equation, the NS equations are much more complex and have higher dimensionality, and (2) the test samples used for calculating the generalization error are extracted from the step-shaped cavity which not only has unseen BCs but also contains very complex unseen flow features, i.e., the flow over the sharp corner and the interleaved vortex structure (see Fig. 7(d)). These features adversely affect the performance of all models in this experiment.

In terms of the assembly and final errors, the GFNet based on FC accumulates the least error. This observation aligns with the results reported in the previous section for the Laplace equation: FC achieves smaller residuals compared to DeepONet and FNO (see Table 7) and hence it provides smaller assembly error. While this observation motivates the use of collocation points in training a GFNet, for the reasons argued in Section 4.2.2, we do not use residuals in approximating the solution of the NS equations. We will research effective ways to incorporate PDE residuals in learning the NS equations in our future work.

4.3.3. Assembly error and genome density

In the previous section we showed that the final MAE of the MF predictor depends on the generalization and assembly errors where the latter error source is directly affected by the PDE residuals of the GFNet. In Section 4.2.3, we demonstrated that the genome density also affects the assembly error where using more auxiliary genomes improved the predicted flow in the single cavity domain, see Fig. 6. Following this observation, we examine the effect of genome density on the assembly error in this section.

To consistently evaluate assembly error against genome density, we evaluate the effect of placing extra layers of auxiliary genomes that overlap with the borders of the previously added auxiliary layers. Following the descriptions

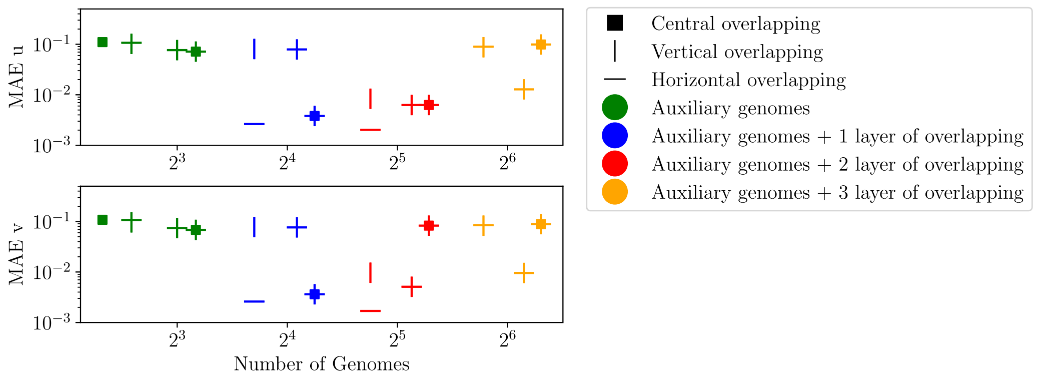


Fig. 9. (Color Online) MF predictor accuracy vs. genome density for the single cavity domain: The accuracy of MF predictor is evaluated under different arrangements and numbers of auxiliary genomes. Here, green indicates that only a single layer of auxiliary genomes is used. These genomes are placed over the borders of the basic genomes and can be centered on the vertical borders (e.g., genomes 5 and 6 in Fig. 3(c)), horizontal borders (e.g., genomes 7 and 8 in Fig. 3(c)), corner of 4 genomes (e.g., genome 9 in Fig. 3(c)), or at any combination of these three configurations. The color blue shows that an extra layer of overlapping auxiliary genomes is placed at the vertical, horizontal, or central borders of the existing auxiliary genomes (e.g., Fig. 6(e)). The colors red and orange indicate, respectively, that two and three extra layers of overlapping auxiliary genomes are used.

in Section 3.3, we divide each layer of these auxiliary genomes into three different categories based on the borders of the existing auxiliary genomes: we distinguish between genomes that overlap with (i) the vertical borders (e.g., genomes 5 and 6 in Fig. 3(c) or genomes 10, 11, 12 and 13 in Fig. 6(e)), (ii) the horizontal borders (e.g., genomes 7 and 8 in Fig. 3(c) or genomes 14, 15, 16 and 17 in Fig. 6(e)), and (iii) the corners where 4 basic genomes reside (e.g., genome 9 in Fig. 3(c) or genomes 18 and 19 in Fig. 6(e)). As different combinations of these additional layers leads to different overall genome arrangements (which, in turn affects the iterative update procedure), we add each of these layers in several steps: adding only central overlapping genomes, adding only vertical or only horizontal genomes, adding both vertical and horizontal, and adding all genomes (i.e., vertical, horizontal and central) at once.

