Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2020-0164

Vol. **28**, No. 5, pp. 2002-2041
November 2020

# Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations

Ameya D. Jagtap[1,*] and George Em Karniadakis[1,2]

[1] *Division of Applied Mathematics, Brown University, 182 George Street, Providence, RI 02912, USA.*
[2] *Pacific Northwest National Laboratory, Richland, WA 99354, USA.*

Received 26 August 2020; Accepted (in revised version) 3 October 2020

**Abstract.** We propose a generalized space-time domain decomposition approach for the physics-informed neural networks (PINNs) to solve nonlinear partial differential equations (PDEs) on arbitrary complex-geometry domains. The proposed framework, named eXtended PINNs (*XPINNs*), further pushes the boundaries of both PINNs as well as conservative PINNs (cPINNs), which is a recently proposed domain decomposition approach in the PINN framework tailored to conservation laws. Compared to PINN, the XPINN method has large representation and parallelization capacity due to the inherent property of deployment of multiple neural networks in the smaller subdomains. Unlike cPINN, XPINN can be extended to any type of PDEs. Moreover, the domain can be decomposed in any arbitrary way (in space and time), which is not possible in cPINN. Thus, XPINN offers both space and time parallelization, thereby reducing the training cost more effectively. In each subdomain, a separate neural network is employed with optimally selected hyperparameters, e.g., depth/width of the network, number and location of residual points, activation function, optimization method, etc. A deep network can be employed in a subdomain with complex solution, whereas a shallow neural network can be used in a subdomain with relatively simple and smooth solutions. We demonstrate the versatility of XPINN by solving both forward and inverse PDE problems, ranging from one-dimensional to three-dimensional problems, from time-dependent to time-independent problems, and from continuous to discontinuous problems, which clearly shows that the XPINN method is promising in many practical problems. The proposed XPINN method is the generalization of PINN and cPINN methods, both in terms of applicability as well as domain decomposition approach, which efficiently lends itself to parallelized computation. The XPINN code is available on *https://github.com/AmeyaJagtap/XPINNs*.

**AMS subject classifications**: 35E05, 35E15, 68T99, 76J20, 74B05

*Corresponding author. *Email addresses:* ameyadjagtap@gmail.com, ameya_jagtap@brown.edu (A. D. Jagtap), george_karniadakis@brown.edu (G. E. Karniadakis)

 

**Key words**: PINN, XPINN, domain decomposition, irregular domains, machine learning, physics-informed learning.

# 1 Introduction

Recently deep neural networks (DNNs) have gained a lot of attention in the field of scientific machine learning (SciML). Thanks to their universal approximation properties, they can be exploited to construct alternative approaches for solving PDEs. In particular, they offer nonlinear approximation through the composition of hidden layers, which does not limit the approximation to the linear spaces. The training of a DNN based model (black-box surrogate model) usually requires a large amount of labeled data, which are often unavailable in many scientific applications. However, when the governing PDEs are known, their solutions can be learned in a physics-informed fashion with relatively small amounts of data. The physics-informed loss functions are constructed based on PDE residuals and the DNN is trained by minimizing this loss function, which, in turn, satisfies the governing physical laws. The notion of a neural network to solve PDEs was proposed in 90's, see [1–4], and more recently in [5–7], which had rapid success due to remarkable advances in the GPU hardware but also the stochastic gradient descent algorithms such as Adam [20]. More broadly, Owhadi [33] constructed physics-informed learning machines that made use of systematically structured prior information about the solution. Brunton, et al. [34] proposed the SINDy framework for dictionary learning of dynamical systems. Ling et al. [39] uses DNN to model the Reynolds stresses in a Reynolds-averaged Navier-Stokes model. Wang et al. [29] proposed the physics-informed machine learning approach for turbulence modeling. Tompson et al. [40] used a convolutional neural network to solve a large sparse linear system for Navier-Stokes equations. In [38], the authors proposed a deep Galerkin method, which is a deep learning algorithm for solving PDEs. Recently, Raissi et al. [37] used automatic differentiation and proposed physics-informed neural networks (PINNs), where the PDE residual is incorporated into the loss function of fully-connected neural networks as a regularizer, thereby constraining the space of admissible solutions. In this setting, the problem of inferring solutions of PDEs is transformed into an optimization problem of the loss function. The major advantage of PINNs is providing a mesh-free algorithm as the differential operators in the governing PDEs are approximated by *automatic differentiation* [8]. PINNs require a modest amount of data, which can be properly enforced in the loss function. PINNs can solve forward problems, where the solution of governing physical laws is inferred, as well as inverse problems, where unknown coefficients or even differential operators in the governing equations are identified. The PINNs has been applied extensively to solve various PDEs such as fractional PDEs [13, 14], stochastic PDEs [11, 12], with limited training data. Moreover, it has been successfully employed to solve many problems in computational and engineering science like, geostatistical modeling [36], cardiovascular systems [42–44], vortex-induced vibrations [45], high Mach number compressible

flows [23], turbulent fluid flows [26,27], quantification of surface breaking cracks [19], uncertainty quantification [15,16], elastodynamics [28] etc. On the theoretical side, recently, Shin et al. [31] established the mathematical foundation of the PINNs methodology. Also, Mishra and Molinaro presented estimates on the generalization error of PINNs for both forward and inverse problems, see [17,18].

One of the main limitations of PINNs is the large training cost, which can adversely affect their performance, especially for solving real-life applications, which require a PINN model to run in real-time. Therefore, it is crucial to accelerate the convergence of such models without sacrificing the performance. This issue was first addressed in the conservative PINN (cPINN) method for conservation laws [46] by employing the domain decomposition approach in PINN framework. Domain decomposition has been a fundamental development in standard numerical methods, e.g. finite elements, for solving the governing physical laws in the form of PDEs on parallel computers. In particular, the computational domain is partitioned into several subdomains and these subdomains interact only through their shared boundaries where some appropriate continuity conditions are imposed. The global solution is recovered by a succession of solutions of independent sub-problems associated with the entire domain. Apart from cPINN framework, other domain decomposition approaches employed in SciML are, Li et al. [24] employed a local neural network on discrete subdomains and solved PDEs using the variational principle. A similar approach was proposed by Kharazmi et al. [48], where the authors developed the variational PINN framework. The domain decomposition in PINN framework is very useful in two aspects: 1) in different domains the system may have different behavior, or some physical properties may be quite different (e.g., problems involving discontinuous features like shock waves); 2) decomposition of a large domain into subdomains may help to reduce the requirement on the structure complexity of the neural network in each subdomain so as to facilitate the training. Given the finite number of subdomains, a massively parallel computation can be performed. Such domain decomposition offers easy parallelization thereby decreasing the training cost drastically. Moreover, large-scale problems can be effectively handled by domain decomposition. The cPINN method can be extended to other types of equations (and not necessarily conservation laws) by employing properties of the solution/governing equation at hand. In this regard, we propose a generalized domain decomposition approach namely, the eXtended PINNs (*XPINNs*). Like PINNs, XPINN can be used to solve any PDE. Also, it has all the advantages of cPINN like deployment of separate neural networks in each subdomain, efficient hyper-parameter adjustment for all networks, easy parallelization capacity, large representation capacity (due to multiple networks), etc. Moreover, it has the following additional merits over cPINN.

- **Generalized space-time domain decomposition**: The XPINN formulation offers highly irregular, convex/non-convex space-time domain decomposition with $C^0$ or more regular boundaries. Such domain decomposition can be useful in many applications like multi-physics/multi-scale computations, simulation involving non-

smooth features such as cracks, shock waves, etc. Due to such decomposition, the XPINN method easily lends itself for space-time parallelization, thereby reducing training cost more effectively.

- **Extension to any differential equation(s)**: Unlike cPINN method, the XPINN based domain decomposition approach can be extended to any type of PDE(s), irrespective of its physical nature. This way, any differential equation can be solved efficiently and this makes XPINN method a genuinely generalized domain decomposition based PINN method, which can be easily parallelized.

- **Simple interface conditions**: Due to irregular domain decomposition, the interfaces form highly irregular shapes, especially in higher dimensions. In the XPINN, the interface conditions are very simple for any arbitrarily shaped interfaces, which does not need normal direction hence, the proposed approach can be easily extended to any complex geometry, even in higher dimensions. Moreover, such a simple interface condition is very useful in dynamic interface problems where the interface is moving.

Accurately solving complex systems of equations, especially in higher dimensions has become one of the biggest challenges in scientific computing. The advantages of XPINN makes it a suitable candidate for such complex simulations in higher dimensions, which in general require a large training cost.

The rest of the paper is arranged as follows: In Section 2 we describe the general form of governing parameterized differential equations. In Section 3 we present the proposed methodology and describe the XPINN framework in detail. In Section 4 we perform various computational experiments on irregular subdomains. In particular, we consider the Poisson's equation in an irregular domain, the viscous Burgers equation with space-time domain decomposition, the two-dimensional Navier's equations of elasticity for linear isotropic material and the compressible Euler equations involving discontinuous features like shock waves. We consider both forward and inverse PDE problems using the proposed methodology. Finally, we summarize our findings in Section 5.

## 2 Problem formulation

The general form of a parametrized PDE is given by

$$\begin{aligned} \mathcal{L}_{\mathbf{x}}(u;\lambda) &= f(\mathbf{x}), \quad \mathbf{x}\in\Omega\subset\mathbb{R}^d, \\ \mathcal{B}_k(u) &= g_k(\mathbf{x}), \quad \mathbf{x}\in\Gamma_k\subset\partial\Omega \end{aligned} \tag{2.1}$$

for $k=1,2,\cdots,n_b$, where $\mathcal{L}_{\mathbf{x}}(\cdot)$ is the differential operator, $u$ is the solution, $\lambda=\{\lambda_1,\lambda_2,\cdots\}$ are the model parameters, $\mathcal{B}_k(\cdot)$ can be Dirichlet, Neumann, or mixed boundary conditions and $f(\mathbf{x})$ is the forcing term. Note that for transient problems we consider time $t$ as one of the component of $\mathbf{x}$, and the initial conditions can be simply treated as a particular

type of boundary condition on the given spatio-temporal domain. The above setup encapsulates a wide range of problems in engineering and science. For equation (2.1), we define the residual $\mathcal{F}(u)$ as $\mathcal{F}(u) := \mathcal{L}_{\mathbf{x}}(u;\lambda) - f(\mathbf{x})$.

In this paper, we shall solve both forward problems, where the solution of a PDE is inferred given fixed model parameters ($\lambda$) as well as inverse problems, where the unknown model parameters are learned from the observed data. The given mathematical model is converted into a surrogate model, more specifically, a given problem of solving PDE is converted into an optimization problem, where global minima of loss function correspond to the solution of the PDE. The loss function can be defined using training data points like initial and boundary conditions and the residual of the given PDE evaluated at random locations in the space-time domain.

# 3 Methodology

In the literature, different types of DNNs with different architectures such as fully connected neural networks, recurrent neural networks, convolutional neural networks, etc are available. Among them, the most basic network is the feed-forward fully connected DNN, where the neurons of adjacent layers are fully connected. In this work, we are going to use fully connected DNNs for all the computations.

## 3.1 Mathematical setup for fully connected neural networks

Let $\mathcal{N}^L : \mathbb{R}^{D_i} \to \mathbb{R}^{D_o}$ be a feed-forward neural network of $L$ layers and $N_k$ neurons in $k^{th}$ layer ($N_0 = D_i$, and $N_L = D_o$). The weight matrix and bias vector in the $k^{th}$ layer ($1 \leq k \leq L$) are denoted by $\mathbf{W}^k \in \mathbb{R}^{N_k \times N_{k-1}}$ and $\mathbf{b}^k \in \mathbb{R}^{N_k}$, respectively. The input vector is denoted by $\mathbf{z} \in \mathbb{R}^{D_i}$ and the output vector at $k^{th}$ layer is denoted by $\mathcal{N}^k(\mathbf{z})$ and $\mathcal{N}^0(\mathbf{z}) = \mathbf{z}$. We denote the activation function by $\Phi$ which is applied layer-wise along with the scalable parameters $na^k$, where $n$ is the scaling factor. Layer-wise introduction of the additional parameters $a^k$ changes the slope of activation function in each hidden-layer, thereby increasing the training speed. Moreover, these activation slopes can also contribute to the loss function through the slope recovery term, see [21, 22] for more details. Such locally adaptive activation functions enhance the learning capacity of the network, especially during the early training period. In the recent study, Jagtap et al. [30] proposed the Kronecker Neural Networks, which is a general framework for the neural network with adaptive activation functions. In particular, they proposed rowdy activation functions, which allows faster network training than the locally adaptive activation functions. But in this paper we are employing the locally adaptive activation functions. Mathematically, one can prove this by comparing the gradient dynamics of the adaptive activation function method against that of the fixed activation method. The gradient dynamics of the adaptive activation modifies the standard dynamics (fixed activation) by multiplying a conditioning matrix by the gradient and by adding the approximate second-order term. In this paper, we used scaling factor $n = 5$ for all hidden-layers and initialize $na^k = 1$, $\forall k$, see [22] for details.

The $(L-1)$-hidden layer feed-forward neural network is defined by

$$\mathcal{N}^k(\mathbf{z}) = \mathbf{W}^k \Phi(a^{k-1}\mathcal{N}^{k-1}(\mathbf{z})) + \mathbf{b}^k \in \mathbb{R}^{N_k}, \quad 2 \leq k \leq L \tag{3.1}$$

and $\mathcal{N}^1(\mathbf{z}) = \mathbf{W}^1\mathbf{z} + \mathbf{b}^1$, where in the last layer, the activation function is identity. By letting $\tilde{\mathbf{\Theta}} = \{\mathbf{W}^k, \mathbf{b}^k, a^k\} \in \mathcal{V}$ as the collection of all weights, biases, and slopes, and taking $\mathcal{V}$ as the parameter space, we can write the output of the neural network as

$$u_{\tilde{\mathbf{\Theta}}}(\mathbf{z}) = \mathcal{N}^L(\mathbf{z}; \tilde{\mathbf{\Theta}}),$$

where $\mathcal{N}^L(\mathbf{z}; \tilde{\mathbf{\Theta}})$ emphasizes the dependence of the neural network output $\mathcal{N}^L(\mathbf{z})$ on $\tilde{\mathbf{\Theta}}$. In general, weights and biases are initialized from known probability distributions. Xavier initialization [47] is one of the most widely used initialization methods, which initializes the weights in the network by drawing them from a distribution with zero mean and a finite variance. The optimal value of variance is $1/N$, where $N$ is the number of nodes feeding into that layer.

### 3.2 A brief overview of the Physics-Informed Neural Networks

There are various ways to construct a surrogate model for the simulation of underlying physical processes governed by PDEs. The black-box surrogate model offers purely data-driven based training of DNNs, where the aim is to find the optimal parameters $\tilde{\mathbf{\Theta}}$ of the network such that the mismatch between the training data and the network prediction is locally minimized. The major drawback of such a surrogate model is that it requires a large amount of data, which is generally unavailable. PINN [37] is the surrogate model, which leverages the governing PDEs in the loss function and by minimizing the violation of governing PDEs by the solution of NN, the space of admissible solutions can be drastically reduced. The advantage of the PINN model is that accurate solutions and fast convergence can be obtained using very small amounts of data available from initial/boundary conditions or carefully performed experiments.

Let $\{\mathbf{x}_u^{(i)}\}_{i=1}^{N_u}$ and $\{\mathbf{x}_F^{(i)}\}_{i=1}^{N_F}$ be the set of randomly selected training and residual points, respectively. These points are usually drawn from a distribution, which is usually not known a priori and often need to be chosen from the given input training data. The PINN algorithm aims to learn a surrogate $u = u_{\tilde{\mathbf{\Theta}}}$ for predicting the solution $u$ of the given PDE. The PINN loss function is given as

$$\mathcal{J}(\tilde{\mathbf{\Theta}}) = W_u \, \mathrm{MSE}_u(\tilde{\mathbf{\Theta}}; \{\mathbf{x}_u^{(i)}\}_{i=1}^{N_u}) + W_{\mathcal{F}} \, \mathrm{MSE}_{\mathcal{F}}(\tilde{\mathbf{\Theta}}; \{\mathbf{x}_F^{(i)}\}_{i=1}^{N_F}), \tag{3.2}$$

where $W_u$ and $W_{\mathcal{F}}$ are the weights for the data mismatch and residual terms, respectively. The mean squared error (MSE) is given by

$$\mathrm{MSE}_u(\tilde{\mathbf{\Theta}}; \{\mathbf{x}_u^{(i)}\}_{i=1}^{N_u}) = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u^{(i)} - u_{\tilde{\mathbf{\Theta}}}(\mathbf{x}_u^{(i)}) \right|^2,$$

$$\mathrm{MSE}_{\mathcal{F}}(\tilde{\mathbf{\Theta}}; \{\mathbf{x}_F^{(i)}\}_{i=1}^{N_F}) = \frac{1}{N_F} \sum_{i=1}^{N_F} \left| \mathcal{F}_{\tilde{\mathbf{\Theta}}}(\mathbf{x}_F^{(i)}) \right|^2.$$

The term $\text{MSE}_u$ is the MSE for data mismatch term (DNN part). To make the problem well-posed, this term enforces the given initial/ boundary conditions as a constraint. The term $\text{MSE}_{\mathcal{F}}$ is the MSE for PDE residual (PDE part) where $\mathcal{F}_{\tilde{\Theta}} := \mathcal{F}(u_{\tilde{\Theta}})$ represents the residual of the governing PDEs. Here, $N_u$ and $N_F$ represent the number of training data and residual points, respectively. The parameters of the neural networks $u_{\tilde{\Theta}}$ can be estimated by minimizing the above loss function.

**Remark 3.1.** The training data points are not only obtained from the initial and boundary conditions of the given PDE, but they can also be obtained at specified locations in the domain from the experiments and define a low- or high-fidelity data as per the possible error involved in the measurements. A high resolution numerical method can also provide an ample amount of synthetic training data.

**Remark 3.2.** To construct the residual $\mathcal{F}_{\tilde{\Theta}}$ in the loss function, derivatives of the solution with respect to the independent variables are required, which can be computed using automatic differentiation (AD) [8]. AD is an accurate way to calculate derivatives in a computational graph compared to numerical differentiation since they do not suffer from errors such as truncation and round-off errors. Thus, evaluation of PDE operator $\mathcal{L}_{\mathbf{x}}(\cdot)$ acting on $u_{\tilde{\Theta}}$ is achieved easily with such graph-based differentiation, which can be incorporated in the loss function along with initial and boundary conditions. Hence, the PINN method is a grid-free method, avoiding the tyranny of mesh generation.

### 3.3 Extended Physics-Informed Neural Networks

In this section we shall discuss the XPINN methodology, which is basically a PINN method on the decomposed domains. Here we describe the basic terminology used in the follow up presentation.

- **Subdomains**: The subdomains $\Omega_q$, $q=1,2,\cdots N_{sd}$ refer to the non-overlapping subset of the whole computational domain $\Omega$ such that $\Omega = \bigcup_{q=1}^{N_{sd}} \Omega_q$ and $\Omega_i \cap \Omega_j = \partial\Omega_{ij}$, $i \neq j$. $N_{sd}$ represents the total number of subdomains. In the non-overlapping domain decomposition, the subdomains intersect only on their interface $\partial\Omega_{ij}$.

- **Interface**: The interface is the common boundary between two or more subdomains, where the corresponding Sub-Nets communicate with each other.

- **Sub-Net**: The sub-net also called as sub-PINNs refers to the individual PINN with their own set of optimized hyperparameters employed in each subdomain.

- **Interface Conditions**: These conditions are used to stitch the decomposed subdomains together in order to obtain a solution of the governing PDEs over the complete domain. Based on the nature of the governing equations, one or more interface conditions can be applied along the common interface such as solution continuity, flux continuity, etc.

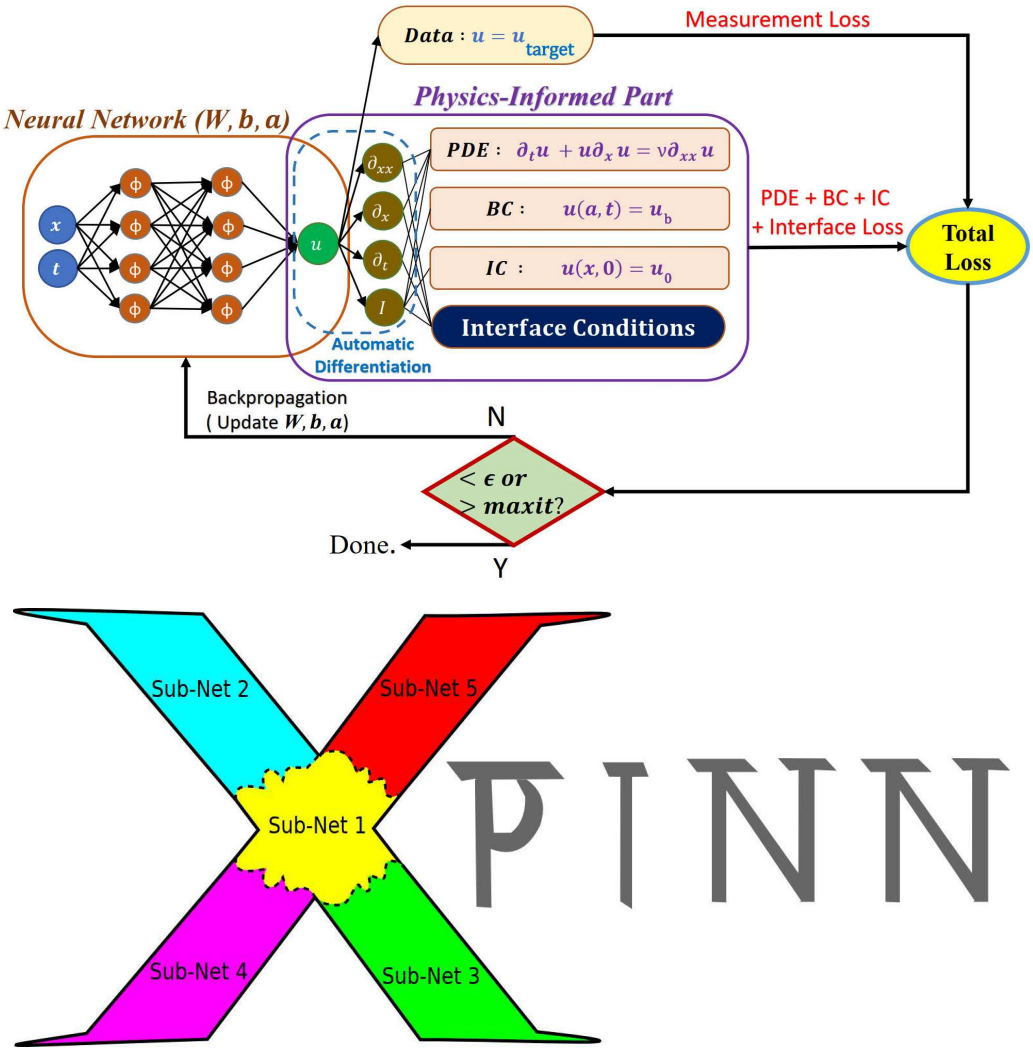


Figure 1: The top figure is the schematic of XPINN sub-net employed in a subdomain where neural network and physics-informed part for viscous Burgers equation are shown. The bottom figure shows the irregularly shaped subdomain divisions in 'X'-shaped domain, where sub-net is employed in each subdomain and they are stitched together using the interface conditions. In this case, the domain boundary is shown by black continuous line, whereas the interface is shown by black dash line.

Fig. 1 (top) shows a schematic of the XPINN Sub-Net, where along with DNN and PDE parts, additional interface conditions are also contributing to the loss function. The basic interface conditions for XPINN include the *residual continuity condition* in strong form as well as enforcing the average solution given by different Sub-Net's along the common interface. As discussed in the cPINN framework [46], for stability it is not necessary to enforce the average solution along the common interface, but the compu-

tational experiments reveal that it will drastically speed-up the convergence rate. Fig. 1 (bottom) shows the schematic representation of XPINN where the 'X'-shaped domain is divided into irregular subdomains, and Sub-Net's are employed in each subdomain with different network architecture to obtain the solution of the same underlying PDE. Such domain decomposition also offers easy parallelization of the network, which is quite important in terms of achieving computational efficiency. XPINN has all the advantages of cPINN like parallelization capacity, large representation capacity, efficient choice for hyper-parameters like optimization method, activation function, depth or width of the network etc. depending on some intuitive knowledge of the solution regularity in each subdomain, etc. In case of smooth zones, we can use a shallow network, whereas a deep neural network can be employed in a region where a complex solution is expected. Apart from this, there are various advantages of the XPINN approach over the cPINN method. Unlike cPINN, XPINN can be used to solve any type of PDEs and not necessarily conservation laws. In case of XPINN, there is no need to find the normal direction in order to apply normal flux continuity condition. This significantly reduces the complexity of the algorithm, especially in the case of large-scale problems with complex domains as well as for moving interface problems.

Consider the computational domain, which is divided into $N_{sd}$ number of non-overlapping regular/irregular subdomains. In the XPINN framework, the output of the neural network in the $q^{th}$ subdomain is given by

$$u_{\tilde{\Theta}_q}(\mathbf{z}) = \mathcal{N}^L(\mathbf{z};\tilde{\Theta}_q) \in \Omega_q, \quad q = 1,2,\cdots,N_{sd}.$$

Then, the final solution is obtained as

$$u_{\tilde{\Theta}}(\mathbf{z}) = \sum_{q=1}^{N_{sd}} u_{\tilde{\Theta}_q}(\mathbf{z}) \cdot \mathbb{1}_{\Omega_q}(\mathbf{z}), \tag{3.3}$$

where the indicator function $\mathbb{1}_{\Omega_q}(\mathbf{z})$ is defined as

$$\mathbb{1}_{\Omega_q}(\mathbf{z}) := \begin{cases} 0 & \text{if } \mathbf{z} \notin \Omega_q, \\ 1 & \text{if } \mathbf{z} \in \Omega_q \backslash \text{Common interface in the } q^{th} \text{ subdomain}, \\ \frac{1}{\mathcal{S}} & \text{if } \mathbf{z} \in \text{Common interface in the } q^{th} \text{ subdomain}, \end{cases}$$

where $\mathcal{S}$ represent the number of subdomains intersecting along the common interface.

### 3.3.1 Subdomain loss function for the forward and the inverse problems

Let $\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{u_q}}$, $\{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{F_q}}$ and $\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}}$ be the set of randomly selected training, residual, and the common interface points, respectively in the $q^{th}$ subdomain. The $N_{u_q}$,$N_{F_q}$, and $N_{I_q}$ represent the number of training data points, the number of residual points, and the number of points on the common interface in the $q^{th}$ subdomain, respectively.