We conduct this experiment on the single cavity domain for the NS equation using an FC architecture as a GFNet. The results are summarized in Fig. 9 and demonstrate two interesting trends. (1) There is a local minimum in the MAE which appears as a result of balancing propagated errors and boundary information. That is, as we increase the number of genomes the both the BCs and errors propagate to the interior of the domain and after the local minimum the propagated errors dominate the transferred information. (2) The extra horizontal overlapping genomes improve the accuracy of MF predictor for the single cavity domain more than the other genomes. We attribute this observation to the fact that the flow is predominately affected by the applied lid-velocity at the top of the domain which means that the information is mostly propagated downwards.

Locating the abovementioned optimum has two primary challenges. First, the optimum configuration is problem dependent. For instance, adding extra layers for the step-shape cavity domain would add extra assembly errors and increase final MAEs regardless of the employed arrangement. Second, computational costs increase as more genomes are employed in MF Predictor. In our experiments, we have observed that the simple arrangement described in Section 3.3 consistently outperforms other arrangements (either simpler or more complex) and therefore we recommend it as the default technique. In our future work we will research automatic and adaptive techniques to arrange the basic and auxiliary genomes.

4.4. Enforcing exact boundary conditions in GFNet

For the step-shaped cavity, GFNet yields inaccurate predictions near the walls where the streamline should be parallel with the boundaries, see Figs. 7(d) and 7(e). This is because GFNet applies the BCs by penalizing the MSE at the boundary points (see Fig. 4), i.e., the input BCs are not strictly enforced. The BC for stationary walls is $u = v = 0$ where even very small errors greatly affect the velocity direction and cause unphysical streamlines. To reduce this error, which will improve the accuracy of both GFNet and MF predictor, we design the architecture of GFNet to strictly reproduce the input BCs.

To design a GFNet that reproduces a BC exactly, we follow [92] and decompose the approximated solution into three parts:

$$u(\boldsymbol{x}) = G(\boldsymbol{x}) + \phi(\boldsymbol{x}) \cdot \mathcal{N}(\boldsymbol{x}|\boldsymbol{\theta}), \tag{10}$$

where $G(\boldsymbol{x})$ is a smooth function defined on Ω that satisfies the BC, i.e., $G(\boldsymbol{x}) = g(\boldsymbol{x}), \boldsymbol{x} \in \partial\Omega$, $\phi(\boldsymbol{x})$ is a smooth function that is equal to zero on $\partial\Omega$, and $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\theta})$ denotes the neural network which can be implemented with any of the models explored in Section 4.3. $G(\boldsymbol{x})$ can be constructed by either extrapolating $g(\boldsymbol{x})$ from $\partial\Omega$ to Ω [93] or training a neural network that interpolates the boundary points [92]. To avoid masking the network, $\phi(\boldsymbol{x})$ is recommended to be nonzero inside Ω. The typical choice of $\phi(\boldsymbol{x})$ is the signed distance function that measures the distance from $\boldsymbol{x} \in \Omega$ to $\partial\Omega$ [92–94].

Note that Eq. (10) only solves the PDE with a specific BC on one domain. We make the following adjustments to enforce the input BC in GFNet and retain its transferability across unseen BCs. First, our model takes both $\boldsymbol{g}$ and $\boldsymbol{x}$ as inputs. We follow [93] to extrapolate $\boldsymbol{g}$ by weighted averaging,

$$G(\boldsymbol{g}, \boldsymbol{x}) = \sum_{i=1}^{N_{bc}} \frac{|\boldsymbol{x} - \boldsymbol{x}_i^{bc}|^{-2} \boldsymbol{g}_i}{\sum_{j=1}^{N_{bc}} |\boldsymbol{x} - \boldsymbol{x}_j^{bc}|^{-2}}, \tag{11}$$

where $|\boldsymbol{x} - \boldsymbol{x}_i^{bc}|$ represents the distance between $\boldsymbol{x} \in \Omega$ and the ith boundary point and $\boldsymbol{g}_i = g(\boldsymbol{x}_i^{bc})$. In practice, we add a small constant $\epsilon = 10^{-10}$ to the distance to avoid division by zero. Second, given that GFNet only infers flows in square genomes, we simply choose $\phi(\boldsymbol{x}) = x(l - x)y(l - y)$ where l is the square genome's edge length. Finally, we apply the neural network defined in Section 3.2, $\mathcal{N}(\boldsymbol{g}, \boldsymbol{x}|\boldsymbol{\theta})$, to Eq. (10) to achieve transferability across unseen BCs.