Similar to PINN, the XPINN algorithm aims to learn a surrogate $u_q = u_{\tilde{\Theta}_q}$, $q = 1,2,\cdots,N_{sd}$ for predicting the solution $u = u_{\tilde{\Theta}}$ of the given PDE over the entire computational domain using Eq. (3.3). The loss function of XPINN is defined subdomain-wise, which has a similar structure as the PINN loss function in each subdomain but is endowed with the interface conditions for stitching the subdomains together. For the forward problem, the loss function in the $q^{th}$ subdomain is defined as

$$\begin{aligned}\mathcal{J}(\tilde{\Theta}_q) = {} & W_{u_q}\,\mathrm{MSE}_{u_q}(\tilde{\Theta}_q;\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{uq}}) + W_{\mathcal{F}_q}\,\mathrm{MSE}_{\mathcal{F}_q}(\tilde{\Theta}_q;\{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{Fq}}) \\ & + W_{I_q}\underbrace{\mathrm{MSE}_{u_{avg}}(\tilde{\Theta}_q;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{Iq}})}_{\text{Interface condition}} + W_{I_{\mathcal{F}_q}}\underbrace{\mathrm{MSE}_{\mathcal{R}}(\tilde{\Theta}_q;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{Iq}})}_{\text{Interface condition}} \\ & + \underbrace{\text{Additional Interface Condition's}}_{\text{Optional}}, \end{aligned} \tag{3.4}$$

where $q=1,2,\cdots,N_{sd}$. The $W_{u_q}$,$W_{\mathcal{F}_q}$,$W_{I_{\mathcal{F}_q}}$ and $W_{I_q}$ are the data mismatch, residual and interface (both, residual as well as average solution continuity along the interface) weights, respectively. The MSE is given for each term by

$$\mathrm{MSE}_{u_q}(\tilde{\Theta}_q;\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{uq}}) = \frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}\left|u^{(i)} - u_{\tilde{\Theta}_q}(\mathbf{x}_{u_q}^{(i)})\right|^2,$$

$$\mathrm{MSE}_{\mathcal{F}_q}(\tilde{\Theta}_q;\{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{Fq}}) = \frac{1}{N_{F_q}}\sum_{i=1}^{N_{Fq}}\left|\mathcal{F}_{\tilde{\Theta}_q}(\mathbf{x}_{F_q}^{(i)})\right|^2,$$

$$\mathrm{MSE}_{u_{avg}}(\tilde{\Theta}_q;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{Iq}}) = \sum_{\forall q^+}\left(\frac{1}{N_{I_q}}\sum_{i=1}^{N_{Iq}}\left|u_{\tilde{\Theta}_q}(\mathbf{x}_{I_q}^{(i)}) - \left\{\left\{u_{\tilde{\Theta}_q}(\mathbf{x}_{I_q}^{(i)})\right\}\right\}\right|^2\right),$$

$$\mathrm{MSE}_{\mathcal{R}}(\tilde{\Theta}_q;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{Iq}}) = \sum_{\forall q^+}\left(\frac{1}{N_{I_q}}\sum_{i=1}^{N_{Iq}}\left|\mathcal{F}_{\tilde{\Theta}_q}(\mathbf{x}_{I_q}^{(i)}) - \mathcal{F}_{\tilde{\Theta}_{q^+}}(\mathbf{x}_{I_q}^{(i)})\right|^2\right),$$

where the terms $\mathrm{MSE}_{u_q}$ and $\mathrm{MSE}_{\mathcal{F}_q}$ are same as before described in the PINN algorithm and $\mathcal{F}_{\tilde{\Theta}_q} := \mathcal{F}(u_{\tilde{\Theta}_q})$ represent the residual of the governing PDEs in the $q^{th}$ subdomain. The $\mathrm{MSE}_{\mathcal{R}}$ is the residual continuity condition on the common interface given by two different neural networks on subdomains $q$ and $q^+$, respectively; the superscript $+$ over $q$ represents the neighbouring subdomain(s). Both $\mathrm{MSE}_{\mathcal{R}}$ and $\mathrm{MSE}_{u_{avg}}$ terms are defined for all neighbouring subdomain(s) $q^+$, which is shown in their respective expressions by the summation sign over all $q^+$. The average value of $u$ along the common interface is given by $\{\{u_{\tilde{\Theta}_q}\}\} = u_{avg} := \frac{u_{\tilde{\Theta}_q} + u_{\tilde{\Theta}_{q^+}}}{2}$ (assuming that along the common interface only two subdomains intersect). The interface conditions ensures that the information from one subdomain can be propagated throughout the neighboring subdomains.

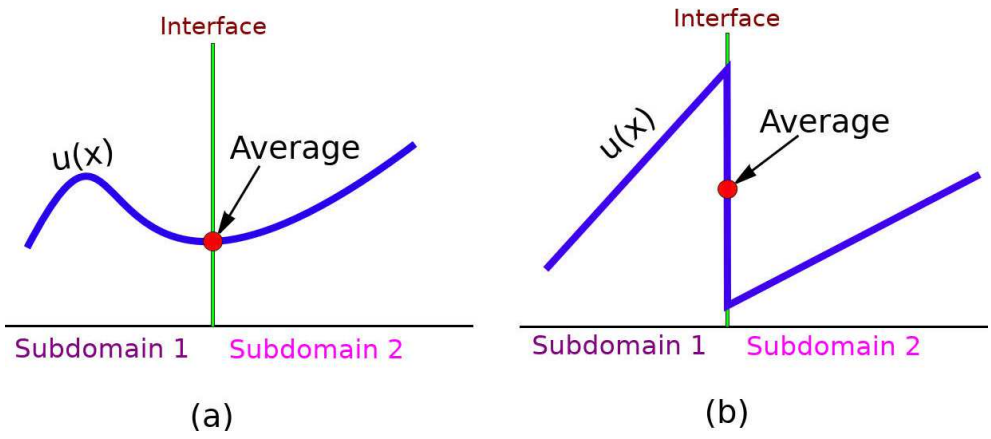


Figure 2: (a) Continuous and (b) discontinuous functions along the interface with average values.

**Remark 3.3.** In any domain decomposition method, interface conditions play an important role not only to stitch the subdomains together but also in terms of convergence. More specifically, the type of interface conditions decides the solution regularity across the interface, which can affect the convergence rate. In the proposed XPINN method, the enforcement of the average solution gives $C^0$ solution continuity across interface. Moreover, the residual continuity property can theoretically enforce the solution regularity in the classical sense, i.e., the solution across the interface is sufficiently continuous such that it satisfies its governing PDE, which is computed using AD. Therefore, XPINN can be used to solve any differential equations on decomposed domains. Apart from these conditions, *additional interface conditions* such as flux continuity, $C^k$ solution continuity ($k>0$) etc. can be imposed depending on the type of the PDEs and the interface orientation. As an example, for conservation laws, both normal spatial flux and residual continuity conditions can be imposed on spatially divided subdomains, which makes the XPINN method a generalized domain decomposition method.

**Remark 3.4.** The weights $W_{u_q}, W_{\mathcal{F}_q}, W_{I_{\mathcal{F}_q}}$ and $W_{I_q}$ play an important role in the convergence of the minimizer. As shown by Wang et al. [9], these weights can be chosen dynamically, leading to faster convergence compared to static weights. However, such dynamic weights put additional computational burden, which can be significant in case of multiple loss function based methods like cPINN and XPINN.

**Remark 3.5.** Fig. 2 shows the enforcement of the average solution $u_{avg}$ along the interface for both smooth and discontinuous solutions. In case of a smooth solution, the enforcement of the average solution is similar to the solution continuity condition (Fig. 2(a)). However, in case of discontinuous solution, which we can expect in case of the hyperbolic conservation laws, such enforcement imposes the adjacent networks to satisfy the average value of the discontinuous solution along the interface (Fig. 2(b)).

**Remark 3.6.** A sufficient number of interface points ($N_{I_q}$) must be taken while stitching the subdomains together. This is important for faster convergence of the algorithm, especially for the internal subdomains, which does not have any training data points. We can also use the knowledge of solutions along the interface lines obtained from each sub-network to properly select the locations of interface points.

Next, we shall consider the inverse problems, in particular, the problem of data-driven discovery of PDEs from the known solution data. Inverse problems are some of the most important problems in science, and, in general, they are ill-posed in nature. In the inverse problem, where model parameters ($\lambda$) are identified from the known solution, the loss function in the $q^{th}$ subdomain is given as

$$\begin{aligned}\mathcal{J}(\tilde{\mathbf{\Theta}}_q,\lambda) = {} & W_{u_q}\,\mathrm{MSE}_{u_q}(\tilde{\mathbf{\Theta}}_q,\lambda;\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{u_q}}) + W_{\mathcal{F}_q}\,\mathrm{MSE}_{\mathcal{F}_q}(\tilde{\mathbf{\Theta}}_q,\lambda;\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{u_q}}) \\ & + W_{I_q}\underbrace{\{\mathrm{MSE}_{u_{avg}}(\tilde{\mathbf{\Theta}}_q,\lambda;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}}) + \mathrm{MSE}_{\lambda}(\tilde{\mathbf{\Theta}}_q,\lambda;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}})\}}_{\text{Interface condition's}} \\ & + W_{I_{\mathcal{F}_q}}\underbrace{\mathrm{MSE}_{\mathcal{R}}(\tilde{\mathbf{\Theta}}_q,\lambda;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}})}_{\text{Interface condition}} + \underbrace{\text{Additional Interface Condition's}}_{\text{Optional}}, \end{aligned} \tag{3.5}$$

where $q=1,2,\cdots,N_{sd}$, and

$$\mathrm{MSE}_{\mathcal{F}_q}(\tilde{\mathbf{\Theta}}_q,\lambda;\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{u_q}}) = \frac{1}{N_{u_q}}\sum_{i=1}^{N_{u_q}}\left|\mathcal{F}_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^{(i)})\right|^2,$$

$$\mathrm{MSE}_{\lambda}(\tilde{\mathbf{\Theta}}_q,\lambda;\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}}) = \sum_{\forall q^+}\left(\frac{1}{N_{I_q}}\sum_{i=1}^{N_{I_q}}\left|\lambda_q(\mathbf{x}_{I_q}^{(i)}) - \lambda_{q^+}(\mathbf{x}_{I_q}^{(i)})\right|^2\right).$$

Other MSE terms are same as in case of the forward problem. The $\mathrm{MSE}_\lambda$ term gives the continuity condition for the value of constant parameters on the interface.

### 3.3.2 Optimization method

We seek to find $\tilde{\mathbf{\Theta}}_q^*$ that minimizes the loss function $\mathcal{J}(\tilde{\mathbf{\Theta}}_q)$ in each subdomain. Even though there is no theoretical guarantee that the above mentioned procedure converges to a global minimum, our experiments indicate that as long as the given PDE is well-posed and has a unique solution, the XPINN formulation is capable of achieving an accurate solution provided that a sufficiently expressive network and a sufficient number of residual points are used. There are several optimization algorithms available to minimize the loss function. In general, a gradient-based optimization method is employed for the training of parameters. In the basic form, given an initial value of parameters $\tilde{\mathbf{\Theta}}_q$ in the $q^{th}$ subdomain, the parameters are updated as

$$\tilde{\mathbf{\Theta}}_q^{m+1} = \tilde{\mathbf{\Theta}}_q^m - \eta_l \left.\frac{\partial \mathcal{J}(\tilde{\mathbf{\Theta}}_q)}{\partial \tilde{\mathbf{\Theta}}_q}\right|_{\tilde{\mathbf{\Theta}}_q=\tilde{\mathbf{\Theta}}_q^m}, \quad q=1,2,\cdots,N_{sd},$$

where $\eta_l$ is the learning rate.

The stochastic gradient descent (SGD) method is the widely used optimization method. In SGD, a small set of points are randomly sampled to find the direction of the gradient in every iteration. The SGD algorithm works well to avoid bad local minima during training of DNN under one point convexity property. A brief survey on SGD is given by Ruder [41]. In particular, we shall use the Adam optimizer which is one version of SGD [20].

**Remark 3.7.** Note that, due to highly non-convex nature of the XPINN loss function, it is very challenging to locate its global minimum. However, for several local minima, values of the loss function are comparable and the accuracy of the corresponding predicted solutions is similar. These are the instances of the so-called *operator mimicking phenomenon*, where the strong non-convexity of the loss function may lead to equally good multiple local minima.

**Remark 3.8.** In terms of coding, we can employ the same XPINN code to solve both forward as well as inverse problems, since for an inverse problem code we only need to add the additional model parameters involved in the governing PDEs to the list of the parameters to be optimized without modifying the other parts of forward problem code. The XPINN codes are written in Python, and the deep learning framework Tensorflow [35] is used to take advantage of its inbuilt automatic differentiation capability.

Algorithm 1 summarizes the XPINN methodology in detail. The XPINN code is available on *https://github.com/AmeyaJagtap/XPINNs*.

### 3.4 A brief note on the errors involved in the XPINN methodology

In this section we shall discuss different sources of error associated with the XPINN formulation in solving the differential equations. Since the XPINN formulation is similar to cPINN (except the interface conditions), we expect it to have a similar sources of errors.

Let $F_q$ and $u_q^{ex}$ be the family of functions that can be represented by the chosen neural network and the exact solution of PDE, respectively in each subdomain $q=1,2,\cdots,N_{sd}$. Then, we define $u_{a_q}=\arg\min_{f\in F_q}||f-u_q^{ex}||$ as the best approximation to the exact solution $u_q^{ex}$. Let $u_{\tau_q}=\arg\min_{\tilde{\Theta}_q}\mathcal{J}(\tilde{\Theta}_q)$ be the solution obtained by training the $q^{th}$ Sub-Net and $u_{g_q}=\arg\min_{\tilde{\Theta}_q}\mathcal{J}(\tilde{\Theta}_q)$ be the solution of the $q^{th}$ Sub-Net at global minimum. Therefore, the total error in each subdomain $q=1,2,\cdots,N_{sd}$ with its own Sub-Net consists of the following errors

$$\begin{aligned}
\mathcal{E}_{\text{app }\ q} &= ||u_{a_q}-u_q^{ex}||, \quad \text{Approximation Error,}\\
\mathcal{E}_{\text{gen }\ q} &= ||u_{g_q}-u_{a_q}||, \quad \text{Generalization Error,}\\
\mathcal{E}_{\text{opt }\ q} &= ||u_{\tau_q}-u_{g_q}||, \quad \text{Optimization Error.}
\end{aligned}$$

**Algorithm 1:** XPINN algorithm

**Step 0**: Divide the computational domain into $N_{sd}$ number of non-overlapping regular/irregular subdomains.

**Step 1**: Specify the training set over all subdomains $\Omega_q$

*Training data*: $u_{\tilde{\mathbf{\Theta}}_q}$ network $\{\mathbf{x}_{u_q}^{(i)}\}_{i=1}^{N_{uq}}$, $q=1,2,\cdots,N_{sd}$.

*Residual training points*: $\mathcal{F}_{\tilde{\mathbf{\Theta}}_q}$ network $\{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{Fq}}$, $q=1,2,\cdots,N_{sd}$.

*Interface points*: $\{\mathbf{x}_{I_q}^{(i)}\}_{i=1}^{N_{I_q}}$, $q=1,2,\cdots,N_{sd}$.

**Step 2**: For each subdomain, identify the common interface points with the neighbouring subdomains.

**Step 3**: Construct the neural network $u_{\tilde{\mathbf{\Theta}}_q}$ with random initialization of parameters $\tilde{\mathbf{\Theta}}_q$ in each subdomain.

**Step 4**: Construct the residual neural network $\mathcal{F}_{\tilde{\mathbf{\Theta}}_q}$ in each subdomain by substituting surrogate $u_{\tilde{\mathbf{\Theta}}_q}$ into the governing equations using automatic differentiation and other arithmetic operations.

**Step 5**: Specify the loss function $\mathcal{J}(\tilde{\mathbf{\Theta}}_q)$ in the $q^{th}$ subdomain.

**Step 6**: Find the best parameters using suitable optimization method for minimizing the loss function in each subdomain

$$\tilde{\mathbf{\Theta}}_q^* = \arg\min_{\tilde{\mathbf{\Theta}}_q \in \mathcal{V}_q} \mathcal{J}(\tilde{\mathbf{\Theta}}_q), \quad q=1,2,\cdots,N_{sd}, \tag{3.6}$$

where $\mathcal{V}_q$ is the parameter space in $q^{th}$ subdomain.

Fig. 3 shows a schematic view of the subdomain-wise errors involved in the XPINN framework. These errors are interconnected through the interface conditions as shown by the double-headed arrows.

A deep neural network may reduce the approximation error by increasing the network expressivity, however, it may yield a large generalization error, which is known as the *bias-variance* trade-off. In PINNs, two important factors in the generalization error are the number and distribution of residual points ($N_{F_q}$ and $\{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{Fq}}$) as these factors can adversely alter the landscape of the loss function. The optimization error is introduced due to the complexity of the loss function. The structure of the network, i.e, depth, width etc., strongly affect the optimization error. Other hyperparameters of the network such as the learning rate or the number of iterations can be tuned to further control and improve this error. Thus, we define the total error in XPINN as

$$\mathcal{E}_{XPINN} := ||u_\tau - u^{ex}|| \leq ||u_\tau - u_g|| + ||u_g - u_a|| + ||u_a - u^{ex}||, \tag{3.7}$$

where $(u^{ex}, u_\tau, u_g, u_a)(\mathbf{z}) = \sum_{q=1}^{N_{sd}} (u_q^{ex}, u_{\tau_q}, u_{g_q}, u_{a_q})(\mathbf{z}) \cdot \mathbb{1}_{\Omega_q}(\mathbf{z})$.

Similar to cPINN method, the domain decomposition approach of the XPINN frame-

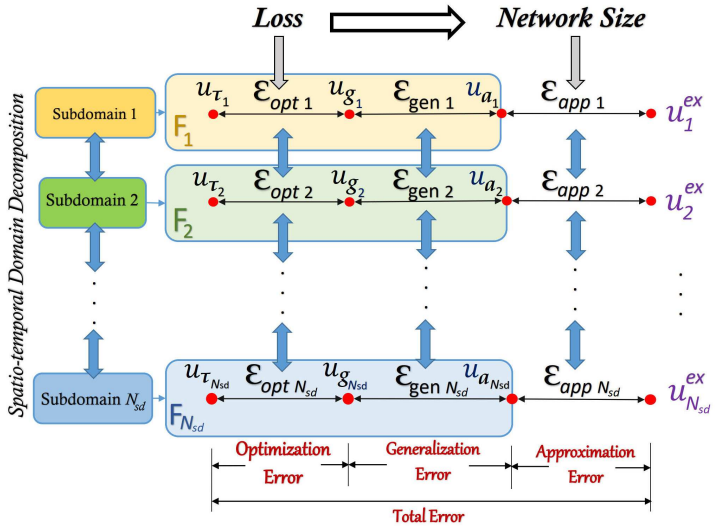


Figure 3: Illustrating the errors involved in the XPINN framework. The total error in each subdomain consists of the optimization error, the generalization error and the approximation error. Here, $u_q^{ex}$ represents the exact solution of the governing equation in the $q^{th}$ subdomain, $u_{\tau_q}$ is the solution obtained by training the NN in the $q^{th}$ subdomain, $u_{g_q}$ is the NN solution at the global minimum in the $q^{th}$ subdomain, and $u_{a_q}$ is the best function close to $u_q^{ex}$ in the corresponding function space $F_q$, $q=1,2,\cdots,N_{sd}$.

work has various advantages over the PINN algorithm: XPINN can reduce the generalization error by selecting $N_{F_q}$ and $\{\mathbf{x}_{F_q}^{(i)}\}_{i=1}^{N_{F_q}}$ carefully. Moreover, the freedom given by XPINN methodology in terms of arbitrary domain decomposition can be used to divide the domain according to the required number and the location of residual points. This can be achieved with some prior (or sparse) knowledge about the solution behavior such that the localized powerful neural network can be employed in that subdomain. XPINN can also reduce the approximation error by selecting the network size, and other hyperparameters such as depth/width of the network, activation function, optimization algorithm, learning rate etc. in each subdomain depending on the nature and complexity of the solution. The optimization error in XPINN can be reduced by using the different network size for each subdomain.

### 3.5 Comparison of PINN, cPINN and XPINN frameworks

This section gives a comparison of PINN, cPINN and XPINN methodology in terms of various parameters like domain decomposition, applicability, interface conditions etc. Table 1 shows this comparison. In all aspects, the XPINN framework is superior to the PINN and cPINN frameworks. XPINN combines all the advantages of the PINN and the cPINN methods, and thus it can be very efficient in solving scientific problems using the machine learning framework.

Table 1: Comparison of PINN, cPINN and XPINN frameworks. 'DEs' represent Differential Equations.

| | PINN | cPINN | XPINN |
|---|---|---|---|
| Spatial domain Decomposition | ✗ | ✓ | ✓ |
| Parallelization capacity | ✗ | ✓ | ✓ |
| Localized representation capacity | ✗ | ✓ | ✓ |
| Efficient hyperparameter adjustment | ✗ | ✓ | ✓ |
| Applicability | Any DEs | Conservation laws | Any DEs |
| Interface conditions | - | Complex | Simple |
| Spatio-temporal domain decomposition | ✗ | ✗ | ✓ |

The XPINN framework offers many advantages over the PINN and cPINN frameworks but, it also share one limitation with its predecessors. The absolute error in the PDE solution does not go below levels of about $10^{-5}$ due to the inaccuracy involved in solving a high-dimensional non-convex optimization problem that may result in bad minima.

# 4 Computational results and discussion

In this section we shall solve the partial differential equations using the proposed XPINN methodology, where the computational domain is arbitrarily decomposed. Various examples such as the two- and three-dimensional Poisson's equation, the viscous Burgers equation, the two-dimensional Navier's equations of elasticity, and the two-dimensional compressible Euler equations with shock waves are solved in highly irregular subdomains. Moreover, both forward and inverse problems are solved using the proposed method. In order to reduce the optimization error, all examples are solved using double precision arithmetic.

## 4.1 A pedagogical example: Comparison of PINNs and XPINNs for linear advection equation

This example provides a comparison between PINN and XPINN results for a simple linear advection equation. The one-dimensional linear advection equation is given by

$$u_t + cu_x = 0, \quad x \in \Omega \subset \mathbb{R}, \quad t > 0$$

with initial condition

$$u(x,0) = \begin{cases} 1 & \text{if } -0.2 \leq x \leq 0.2, \\ 0 & \text{otherwise,} \end{cases}$$

which is a square pulse. The exact solution can be easily obtained from the initial condition since, the initial profile is advected from left to right with wave speed $c = 0.5$.

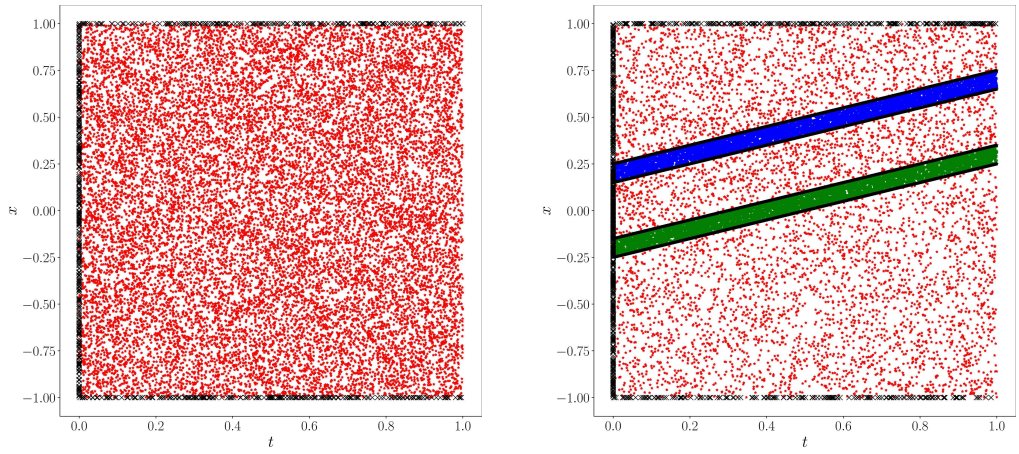


Figure 4: Linear Advection Equation: Training data and residual points for PINN (left) and XPINN (right) methods. In case of XPINN, the subdomains 1, 2 and 3 are shown by red, blue and green points, respectively.

Fig. 4 shows the training data and residual points for PINN (left) and XPINN (right) over the $[-1,\ 1]\times[0,\ 1]$ spatio-temporal domain. For PINN, we used 13 hidden-layers with 20 neuron in each layer and the number of residual points are 14000, whereas the XPINN architecture is given in Table 2. In both cases the total number of residual points is the same, the total number of boundary/initial data points is the same, the learning rate is 8e-4, and the activation function is hyperbolic tangent. In both PINN and XPINN, the network representation capacity in terms of the number of tunable parameters (weights and biases) is approximately the same, which gives a fair comparison on the predictive accuracy of the solution. In both cases, the network training is performed up to 15k iterations. Fig. 5 shows the results for PINN and XPINN at three different times 0.3, 0.6 and 0.9 given by first, second and third columns, respectively. The first row gives the solution over the entire spatial domain whereas the second and third rows give the zoomed-view of the solutions. In all cases, the inference accuracy of XPINN (with relative $L_2$ error 6.0245e-2) is better than that of PINN (with relative $L_2$ error 9.6589e-2). The main reason behind this is the employment of a powerful localized neural network in the XPINN subdomain as opposed to global neural network in PINN.

Table 2: Linear Advection Equation: XPINN architecture in each subdomain.

| Subdomain number | 1 | 2 | 3 |
|---|---|---|---|
| # Layers | 2 | 6 | 6 |
| # Neurons | 20 | 20 | 20 |
| # Residual points | 6000 | 4000 | 4000 |

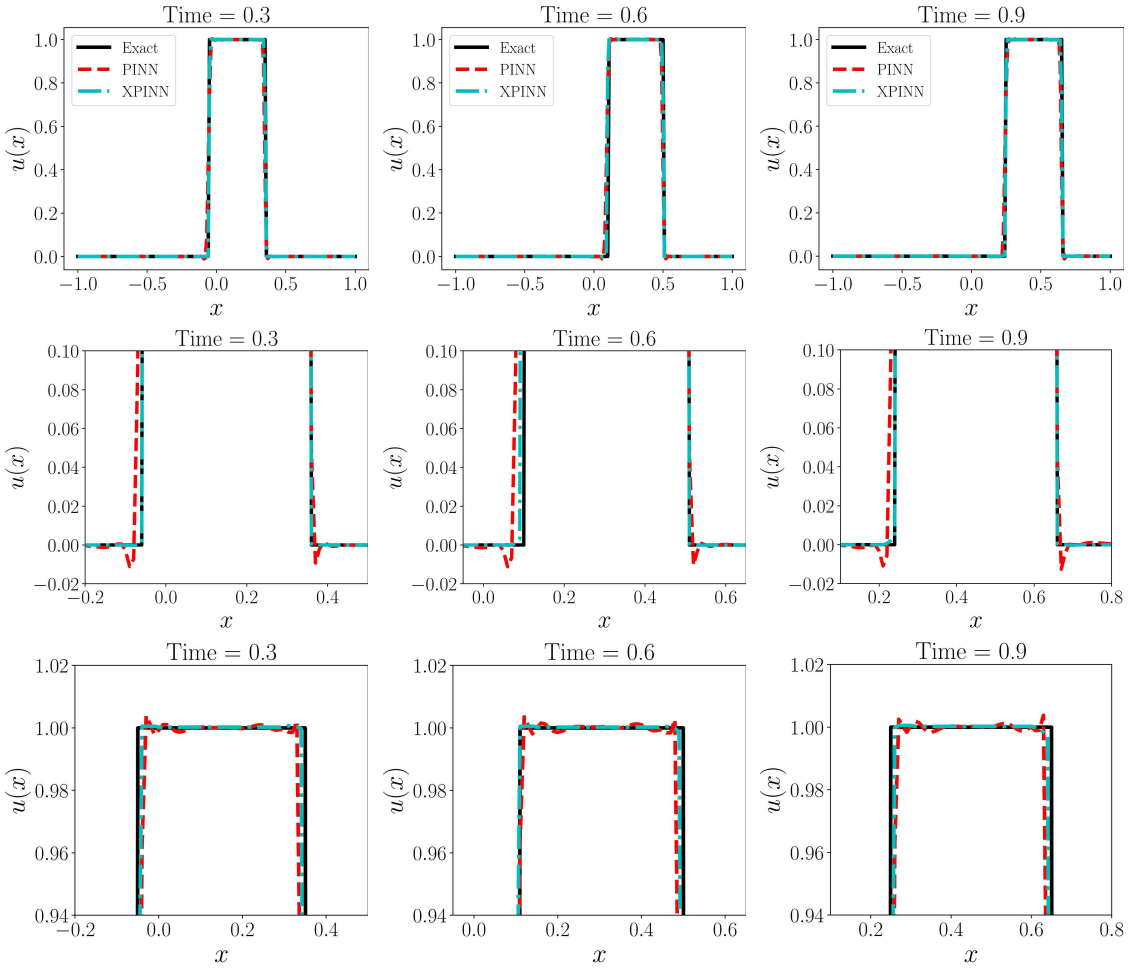


Figure 5: Linear Advection Equation: Results for PINN and XPINN at three different time 0.3, 0.6 and 0.9 given by the first, second and the third columns, respectively. The first row gives the solution over the entire spatial domain whereas second and third rows give the zoomed-view of the solutions.