We now approximate the solution of the NS equations with a FC-based GFNet that enforces the input BC as described above. We denote this GFNet as FC+ BC and use the training data and the hyperparameters' tuning strategies explained in Section 4.2. In our tests, the architecture $400^6 \otimes 3$ delivers the best accuracy, improving GFNet's training MAE of u and v by 62% and 84%, respectively, see Table 6. It also shows that FC+BC not only decreases the optimization and generalization errors but also reduces the assembly error by an order of magnitude. To explain this improvement, we revisit the example in Fig. 2(c) where the two genomes Ω_1 and Ω_2 share the BC at border $\partial\Omega_1 \cap \partial\Omega_2$. For FC-based GFNet, its predictions (u_1 and u_2 in Fig. 2(c)) based on BCs from $\partial\Omega_1$ and $\partial\Omega_2$ can differ at the shared border. This discrepancy results in a discontinuous solution per iteration and contributes to the assembly error accumulated in MF predictor. Enforcing the input BC in GFNet eliminates the discrepancy at the shared border and therefore drastically reduces the assembly error. Similar to the FC-based GFNet, the assembly error of MF predictor with FC+ BC mainly comes from the unseen flow features. Nonetheless, imposing the exact BC drastically reduces the unphysical errors in streamlines, which become parallel with the boundaries in most of the near-wall regions, see Fig. 10(c).

For elliptic PDEs with non-Dirichlet BCs, we note that FC+ BC can be easily extended to enforce Neumann/Robin BCs and when used in MF predictor the assembled inference converges to the PDE's exact solution. We refer the readers to Appendix B and [93] for more details.

5. Conclusions

We demonstrated that a well-designed and well-trained deep neural network that takes BCs as inputs (GFNet), coupled with a novel iterative algorithm for assembling its predictions (MF predictor) can result in a *transferable* deep learning framework for operator learning. To the best of our knowledge, such a framework is the first of its kind. The main advantage of this framework is that the model, GFNet, needs to be trained only once and can then be used forever without re-training to solve boundary value problems in larger and complex domains with unseen sizes, shapes, and BCs.

Our framework demonstrates the capability to infer the solution of Laplace and Navier–Stokes equations on domains that are 1200× and 12× larger that the training domains, respectively. Such scalable predictions are achieved without any re-training. Compared with the state-of-the-art PINN for both PDEs, we demonstrate a remarkable 1-3 orders of magnitude speedups while achieving comparable or better accuracy across a range of

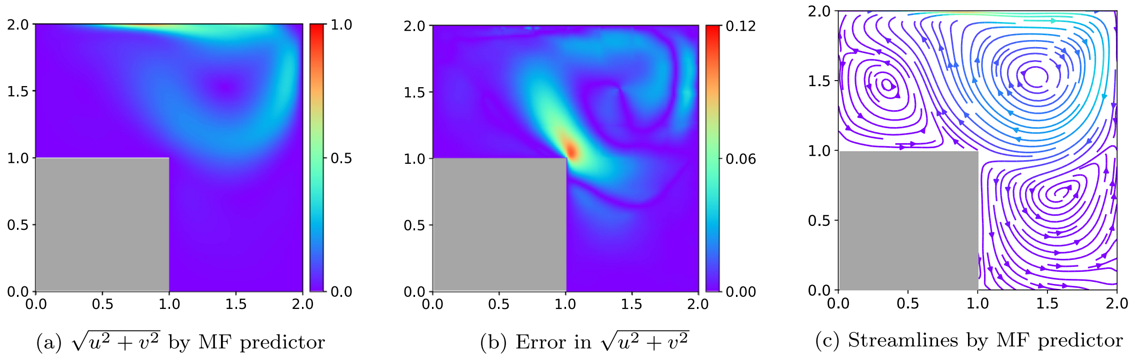


Fig. 10. (Color Online) Velocity magnitude $\sqrt{u^2+v^2}$ and streamlines for the step-shaped lid-driven cavity using the GFNet with exact BC enforced. Compared to the GFNet using FC, enforcing the exact BC improves MAEs to 8.07e−3 and 1.00e−2 for velocity components u and v, respectively. The error plot shows that the errors still primarily from the sharp corner and the interleaved vortex structures. The streamline plot reveals that enforcing the exact BC significantly diminishes the unphysical errors in the near-wall regions, as the streamlines are largely parallel with the boundaries.

BCs and domains unseen during training. Moreover, we present an in-depth error analysis of our framework, based on which we compare our GFNet with DeepONet and FNO and demonstrate that our model can effectively reduce the assembly error accumulated in MF predictor. At last but not least, we improve GFNet to exactly reproduce the input BC, which further reduces all types of errors by up to 84%. We anticipate this research will open new directions for physics-informed surrogate modeling in computational sciences and engineering.