## 4.2 Poisson's equation in a two-dimensional arbitrary domain

Consider a two-dimensional Poisson's equation given by

$$\Delta u = f(x,y), \quad x,y \in \Omega \subset \mathbb{R}^2.$$

In particular, we used the exact solution $u(x,y)=\exp(x)+\exp(y)$ from which we can infer the forcing term $f(x,y)$. The expression for the boundary is given in polar coordinates as $r=1.5+0.14\ \sin(4\theta)+0.12\ \cos(6\theta)+0.09\ \cos(5\theta)$, where $\theta \in [0,2\pi)$, and the boundary points are obtained as $x=0.02+r\cos(\theta)$ and $y=r\sin(\theta)$. The computational domain is further divided into three highly irregular, non-convex subdomains as shown in Fig. 6, where the two internal subdomain boundaries are given by

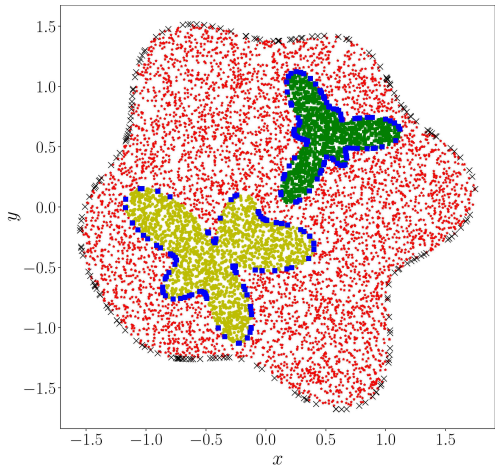


Figure 6: Three irregular subdomains for two-dimensional Poisson's equation. Red stars, yellow circles and green squares correspond to subdomain 1, 2 and 3, respectively. The training data point along the boundary are shown by black cross and the interface points are shown by the blue squares.

$$r_1 = 0.5 + 0.18\ \sin(3\theta) + 0.08\ \cos(2\theta) + 0.2\ \cos(5\theta);$$
$$r_2 = 0.34 + 0.04\ \sin(5\theta) + 0.18\ \cos(3\theta) + 0.1\ \cos(6\theta);$$

and the corresponding interface points are obtained as $(x_1,y_1) = (-0.4 + r_1\cos(\theta), -0.4 + r_1\sin(\theta))$ and $(x_2,y_2) = (0.5 + r_2\cos(\theta), 0.6 + r_2\sin(\theta))$, respectively. For the Poisson's equation, the residual term is defined as $\mathcal{F}(u) := \Delta u - f(x,y)$. Table 3 gives the details of network architecture used in the three subdomains. The activation function is hyperbolic tangent and the learning rate is 0.0006. The number of interface points is 100 on both interfaces and their locations are shown in Fig. 6. The values of $W_{u_q} = 20, W_{\mathcal{F}_q} = 1, W_{I_{\mathcal{F}_q}} = 1$ and $W_{I_q} = 20$ and 100 interface points are used on both interfaces. The number of boundary training data points is 200. Both interface and boundary training data points are chosen randomly as shown in Fig. 6. Fig. 7 (top row) shows the exact solution, predicted solution and the absolute point-wise error in the whole domain, and the white line shows the position of arbitrary internal subdomains. XPINN can stitch these arbitrarily shaped subdomains together very well, which can be seen from the point-wise error plot. By increasing the number of interface points, the interface imprinting effect can be reduced significantly. The bottom figure shows the loss function variation in each subdomain up to 25k iterations.

Table 3: Two-dimensional Poisson's equation: Neural network architecture in each subdomain.

| Subdomain number | 1 | 2 | 3 |
|---|---|---|---|
| # Layers | 2 | 4 | 3 |
| # Neurons | 30 | 20 | 25 |
| # Residual points | 7000 | 1800 | 1200 |

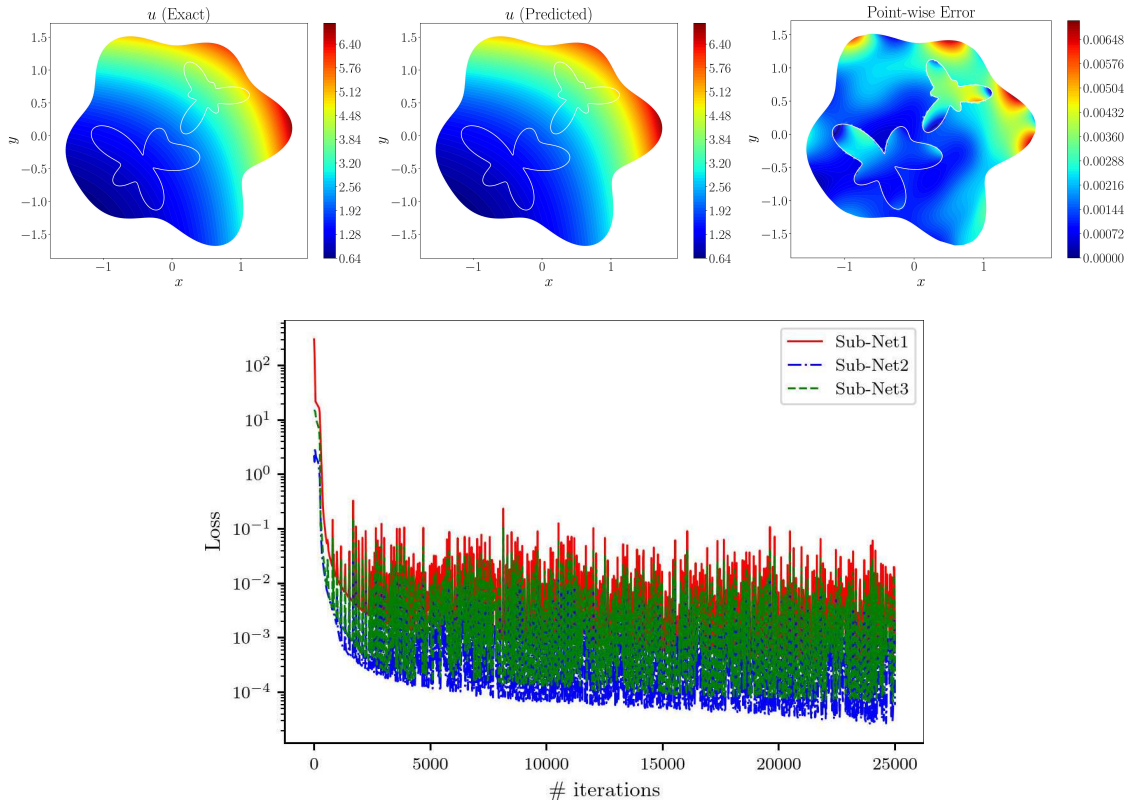


Figure 7: Two-dimensional Poisson's equation: Exact, predicted solutions and the point-wise error (top row, left to right). The white line shows the position of internal subdomains. The variation of loss functions with the number of iterations for three subdomains is shown in the bottom figure.

### 4.2.1 Effect of depth and width of the network

The network hyperparameters such as depth, width, and activation function play a crucial role in the convergence of the loss function. Due to high expressivity of a deep neural network, it can be shown that with increase in the depth, the network can approximate very complex solutions. To analyze the performance of the proposed XPINN method, we shall discuss the effect of depth and width of the network on the predictive accuracy of the solution of Poisson's problem. In this study, we keep all other hyperparameters fixed and their values are same as before. We have chosen different values for width and depth, which is fixed in all subdomains, and for each combination we perform the five different runs up to 25k iterations corresponding to different initialization of weights and biases. Table 4 shows the average relative $L_2$ error in the XPINN solution over the whole domain with varying width and depth of the neural network. We observe that by increasing the depth simultaneously with the width of the network (which in turn increases the expressivity of the network) the predictive accuracy of the solution increases.

Table 4: Two-dimensional Poisson's equation: Relative $L_2$ error in the solution with width and depth of the fully connected feed forward neural networks. These errors represent the average of 5 different runs up to 25k iterations corresponding to different initialization of weights and biases. In all cases the number of residual points $N_{F_1}=8000$, $N_{F_2}=1200$ and $N_{F_3}=800$ are fixed and the number of fixed training data points $N_{u_1}$ in subdomain 1 is 300.

| Depth \ Width | 4 | 8 | 16 | 32 |
|---|---|---|---|---|
| 1 | 1.13243e-02 | 5.84621e-03 | 1.73130e-03 | 1.77051e-03 |
| 2 | 1.96692e-03 | 4.23742e-03 | 2.90450e-03 | 1.63421e-04 |
| 4 | 2.38137e-03 | 2.95869e-03 | 3.23784e-04 | 9.37170e-04 |
| 8 | 7.05701e-03 | 6.45847e-03 | 3.39927e-03 | 5.85394e-04 |

### 4.2.2 Effect of number of interface points

The number and location of interface points are important for high predictive accuracy and fast convergence of the XPINN method. In this section, we investigate the effect of number of interface points on the accuracy of the solution. For this reason, we have chosen different sets of interface points for two internal subdomains (subdomain 2 and 3) and the subdomain-wise relative $L_2$ errors are plotted with the number of interface points for the internal subdomains. Note that neither of the internal subdomains have any training data points and they completely depend on outer subdomain via interface points for the convergence of their respective loss functions.

Fig. 8 shows the variation of relative $L_2$ error for internal subdomains 2 (left) and 3 (right) up to 25k iterations. We choose 10, 100 and 1000 interface points randomly along the boundaries of the two subdomains. We observe that with an increase in the number of interface points, the error decreases faster in both subdomains. Fig. 9 shows the absolute point-wise error in the whole domain for 10, 100 and 1000 interface points. Again, it can be seen that by increasing the number of interface points, the error diminishes faster.

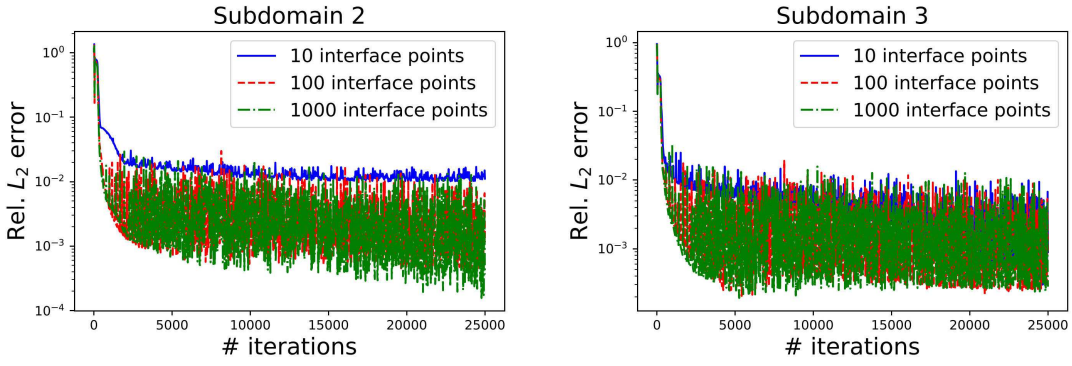


Figure 8: Two-dimensional Poisson's equation: Variation of relative $L_2$ error with number of interface points for internal subdomains 2 (left) and 3 (right).

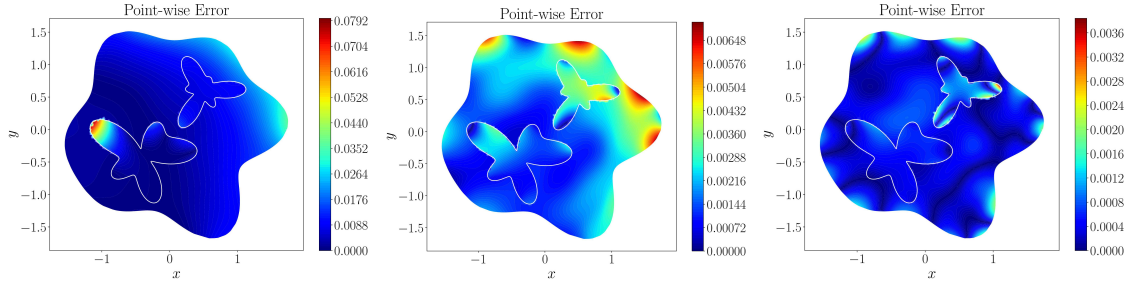


Figure 9: Two-dimensional Poisson's equation: Point-wise error in the whole domain for 10, 100 and 1000 interface points (left to right).

Thus, a sufficiently large number of interface points reduces the interface imprinting effect along the edge of the interface significantly.

### 4.3 Three-dimensional Poisson's equation

Next, we solve the three-dimensional Poisson's equation in the domain shown in Fig. 10 (bottom row). The three-dimensional Poisson's equation is given by

$$\Delta u = f(x,y,z), \quad x,y,z \in \Omega \subset \mathbb{R}^3.$$

We used the exact solution $u(x,y,z) = \exp(x)+\exp(y)+\exp(z)$ from which we can infer the forcing term $f(x,y,z)$ as well as boundary conditions. The 3D domain is divided into three subdomains as shown in Fig. 10 (top row). The outer boundaries of three subdomains are given by the following expressions

$$\begin{aligned} r_1 &= (-z+0.5)+0.12\ \sin(6\theta), \\ r_2 &= \frac{z^2}{6}+0.2, \\ r_3 &= 0.22+0.04\ \sin(6\theta)+0.05\ \sin(5\theta), \end{aligned}$$

where $1 \leq z \leq 1.4$ and $\theta \in [0,\ 2\pi)$. Table 5 gives the details of the network architecture in each subdomain. The learning rate is 0.0006, the values of $W_{u_q} = 40$, $W_{\mathcal{F}_q} = 1$, $W_{I_{\mathcal{F}_q}} = 1$ and $W_{I_q} = 40$, and the number of interface points on interface 1 and 2 are 1450 and 1080, respectively. The number of boundary training data points is 2200. The XPINN code is run upto 100k iterations. Fig. 11 shows the exact solution (top row), predicted solution (middle row) and absolute point-wise error (bottom row) at $z = 1.1$, 1.2 and 1.3 locations (first, second and third columns). These results indicate that XPINN can easily handle such high dimensional problems involving complex domains. Moreover, the interface conditions are easy to implement even in higher dimensions on very complex boundary surfaces.