More work remains to be done for GFNet to be widely used for a larger class of problems in engineering. For instance, our studies employed square genomes which cannot accurately represent curved objects such as airfoils. In addition, we only considered steady flows and boundary value problems whose right-hand-side function can be either zero or a periodic function whose period in each direction equals the size of the genome. The latter choice implies that the BVP solution on one genome is invariant to the genome's position in a large domain which is not a valid assumption for PDEs with space-dependent forcing terms. Moreover, the training of GFNet in this paper is a compromise between accuracy and limited time and GPU resources we can access. We observed a 20% improvement in the accuracy of GFNet for the Laplace equation when we trained the model using all the 32×32 data points. However, this will dramatically increase the training cost and prohibit us from fine-tuning the model with 18 000 samples in a reasonable amount of time. To further improve MF predictor's flexibility and successfully apply our approach to complex geometries and unsteady processes, we can embed boundary coordinates, time, and initial conditions as inputs. However, such a large input layer will require a GFNet with too many parameters which will increase the training complexities and costs. To consider general non-homogeneous PDEs we anticipate using multiple GFNets to learn the effect of forcing terms on the solution. Addressing these challenges and leveraging distributed training/inference is an important future exploration.

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.

Acknowledgments

We appreciate the support from National Science Foundation (award numbers 2045322 and OAC-2103708) as well as Advanced Research Projects Agency-Energy (award number DE-AR0001209). We would also like to thank the two anonymous reviewers whose comments greatly helped us in improving the quality of our work.

Appendix A. Non-smooth boundary conditions

In Section 4.3 we elaborated on the different error sources in our framework faces where the two most dominant ones were the generalization and the assembly errors. In those analyses we employed smooth and differentiable BCs since all our models were trained with such BCs. In this section we quantify the generalization error for non-smooth BCs. We evaluate the accuracy of the models described in Section 4.3 by solving BVPs for the Laplace equation subject to non-smooth BCs. We restrict this evaluation to domains of area 1 to prevent any bias that might arise by the smoothing effect of the Laplacian. We consider 100 non-smooth BCs which are generated with a GP whose kernel is a power exponential with randomly chosen power in the range $[1, 2)$. Five sample BCs are demonstrated in Fig. A.11.

The results are summarized in Table A.8 and indicate that the accuracy of all models has reduced compared to Table 4. This accuracy reduction is due to the fact that all models have only seen smooth BCs during training. In Table A.8 the FNO model has the best generalization error and it is followed by the LPFC and DeepONet models. We believe this trend is because FNO predicts the solution for the entire genome at once (rather than a particular spatial location in the genome) which makes it less sensitive to sharp changes in the BCs. In the case of residuals, the LPFC model trained with the PDE loss outperforms the other operator learners. A better performance in these cases can be achieved by models that enforce BCs which we have studied for the NS equation in Section 4.4.

Appendix B. Neumann and Robin boundary condition

In this paper we only present tests where the Laplace and NS equations are subject to Dirichlet BCs. Nonetheless, our framework can also predict PDE solutions with Neumann and Robin BCs for the following two reasons. First, the Schwarz method that our framework is based on can solve elliptic BVPs with Dirichlet, Neumann, and Robin BCs [69–71].

Second, in our framework we replace the numerical method used in the Schwarz method to solve genome-wise BVPs with GFNet. The GFNet that enforces the input BC (Eq. (10)) can be extended to solve PDE with Neumann and Robin BCs [93]. To strictly reproduce the Robin BC, we can follow [93] and decompose the approximated

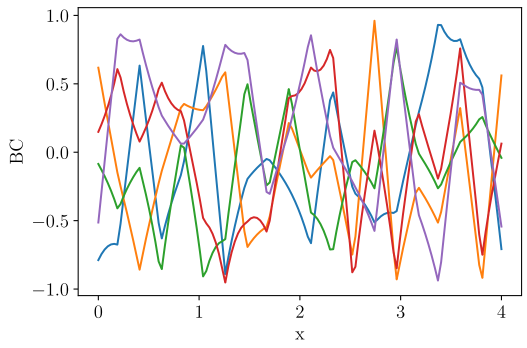


Fig. A.11. (Color Online) Sample non-smooth boundary conditions: To quantify the accuracy of the operator learners trained in Section 4.3, a test dataset with 100 non-smooth BCs is created where the AMG solver is used to solve the Laplace equation.

Table A.8
Accuracy of different GFNets for non-smooth BCs: 100 random non-smooth BCs are created to evaluate LPFC, LPFC best, FNO, and DeepONet models. We report the test MAE and the residual computed with either FD with a step size of 1/32 or AD (training MAE and the validation MAE based on smooth BCs are provided in Table 4).

Operator learner	Test MAE	Residual with FD	Residual with AD
DeepONet	4.07e−2	8.38e+1	8.58e+1
FNO	7.57e−03	1.78e+1	–
LPFC	1.90e−02	4.88e+1	5.47e+1
LPFC Best	1.88e−02	1.70e−2	5.4e−3

solution as:

$$u(\boldsymbol{x}) = [1 + \phi(c + \boldsymbol{n} \cdot \nabla)] \, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\theta}) - \phi g \tag{B.1}$$

where $\phi(\boldsymbol{x})$ is the signed distance function measuring the distance between $\boldsymbol{x} \in \Omega$ to $\partial\Omega$, the underlying Robin BC is $\boldsymbol{n} \cdot \nabla u + cu = g$ at $\partial\Omega$, and $\boldsymbol{n} = -\nabla\phi$ is outward unit normal vector. When $c = 0$, the Robin BC degrades to Neumann BC. We note that Eq. (B.1) simplifies the original formulation in [93] by omitting the second approximation function (see Equation 25 in [93]). We prove that Eq. (B.1) satisfies the Robin BC as follows,

$$\begin{aligned} \boldsymbol{n} \cdot \nabla u + cu \Big|_{\partial\Omega} &= \boldsymbol{n} \cdot \nabla\mathcal{N} + c\mathcal{N}\boldsymbol{n} \cdot \nabla\phi + (\boldsymbol{n} \cdot \nabla\phi)\boldsymbol{n} \cdot \nabla\mathcal{N} - g\,\boldsymbol{n} \cdot \nabla\phi + c\mathcal{N} \\ &= \boldsymbol{n} \cdot \nabla\mathcal{N} - c\mathcal{N} - \boldsymbol{n} \cdot \nabla\mathcal{N} + g + c\mathcal{N} = g. \end{aligned} \tag{B.2}$$

In the above, the first line is based on $\phi = 0$ at $\partial\Omega$ and the second line is derived with $\boldsymbol{n} \cdot \nabla\phi = -\boldsymbol{n} \cdot \boldsymbol{n} = -1$. Despite Eq. (B.1)'s complex form, it can be trivially implemented with a neural network. We refer the readers to [93] for the details on constructing $\phi(\boldsymbol{x})$.