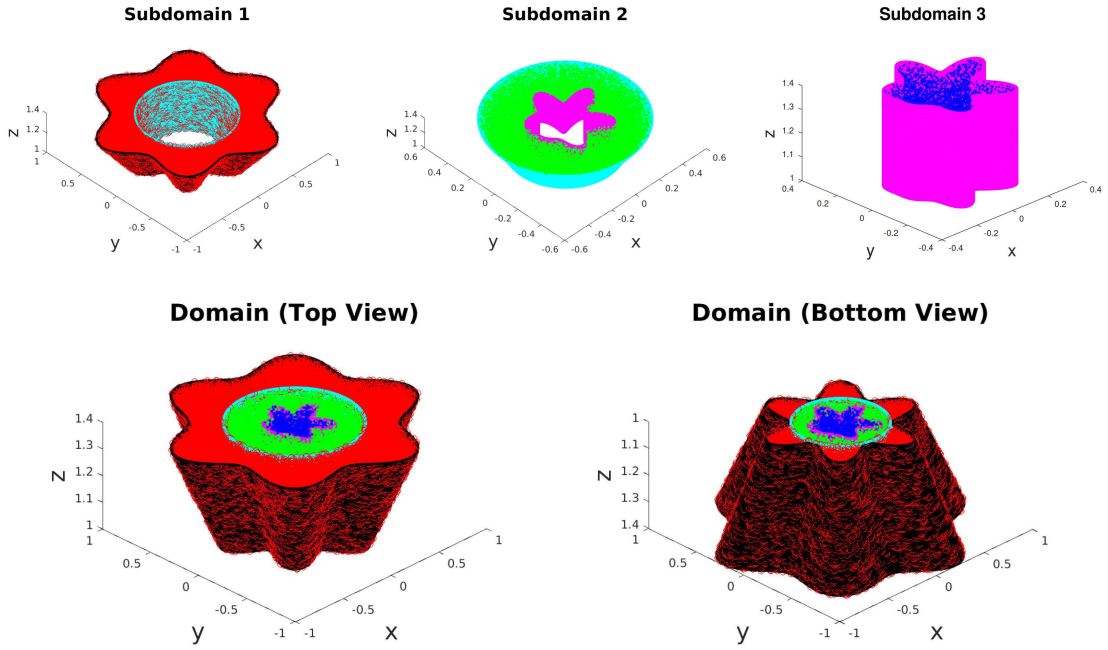


Figure 10: Three-dimensional domain for Poisson's equation: Subdomains 1,2 and 3 are shown in the top row (left to right). The residual points are shown in red, green and blue colors in subdomains 1, 2 and 3, respectively. The boundary points are shown in subdomain 1 by black dots, the interface points are shown in subdomain 2 and 3 by cyan and magenta dots, respectively. The top and bottom views of complete domain are shown in the bottom row. The top and bottom planes given by $z=1$, 1.4 are also a part of the domain boundary.

Table 5: Three-dimensional Poisson's equation: Neural network architecture in each subdomain.

| Subdomain number | 1 | 2 | 3 |
|---|---|---|---|
| # Layers | 3 | 5 | 4 |
| # Neurons | 40 | 25 | 30 |
| # Residual points | 18000 | 8000 | 6000 |
| Adaptive Activation function | tanh | sin | tanh |

## 4.4 Viscous Burgers equation

The one-dimensional viscous Burgers equation is given by

$$u_t + uu_x = \nu u_{xx}, \quad x \in \Omega \subset \mathbb{R}, \quad t > 0$$

with initial condition $u(x,0) = -\sin(\pi x)$, boundary conditions $u(-1,t) = u(1,t) = 0$ and $\nu = 0.01/\pi$. The analytical solution can be obtained using the Hopf-Cole transformation, see Basdevant et al., [10] for more details. The nonlinearity in the convection term develops a very steep solution due to the small value of diffusion coefficient $\nu$.

The computational space-time domain is divided into two subdomains as shown in

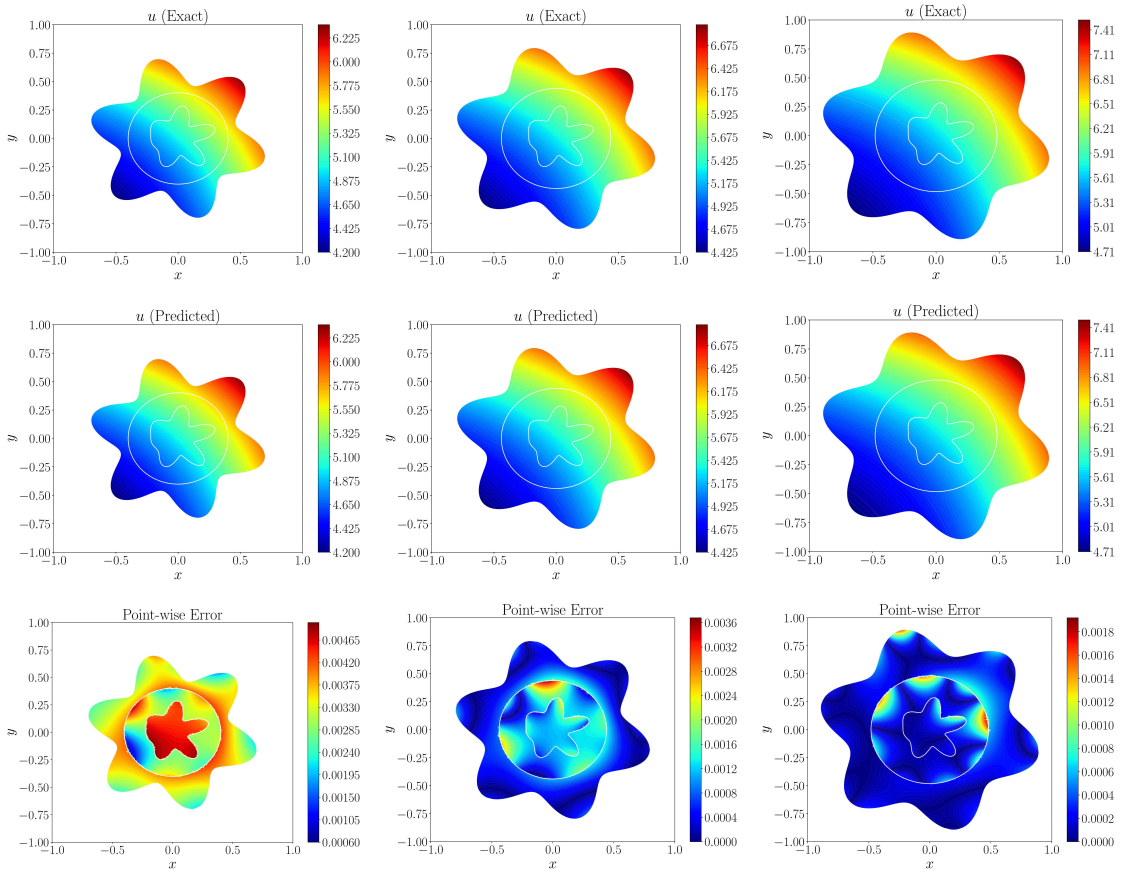


Figure 11: Slices of three-dimensional domain for Poisson's equation: Exact solution (top row), predicted solution (middle row) and point-wise error (bottom row) at $z=1.1$, 1.2 and 1.3 locations shown by first, second and third columns, respectively.

Table 6: One-dimensional viscous Burgers equation: Neural network architecture in each subdomain.

| Subdomain number | 1 | 2 |
|---|---|---|
| # Layers | 6 | 7 |
| # Neurons | 20 | 25 |
| # Residual points | 7000 | 3000 |
| Adaptive Activation function | tanh | sin |

Fig. 12, where the internal subdomain boundary is in the shape of a dolphin. Table 6 shows the network architecture in both subdomains. The learning rate is 8e-4, the values of data mismatch, residual and the interface weights are $W_{u_q}=20$, $W_{\mathcal{F}_q}=1$, $W_{I_{\mathcal{F}_q}}=1$ and $W_{I_q}=20$, respectively, and the number of interface points is 310. The number of boundary and initial training data points is 300.

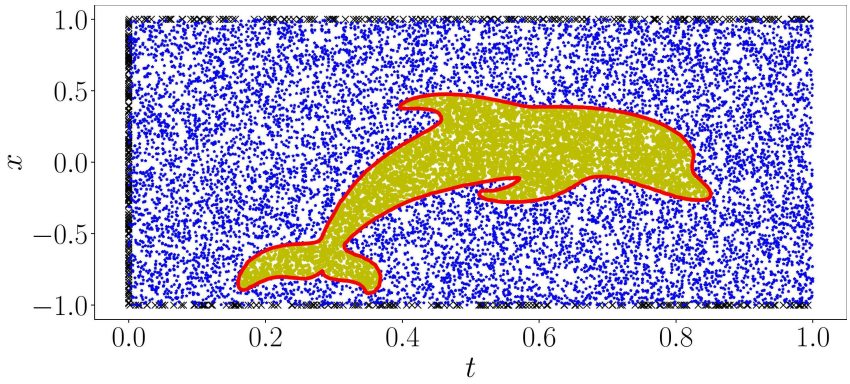


Figure 12: One-dimensional viscous Burgers equation: Residual points in subdomain 1 (blue stars) and subdomain 2 (yellow circles). Red line represent the dolphin shaped interface, which separates the two subdomains, and black cross represents the training data points from initial and boundary conditions.

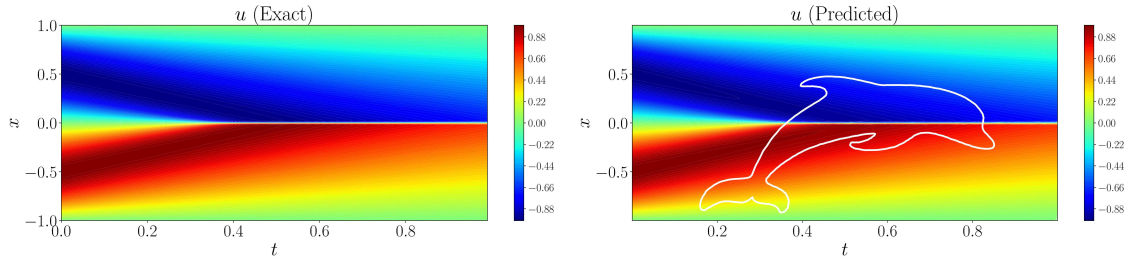


Figure 13: One-dimensional viscous Burgers equation: Contour plot of the exact (left) and predicted (right) solutions over the space-time domain. The white line in the predicted solution represents the dolphin shaped space-time interface.

Fig. 12 shows the location of 300 training data points from initial and boundary conditions. The exact and predicted solutions are shown in the Fig. 13. After 100k iterations, the relative $L_2$ error in the solution is 8.93265e-3. Fig. 14 shows the convergence history of two subdomains and the interface loss functions, which is given by the following expressions

$$\begin{aligned}
\mathcal{J}(\tilde{\Theta}_1) &= W_{u_1}\,\mathrm{MSE}_{u_1} + W_{\mathcal{F}_1}\,\mathrm{MSE}_{\mathcal{F}_1}, && \text{Subdomain 1,}\\
\mathcal{J}(\tilde{\Theta}_2) &= W_{\mathcal{F}_2}\,\mathrm{MSE}_{\mathcal{F}_2}, && \text{Subdomain 2,}\\
\mathcal{J}(\tilde{\Theta}_{\text{Interface}}) &= W_{I_{\mathcal{F}_1}}\,\mathrm{MSE}_{\mathcal{R}} + W_{I_1}\mathrm{MSE}_{u_{avg}}, && \text{Interface.}
\end{aligned}$$

From the figure we see that both the subdomain and interface losses converge together due to residual continuity condition. We can also observe that initially the subdomain 2 loss is very small due to unavailability of the training data points from the actual computational boundary, thus, it completely depends on the subdomain 1 for the convergence. As the subdomain 1 loss started converging, the subdomain 2 follows the same path, i.e., we see a sudden increase of the loss value and then the convergence. The interface loss is

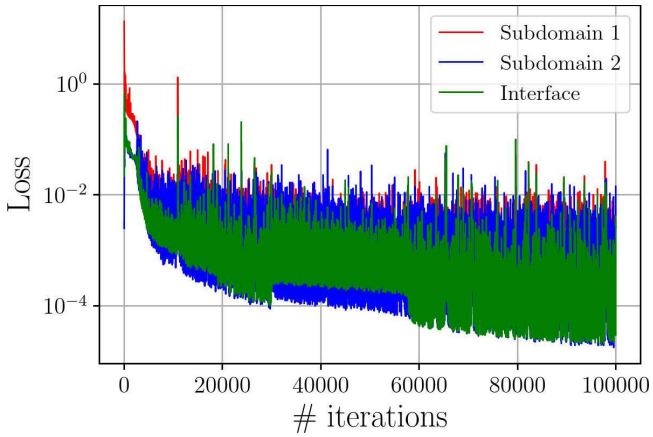


Figure 14: One-dimensional viscous Burgers equation: Loss value for two subdomains as well as for the interface.

converging faster than subdomains loss and is still decreasing even after 100k iterations. The relative $L_2$ error in the predicted solution along the interface is 5.92672e-3.