References

[1] R.W. Fox, A.T. McDonald, J.W. Mitchell, Fox and McDonald's Introduction To Fluid Mechanics, John Wiley & Sons, 2020.
[2] F.P. Incropera, A.S. Lavine, T.L. Bergman, D.P. DeWitt, Fundamentals of Heat and Mass Transfer, Wiley, 2007.
[3] T. Belytschko, W.K. Liu, B. Moran, K. Elkhodary, Nonlinear Finite Elements for Continua and Structures, John wiley & sons, 2013.
[4] E. Kreyszig, Advanced Engineering Mathematics, John Wiley & Sons, 2010.
[5] J. Slotnick, A. Khodadoust, J. Alonso, D. Darmofal, W. Gropp, E. Lurie, D. Mavriplis, CFD Vision 2030 Study: A Path To Revolutionary Computational Aerosciences, Tech. Rep., 2014.
[6] M.C. Kennedy, A. O'Hagan, Predicting the output from a complex computer code when fast approximations are available, Biometrika 87 (1) (2000) 1–13, http://dx.doi.org/10.1093/biomet/87.1.1.
[7] R. Planas, N. Oune, R. Bostanabad, Evolutionary gaussian processes, J. Mech. Des. (2021).
[8] C.E. Rasmussen, Gaussian Processes for Machine Learning, 2006.
[9] E. Alpaydin, Introduction To Machine Learning, MIT Press, 2014.
[10] T. Therneau, B. Atkinson, B. Ripley, M.B. Ripley, Rpart: Recursive partitioning and regression trees, 2014.
[11] R. Bostanabad, Reconstruction of 3d microstructures from 2d images via transfer learning, Comput. Aided Des. 128 (2020) 102906, http://www.sciencedirect.com/science/article/pii/S0010448520300993.
[12] Y.-C. Chan, F. Ahmed, L. Wang, W. Chen, Metaset: Exploring shape and property spaces for data-driven metamaterials design, J. Mech. Des. 143 (3) (2021).
[13] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[14] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, http://dx.doi.org/10.1038/nature14539, https://www.ncbi.nlm.nih.gov/pubmed/26017442.
[15] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, M. Bessa, Deep learning predicts path-dependent plasticity, Proc. Natl. Acad. Sci. U S A 116 (52) (2019) 26414–26420, http://dx.doi.org/10.1073/pnas.1911815116, https://www.ncbi.nlm.nih.gov/pubmed/31843918.
[16] S. Rasp, M.S. Pritchard, P. Gentine, Deep learning to represent subgrid processes in climate models, Proc. Natl. Acad. Sci. 115 (39) (2018) 9684–9689.
[17] S. Saha, Z. Gan, L. Cheng, J. Gao, O.L. Kafka, X. Xie, H. Li, M. Tajdari, H.A. Kim, W.K. Liu, Hierarchical deep learning neural network (hidenn): An artificial intelligence (ai) framework for computational science and engineering, Comput. Methods Appl. Mech. Engrg. 373 (2021) 113452, http://dx.doi.org/10.1016/j.cma.2020.113452, https://www.sciencedirect.com/science/article/pii/S004578252030637X.
[18] Y. Suh, R. Bostanabad, Y. Won, Deep learning predicts boiling heat transfer, Sci. Rep. 11 (1) (2021) 5622, http://dx.doi.org/10.1038/s41598-021-85150-4.
[19] L. Wang, Y.-C. Chan, F. Ahmed, Z. Liu, P. Zhu, W. Chen, Deep generative modeling for mechanistic-based learning and design of metamaterial systems, Comput. Methods Appl. Mech. Engrg. 372 (2020) 113377.
[20] H. You, Y. Yu, N. Trask, M. Gulian, M. D'Elia, Data-driven learning of nonlocal physics from high-fidelity synthetic data, Comput. Methods Appl. Mech. Engrg. 374 (2021) 113553, http://dx.doi.org/10.1016/j.cma.2020.113553, https://www.sciencedirect.com/science/article/pii/S0045782520307386.
[21] F. Chollet, Deep Learning with Python, Manning Publications Co., 2017.
[22] R. Bostanabad, Y.C. Chan, L.W. Wang, P. Zhu, W. Chen, Globally approximate gaussian processes for big data with application to data-driven metamaterials design, J. Mech. Des. 141 (11) (2019) http://dx.doi.org/10.1115/1.4044257.
[23] J.R. Gardner, G. Pleiss, D. Bindel, K.Q. Weinberger, A.G. Wilson, Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration, 2018, arXiv preprint arXiv:1809.11165.
[24] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic Differentiation in Pytorch, 2017.
[25] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045.