### 4.4.1 Effect of dynamic interface weights

As discussed in the methodology section, employment of dynamic weights for interface/training data terms strongly affects the performance of the algorithm, especially in the high-dimensional, multi-subdomains problems due to high computational cost associated with the evaluation of these weights. Interface weights ($W_{I_{\mathcal{F}_q}}$ and $W_{I_q}$) play a vital role in the convergence of the neural network, especially when the network does not have any training data points as in the case of internal subdomains. In the XPINN algorithm, these interface weights are fixed and in general, their values are $W_{I_{\mathcal{F}_q}}=1$ and $W_{I_q}>1$. One can use large values for $W_{I_{\mathcal{F}_q}}$, but the reason for assigning the unity value for the $W_{I_{\mathcal{F}_q}}$ is due to the residual continuity condition. It is also observed in the computational experiments that the higher values of $W_{I_{\mathcal{F}_q}}$ ($>1$) can create a destabilizing effect in the neural network training. Thus, it would be interesting to see the effect of dynamic interface weights $W^{DW}_{I_{\mathcal{F}_q}}$ and how they affect the performance of the network. For the comparison, we define the subdomain-wise loss function as

$$\begin{aligned}
&\mathcal{J}(\tilde{\Theta}_1) = W_{u_1}\,\mathrm{MSE}_{u_1} + W_{\mathcal{F}_1}\,\mathrm{MSE}_{\mathcal{F}_1} + W^{DW}_{I_{\mathcal{F}_1}}\,\mathrm{MSE}_{\mathcal{R}} + W_{I_1}\mathrm{MSE}_{u_{avg}}, && \text{Subdomain 1,}\\
&\mathcal{J}(\tilde{\Theta}_2) = W_{\mathcal{F}_2}\,\mathrm{MSE}_{\mathcal{F}_2} + W^{DW}_{I_{\mathcal{F}_2}}\,\mathrm{MSE}_{\mathcal{R}} + W_{I_2}\mathrm{MSE}_{u_{avg}}, && \text{Subdomain 2.}
\end{aligned}$$

The initial values of $W^{DW}_{I_{\mathcal{F}_q}}$ are unity in both the subdomains. Fig. 15 (top row) shows the convergence history of the loss functions without (left) and with (right) dynamic weights $W^{DW}_{I_{\mathcal{F}_q}}$ till 60k iterations. The bottom row shows the variation of dynamic weights with number of iterations. We observe that both the dynamic weights converge to an average value of 3.5 in just 5000 iterations, which is close to the unity value. Thus, we cannot

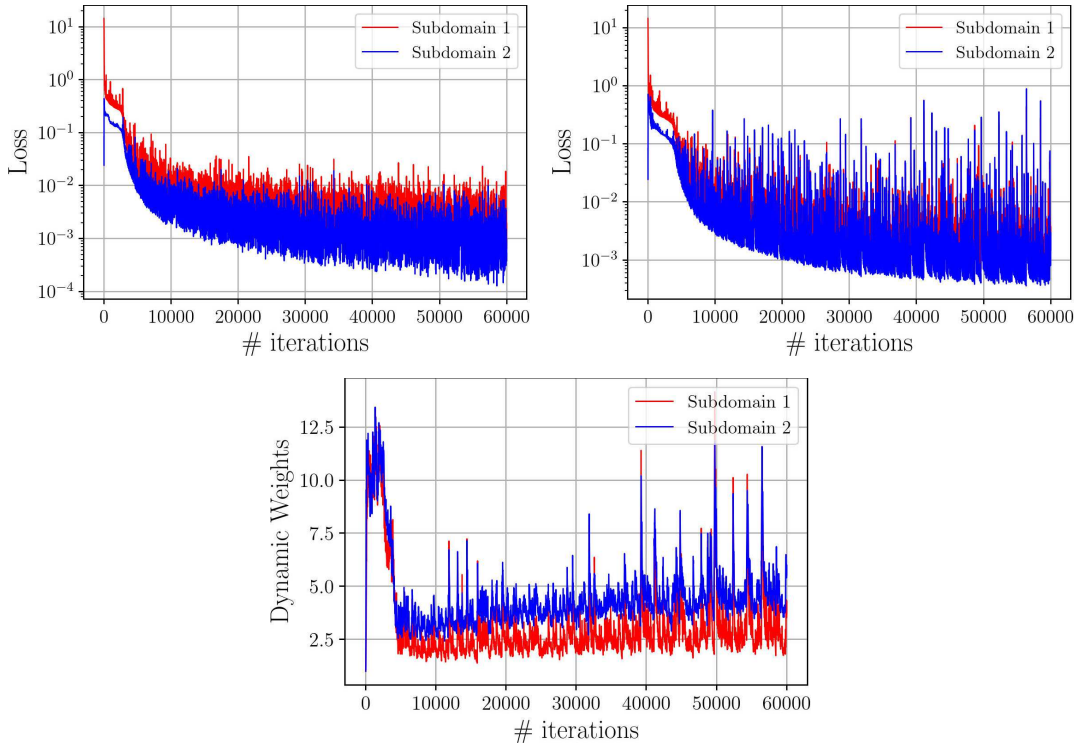


Figure 15: Top row shows the convergence history of the loss functions without (left) and with (right) dynamic weights $W^{DW}_{I_{\mathcal{F}_q}}$, whereas bottom row shows the variation in the values of $W^{DW}_{I_{\mathcal{F}_q}}$ in two subdomains with number of iterations.

see much improvement in the convergence using the dynamics weights. Another observation is that both weights ($W^{DW}_{I_{\mathcal{F}_1}}$ and $W^{DW}_{I_{\mathcal{F}_2}}$) evolve almost similarly and converge to almost the same value. The reason behind this is that both dynamic weights are associated with the same interface condition, evaluated using the same residual term but in different subdomains. This gives an idea of *shared dynamic weights*, where the same dynamic weight is used for similar terms in all subdomains with overlapping interfaces. As an example, same dynamic weights can be used for the $\text{MSE}_{\mathcal{R}}$ terms in two subdomains with overlapping interface, i.e., $W^{DW}_{I_{\mathcal{F}_1}} = W^{DW}_{I_{\mathcal{F}_2}}$. This will drastically reduce the computational cost associated with evaluation of these weights. However, it is important to note that the shared dynamic weights will not work for multi-physics problems, where different residual terms are employed. In such cases, a separate dynamic weight must be employed.

### 4.5 Two-dimensional elasticity equations: Cylinder coating on a shaft

In this test case we shall solve the more complicated two-dimensional plane strain problem of a shaft with a cylinder coating. The computational domain $\Omega$ is the concentric

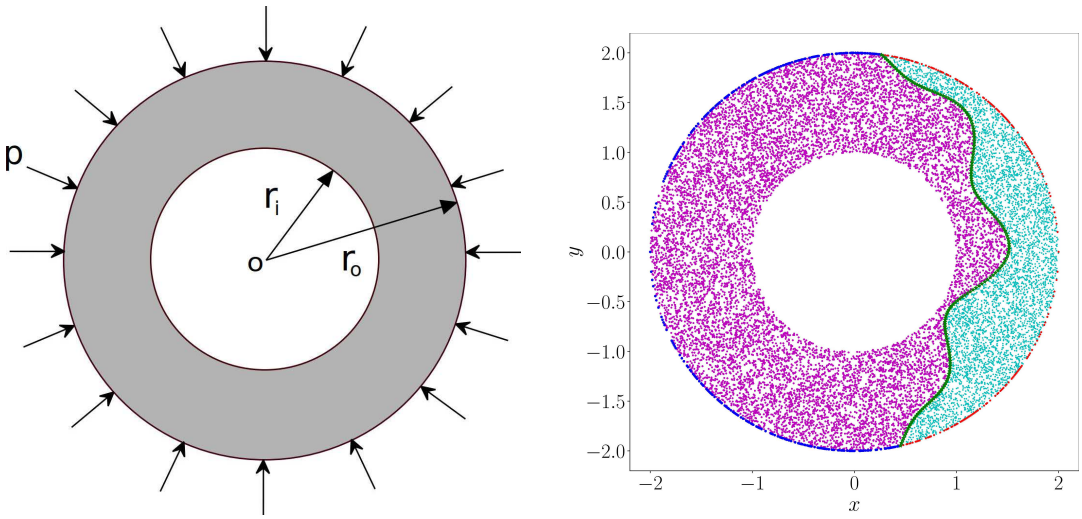


Figure 16: Two-dimensional Navier's equations: The left figure is a schematic of the shaft with a cylindrical coating while the right figure shows the residual points (magenta and cyan colors) in two subdomains. The interface points are shown in green asterisk (*), while the boundary training data points are shown in blue and red colors for subdomain 1 and 2, respectively.

annular domain where the inner and the outer radii are $r_i = 2$ and $r_o = 4$, respectively, see Fig. 16 (left). Consider a coated system on a shaft, which is loaded by a uniform pressure ($p = 1$) and the shaft is considered to be rigid compared to the coating. The governing equations are the Navier's equations of elasticity and in the absence of body force they are given as (see, [32]):

$$\left(2G\frac{1-\nu}{1-2\nu}\right)u_{xx}+Gu_{yy}+\left(\frac{G}{1-2\nu}\right)v_{xy}=0, \quad (x,y)\in\Omega\subset\mathbb{R}^2,$$
$$Gv_{xx}+\left(2G\frac{1-\nu}{1-2\nu}\right)v_{yy}+\left(\frac{G}{1-2\nu}\right)u_{xy}=0, \quad (x,y)\in\Omega\subset\mathbb{R}^2,$$

where $\mathbf{u}=\{u,v\}$ are the displacements in $x$ and $y$ directions, $\nu$ is the Poisson's ratio and $G$ is the shear modulus. The strain $\boldsymbol{\epsilon}$ is related to the displacement gradients by the kinematic relations as

$$\boldsymbol{\epsilon}=\frac{1}{2}\left(\nabla\mathbf{u}+(\nabla\mathbf{u})^T\right),$$

while the stresses $\sigma$ are related to the strains through the constitutive law also known as Hookes law, i.e.,

$$\boldsymbol{\sigma}=2G\left(\boldsymbol{\epsilon}+\frac{\nu}{1-2\nu}\text{tr}(\boldsymbol{\epsilon})\right).$$

The analytical solution in terms of displacements is given as follows

$$u=u_r\cos(\theta), \tag{4.1}$$
$$v=u_r\sin(\theta), \tag{4.2}$$

Table 7: Two-dimensional Navier's equation of elasticity: Neural network architecture in the two subdomains.

| Subdomain number | 1 | 2 |
|---|---|---|
| # Layers | 4 | 3 |
| # Neurons | 30 | 40 |
| # Residual points | 4000 | 9000 |
| Adaptive Activation function | tanh | sin |

Table 8: Two-dimensional Navier's equation of elasticity: Relative $L_2$ errors in the two subdomains.

| | Subdomain 1 | Subdomain 2 |
|---|---|---|
| $u$ | 7.4538e-3 | 9.3429e-3 |
| $v$ | 5.9482e-3 | 6.1044e-3 |

where

$$u_r = \frac{1-\nu^2}{G}\left(-\left(1+\frac{\nu}{1-\nu}\right)\frac{A}{r}+Br\left(1-\frac{\nu}{1-\nu}\right)\right),$$

with

$$A=\frac{-pr_i^2r_o^2(1-2\nu)}{r_i^2+r_o^2(1-2\nu)}, \quad B=\frac{-pr_i^2}{r_i^2+r_o^2(1-2\nu)}.$$

The value of Poisson's ratio is $\nu=0.2$ and shear modulus $G=3.3333e5$.

Fig. 16 (right) shows the irregular domain division into two subdomains. The interface is a part of the curve given by $r=2.45+0.14\ \sin(4\theta)+0.12\ \cos(12\theta)+0.09\ \cos(8\theta)$, and the interface points are obtained as $(x,y)=(-1.15+r\cos(\theta),r\sin(\theta))$. The training data points on subdomains 1 and 2 are shown by red and blue colors points whereas the interface points are shown by green asterisk. The training data points are also obtained along the edge of the shaft using the rigidity condition. Table 7 shows the neural network architecture used in the two subdomains. The learning rate is 8e-4, the values of weights $W_{u_q}=25$, $W_{\mathcal{F}_q}=1$, $W_{I_{\mathcal{F}_q}}=1$ and $W_{I_q}=25$, and the number of interface points on the interface is 650. The number of boundary training data points in subdomains 1 and 2 are 200 and 300, respectively. Due to very small values of deflections, we scaled-up the exact solution by decreasing the shear modulus value by a factor of 1e5 to obtain relatively large values of the boundary training data points. This, in turn, gives the large values of deflections, which are eventually scaled-down by the same scaling factor in order to obtain the actual values of deflections. Fig. 17 shows the exact (top row) and predicted (bottom row) solutions. The relative $L_2$ error in the two subdomains is given in Table 8. Fig. 18 gives the loss variation with the number of iterations up to 480k.

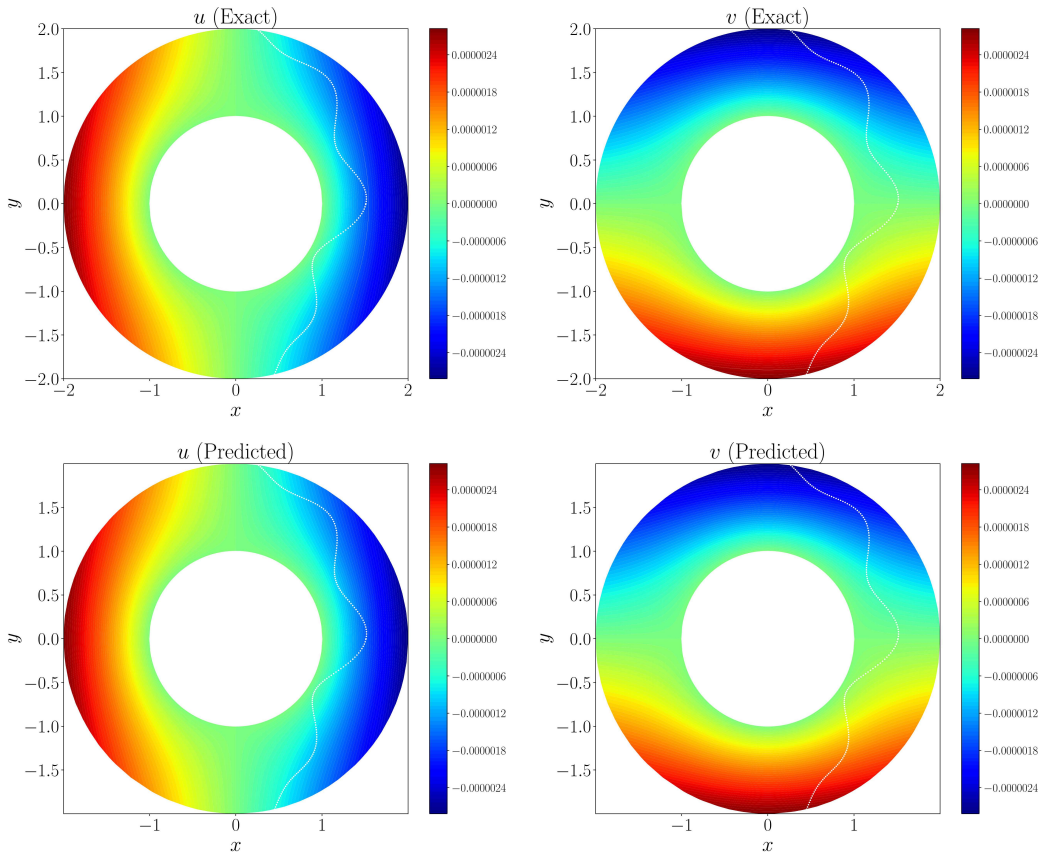


Figure 17: Two-dimensional Navier's equations: Exact solutions (top row), predicted solutions (bottom row). The white line show the interface between the two subdomains.