[26] A. Griewank, On automatic differentiation, Math. Prog. Recent Dev. Appl. 6 (6) (1989) 83–107.
[27] M. Raissi, P. Perdikaris, G.E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030, http://dx.doi.org/10.1126/science.aaw4741, https://www.ncbi.nlm.nih.gov/pubmed/32001523.
[28] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (xpinns): A generalized space–time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041.
[29] C.M. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, K. Kashinath, M. Mustafa, H.A. Tchelepi, P. Marcus, A. Anandkumar, et al., MeshfreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework, 2020.
[30] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Hp-vpinns: Variational physics-informed neural networks with domain decomposition, 2020, arXiv preprint arXiv:.05385.
[31] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, 2020, arXiv preprint arXiv:2003.03485.
[32] X. Meng, Z. Li, D. Zhang, G.E. Karniadakis, Ppinn: Parareal physics-informed neural network for time-dependent pdes, Comput. Methods Appl. Mech. Eng. 370 (2020) 113250.
[33] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (3) (2021) 218–229.
[34] H.A. Schwarz, Esammelte Mathematische Abhandlungen, Vol. 260, American Mathematical Soc., 1972.
[35] K. Greff, R.K. Srivastava, J. Koutník, B.R. Steunebrink, J. Schmidhuber, Lstm: A search space odyssey, IEEE Trans. Neural Netw. Learn. Syst. 28 (10) (2017) 2222–2232, http://dx.doi.org/10.1109/TNNLS.2016.2582924, https://www.ncbi.nlm.nih.gov/pubmed/27411231.
[36] J. Hochreiter, Seppassd schmidhuber, long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780, http://dx.doi.org/10.1162/neco.1997.9.8.1735, https://www.ncbi.nlm.nih.gov/pubmed/9377276.
[37] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: International Conference on Machine Learning, pp. 1310–1318.
[38] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.
[39] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer, pp. 234–241.
[40] C. Jiang, K. Kashinath, P. Marcus, Enforcing physical constraints in cnns through differentiable pde layer, in: ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations.
[41] K. Wu, D. Xiu, Data-driven deep learning of partial differential equations in modal space, J. Comput. Phys. 408 (2020) 109307.
[42] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 481–490.
[43] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Comput. Mech. 64 (2) (2019) 525–545, http://dx.doi.org/10.1007/s00466-019-01740-0.
[44] O. Obiols-Sales, A. Vishnu, N. Malaya, A. Chandramowlishwaran, Cfdnet: a deep learning-based accelerator for fluid simulations, in: Proceedings of the International Conference on Supercomputing, 2020, http://dx.doi.org/10.1145/3392717.3392772, arXiv:2005.04485.
[45] N. Wandel, M. Weinmann, R. Klein, Fast fluid simulations in 3D with physics-informed deep learning, 2020, pp. 1–10, arXiv, arXiv:2012.11893.
[46] N. Geneva, N. Zabaras, Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks, J. Comput. Phys. 403 (2020) 109056, http://dx.doi.org/10.1016/j.jcp.2019.109056.
[47] S. Mo, Y. Zhu, N. Zabaras, X. Shi, J. Wu, Deep convolutional encoder–decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media, Water Resour. Res. 55 (1) (2019) 703–728.
[48] Y. Zhu, N. Zabaras, BayesIan deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification, J. Comput. Phys. 366 (2018) 415–447, http://dx.doi.org/10.1016/j.jcp.2018.04.018, http://www.sciencedirect.com/science/article/pii/S0021999118302341.
[49] W. Dong, J. Liu, Z. Xie, D. Li, Adaptive neural network-based approximation to accelerate eulerian fluid simulation, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019, pp. 1–22.
[50] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1251–1258.
[51] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[52] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826.
[53] N. Trask, R.G. Patel, B.J. Gross, P.J. Atzberger, Gmls-nets: A framework for learning from unstructured data, 2019, arXiv preprint arXiv:1909.05371.
[54] N. Geneva, N. Zabaras, Quantifying model form uncertainty in reynolds-averaged turbulence models with bayesian deep neural networks, J. Comput. Phys. 383 (2019) 125–147, http://dx.doi.org/10.1016/j.jcp.2019.01.021.
[55] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech. 807 (2016) 155–166, http://dx.doi.org/10.1017/jfm.2016.615.
[56] E. Weinan, B. Yu, The deep ritz method: A deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 6 (1) (2018) 1–12, http://dx.doi.org/10.1007/s40304-018-0127-z.
[57] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational physics-informed neural networks for solving partial differential equations, 2019, arXiv preprint arXiv:.00873.
[58] G. Pang, L. Lu, G.E. Karniadakis, Fpinns: Fractional physics-informed neural networks, SIAM J. Sci. Comput. 41 (4) (2019) A2603–A2626, http://dx.doi.org/10.1137/18M1229845.

[59] B. Deng, Y. Shin, L. Lu, Z. Zhang, G.E. Karniadakis, Convergence rate of deeponets for learning operators arising from advection-diffusion equations, 2021, arXiv preprintarXiv:2102.10621.

[60] S. Lanthaler, S. Mishra, G.E. Karniadakis, Error estimates for deeponets: A deep learning framework in infinite dimensions, 2021, arXiv preprint arXiv:2102.09618.

[61] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (4) (1995) 911–917.

[62] A.G. Özbay, S. Laizet, P. Tzirakis, G. Rizos, B. Schuller, Poisson CNN: Convolutional Neural Networks for the Solution of the Poisson Equation with Varying Meshes and Dirichlet Boundary Conditions, 2019, pp. 1–36, arXiv:1910.08613.

[63] R. Maulik, H. Sharma, S. Patel, B. Lusch, E. Jennings, Accelerating RANS turbulence modeling using potential flow and machine learning, 2019, pp. 1–21, arXiv:1910.10878.

[64] S. Shahane, P. Kumar, S.P. Vanka, Convolutional Neural Network for Flow over Single and Tandem Elliptic Cylinders of Arbitrary Aspect Ratio and Angle of Attack, 2020, arXiv . arXiv:2012.10768.

[65] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating eulerian fluid simulation with convolutional networks, in: 34th International Conference on Machine Learning, 2017, (7) ICML, 2017, pp. 5258–5267, arXiv:1607.03597.

[66] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning, pp. 41–48.

[67] K.A. Krueger, P. Dayan, Flexible shaping: How learning in small steps helps, Cognition 110 (3) (2009) 380–394, http://dx.doi.org/10.1016/j.cognition.2008.11.014.

[68] L.N. Olson, J.B. Schroder, PyAMG: Algebraic Multigrid Solvers in Python V4.0, 2018, https://github.com/pyamg/pyamg.

[69] J. Xu, Iterative methods by space decomposition and subspace correction, SIAM Rev. 34 (4) (1992) 581–613.

[70] A. Toselli, O. Widlund, Domain Decomposition Methods-Algorithms and Theory, Vol. 34, Springer Science & Business Media, 2004.

[71] T. Mathew, Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations, Vol. 61, Springer Science & Business Media, 2008.

[72] E.B. Martín Abadi, Ashish. Agarwal, Paul. Barham, A.D. Zhifeng Chen, Craig. Citro, Greg S. Corrado, I.G. Jeffrey Dean, Matthieu. Devin, Sanjay. Ghemawat, Y.J. Andrew Harp, Geoffrey. Irving, Michael. Isard, Rafal. Jozefowicz, M.S. Lukasz Kaiser, Manjunath. Kudlur, Josh. Levenberg, Dan. Mané, J.S. Rajat Monga, Sherry. Moore, Derek. Murray, Chris. Olah, P.T. Benoit Steiner, Ilya. Sutskever, Kunal. Talwar, F.V. Vincent Vanhoucke, Vijay. Vasudevan, M.W. Oriol Vinyals, Pete. Warden, Martin. Wattenberg, Yuan. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015, http://tensorflow.org/.

[73] D.P. Kingma, J.L. Ba, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, 2015, pp. 1–15, arXiv:1412.6980.

[74] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020, pp. 1–28, arXiv:2001.04536.

[75] I.M. Sobol, On quasi-monte carlo integrations, mathematics and computers in simulation, 47, (2–5) 1998, pp. 103–112, http://dx.doi.org/10.1016/S0378-4754(98)00096-2.

[76] X. Jin, S. Cai, H. Li, G.E. Karniadakis, Nsfnets (Navier–Stokes flow nets): Physics-informed neural networks for the incompressible Navier–Stokes equations, J. Comput. Phys. 426 (2021) 109951, http://dx.doi.org/10.1016/j.jcp.2020.109951, arXiv:2003.06496.

[77] D.C. Liu, J. Nocedal, On the limited memory bfgs method for large scale optimization, Math. Program. 45 (1) (1989) 503–528.

[78] P.F. Fischer, An overlapping schwarz method for spectral element solution of the incompressible navier–stokes equations, J. Comput. Phys. 133 (1) (1997) 84–101.

[79] E. Brakkee, P. Wesseling, C. Kassels, Schwarz domain decomposition for the incompressible navier–stokes equations in general co-ordinates, Internat. J. Numer. Methods Fluids 32 (2) (2000) 141–173.

[80] P.F. Fischer, J.W. Lottes, Hybrid schwarz-multigrid methods for the spectral element method: Extensions to navier-stokes, in: Domain Decomposition Methods in Science and Engineering, Springer, 2005, pp. 35–49.

[81] E. Blayo, D. Cherel, A. Rousseau, Towards optimized schwarz methods for the navier–stokes equations, J. Sci. Comput. 66 (1) (2016) 275–295.

[82] H. Jasak, A. Jemcov, Z. Tukovic, Others, OpenFOAM: A C++ Library for complex physics simulations, in: International Workshop on Coupled Methods in Numerical Dynamics, Vol. 1000, IUC Dubrovnik Croatia, 2007, pp. 1–20.

[83] S. Cai, Z. Wang, S. Wang, P. Paris, Physics-informed neural networks (PINNs) for heat transfer problems, J. Heat Transfer (2021) http://dx.doi.org/10.1115/1.4050542.

[84] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.

[85] S. Wang, X. Yu, P. Perdikaris, When and why pinns fail to train: A neural tangent kernel perspective, 2020, arXiv preprint arXiv:2007.14527.

[86] L. Bottou, O. Bousquet, The tradeoffs of large scale learning, in: Proceedings of the 20th International Conference on Neural Information Processing Systems, 2007, pp. 161–168.

[87] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[88] P. Jin, L. Lu, Y. Tang, G.E. Karniadakis, Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness, Neural Netw. 130 (2020) 85–99.

[89] K. Hornik, Approximation capabilities of multilayer neural network, Neural Netw. 4 (1991) (1991) 251–257.

[90] C. Debao, Degree of approximation by superpositions of a sigmoidal function, Approx. Theory Appl. 9 (3) (1993) 17–28, http://dx.doi.org/10.1007/BF02836480.

[91] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, 2020, arXiv preprint arXiv:2010.08895.

[92] J. Berg, K. Nyström, A unified deep artificial neural network approach to partial differential equations in complexch geometries, Neurocomputing 317 (2018) 28–41.

[93] N. Sukumar, A. Srivastava, Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks, 2021, arXiv preprint arXiv:2104.08426.

[94] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Engrg. 361 (2020) 112732.