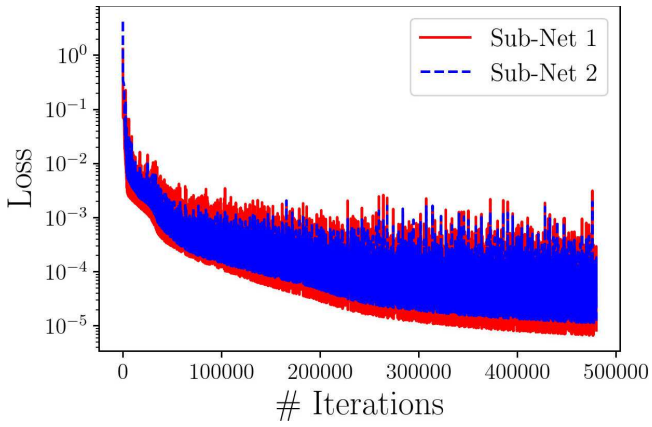


Figure 18: Two-dimensional Navier's equations: Variation of loss with number of iterations.

### 4.6 Compressible Euler equations: Oblique shock wave reflection

The compressible Euler equations can admit discontinuous solutions like shocks and contact waves. In this section, we solve the two-dimensional Euler equations for oblique shock wave reflection problem [49]. The governing compressible Euler equations in two dimensions are given as

$$U_t+G_{1_x}+G_{2_y}=0, \quad (\mathbf{x},t)\in\Omega\times(0,T]\subset\mathbb{R}^2\times\mathbb{R}_+ \tag{4.3}$$

with appropriate initial and boundary conditions. The fluxes $G_i(U)$, $i=1,2$ are functions of the conserved variable $U$. For the two-dimensional Euler equations, $U$ and $G_i$'s are given as

$$U=\begin{pmatrix}\rho\\ \rho u\\ \rho v\\ \rho E\end{pmatrix}, \quad G_1=\begin{pmatrix}\rho u\\ p+\rho u^2\\ \rho uv\\ pu+\rho uE\end{pmatrix}, \quad G_2=\begin{pmatrix}\rho v\\ \rho uv\\ p+\rho v^2\\ pv+\rho vE\end{pmatrix},$$

where $\rho,u,v,E,p$ are density, velocity components in the $x$ and $y$ directions, total energy and pressure, respectively, and $\delta_{ij}$ is Kronecker delta. The total energy is given by

$$E=\frac{p}{\rho(\gamma-1)}+\frac{1}{2}||\mathbf{u}||^2_{L_2}.$$

The first equation is a mass conservation equation, while the second and third are the momentum conservation equations in $x$ and $y$ directions, respectively, and the last equation is the energy conservation equation. We consider the oblique shock reflection test case where the computational domain is $[0,\,3]\times[0,\,1]$, see Fig. 19. Dirichlet boundary conditions are applied at the left, top and bottom boundaries, whereas all primitive variables are extrapolated at the right outgoing boundary. The pre- and post-shock conditions are

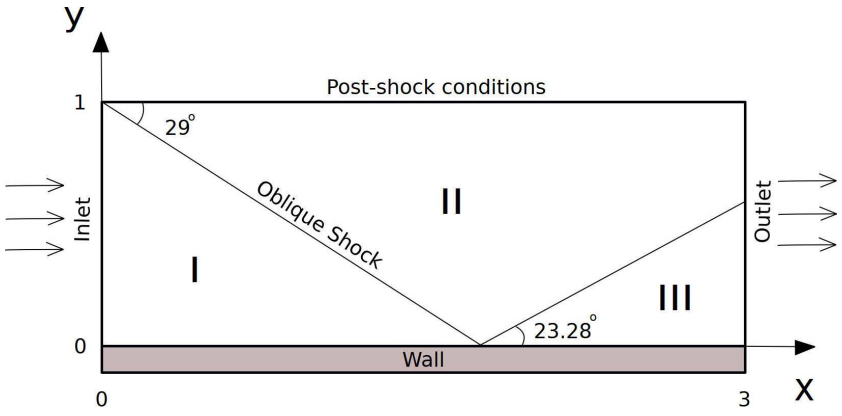


Figure 19: Schematic representation of the entire domain along with incident and reflected shock waves.

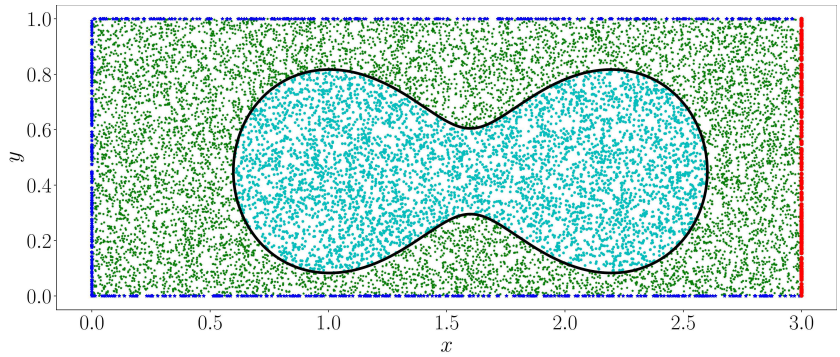


Figure 20: Compressible Euler equations: The inner peanut-shaped subdomain 2 (cyan colored circles) and outer subdomain 1 (green colored stars). The peanut-shaped domain is defined in polar coordinates as $r = 0.7\sqrt{\cos(2\theta)+\sqrt{1.1-\sin(2\theta)^2}}$, $\theta \in [0,\ 2\pi)$.

given as

$$(\rho,\ u,\ v,p) = \begin{cases} (1.0,\ 2.9,\ 0.0,\ 1/1.4) & \text{pre-shock (left boundary),} \\ (1.69997,\ 2.61934,\ -0.50833,\ 1.52819) & \text{post-shock (top boundary).} \end{cases} \tag{4.4}$$

The exact solution is available for the *steady state* case, which is given as

$$(\rho,\ u,\ v,p) = \begin{cases} (1.0,\ 2.9,\ 0.0,\ 1/1.4) & \text{Region I,} \\ (1.69997,\ 2.61934,\ -0.50833,\ 1.52819) & \text{Region II,} \\ (2.68728,\ 2.40140,\ 0.0,\ 2.93407) & \text{Region III.} \end{cases} \tag{4.5}$$

The bottom boundary is the reflecting wall where slip boundary conditions are applied. The computational domain is divided into two subdomains as shown in Fig. 20. The inner peanut-shaped subdomain 2 (cyan colored circles) and outer subdomain 1 (green colored stars) along with the boundary training data points are shown in the figure.

The $q^{th}$ subdomain loss function is given by

$$\begin{aligned} &\mathcal{J}(\tilde{\mathbf{\Theta}}_q) \\ &= W_{\mathcal{F}_q}\ (\mathrm{MSE}_G^{Mass} + \mathrm{MSE}_G^{MomX} + \mathrm{MSE}_G^{MomY} + \mathrm{MSE}_G^{Ene}) \\ &\quad + W_{u_q}\ (\mathrm{MSE}_\rho^D + \mathrm{MSE}_p^D + \mathrm{MSE}_u^D + \mathrm{MSE}_v^D + MSE_{\nabla\rho} + \mathrm{MSE}_\rho^N + \mathrm{MSE}_p^N + \mathrm{MSE}_u^N + \mathrm{MSE}_v^N) \\ &\quad + W_{I_q}\ \mathrm{MSE}_{u_{avg}} + W_{I_{\mathcal{F}_q}}\ \mathrm{MSE}_{\mathcal{R}} + W_{I_{\mathrm{Flux}}}\ \mathrm{MSE}_{\mathrm{Flux}}, \end{aligned} \tag{4.6}$$

where the MSE for mass, momentum ($x$ and $y$ directions), and energy conservation laws

are given as

$$\text{MSE}_G^{Mass}=\frac{1}{N_{F_q}}\sum_{i=1}^{N_{Fq}}|(G_1^{Mass}(\mathbf{x}_{F_q}^i))_x+(G_2^{Mass}(\mathbf{x}_{F_q}^i))_y|^2,$$

$$\text{MSE}_G^{MomX}=\frac{1}{N_{F_q}}\sum_{i=1}^{N_{Fq}}|(G_1^{MomX}(\mathbf{x}_{F_q}^i))_x+(G_2^{MomX}(\mathbf{x}_{F_q}^i))_y|^2,$$

$$\text{MSE}_G^{MomY}=\frac{1}{N_{F_q}}\sum_{i=1}^{N_{Fq}}|(G_1^{MomY}(\mathbf{x}_{F_q}^i))_x+(G_2^{MomY}(\mathbf{x}_{F_q}^i))_y|^2,$$

$$\text{MSE}_G^{Ene}=\frac{1}{N_{F_q}}\sum_{i=1}^{N_{Fq}}|(G_1^{Ene}(\mathbf{x}_{F_q}^i))_x+(G_2^{Ene}(\mathbf{x}_{F_q}^i))_y|^2,$$

whereas the MSE for Dirichlet and Neumann boundary conditions are given as

$$\text{MSE}_\rho^D=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|\rho^i-\rho_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2;\quad \text{MSE}_\rho^N=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|\nabla\rho_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2,$$

$$\text{MSE}_p^D=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|p^i-p_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2;\quad \text{MSE}_p^N=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|\nabla p_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2,$$

$$\text{MSE}_u^D=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|u^i-u_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2;\quad \text{MSE}_u^N=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|\nabla u_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2,$$

$$\text{MSE}_v^D=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|v^i-v_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2;\quad \text{MSE}_v^N=\frac{1}{N_{u_q}}\sum_{i=1}^{N_{uq}}|\nabla v_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{u_q}^i)|^2.$$

The term $\text{MSE}_{\nabla\rho}=\frac{1}{N_{\nabla_q}}\sum_{i=1}^{N_{\nabla q}}|\nabla\rho^i-\nabla\rho_{\tilde{\mathbf{\Theta}}_q}(\mathbf{x}_{\nabla_q}^i)|^2$ represents the data mismatch term for density gradient obtained from the Schlieren photography experimental technique traditionally used in high-speed aerodynamics. Here, the density gradient is known in the smooth regions using exact solution, especially near discontinuities, whose values are enforced in the loss function using $N_{\nabla_q}=1000$ points in the vicinity of shock waves, see [23] for the implementation details. The interface conditions include continuity of the mass, momentum and the energy equations as well as enforcing the average values of the primitive variables like density, velocities in $x$ and $y$ directions, and pressure along the interfaces. The learning rate is 0.0008. The $\text{MSE}_{\text{Flux}}$ term imposes the normal flux continuity conditions. The weights for the data mismatch, residual and interface terms are taken as $W_{u_q}=30$, $W_{\mathcal{F}_q}=1$, $W_{I_{\mathcal{F}_q}}=1$, $W_{I_q}=30$ and $W_{I_{\text{Flux}}}=30$, respectively; and the number of interface points on interface is 600. The number of boundary training data is 500. Table 9 gives the architecture of the neural network in two subdomains. Fig. 21 shows the contour plots of the primitive variables and Fig. 22 shows the comparison of exact and predicted solution at the $y=0.5$ location for density, $x$,$y$ direction velocities

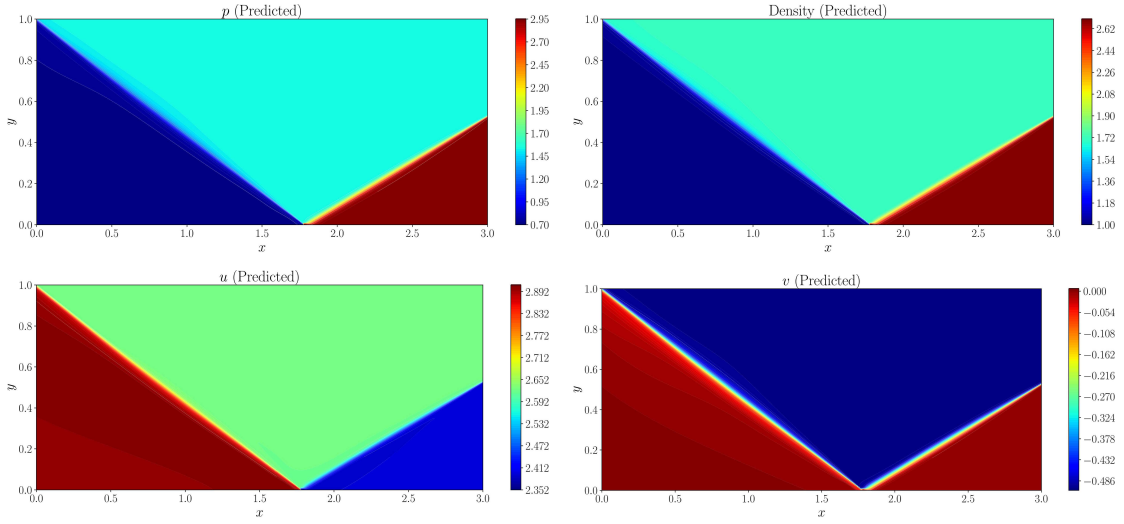


Figure 21: Compressible Euler equations: Contour plots of pressure, density and velocities in $x$ and $y$ directions.

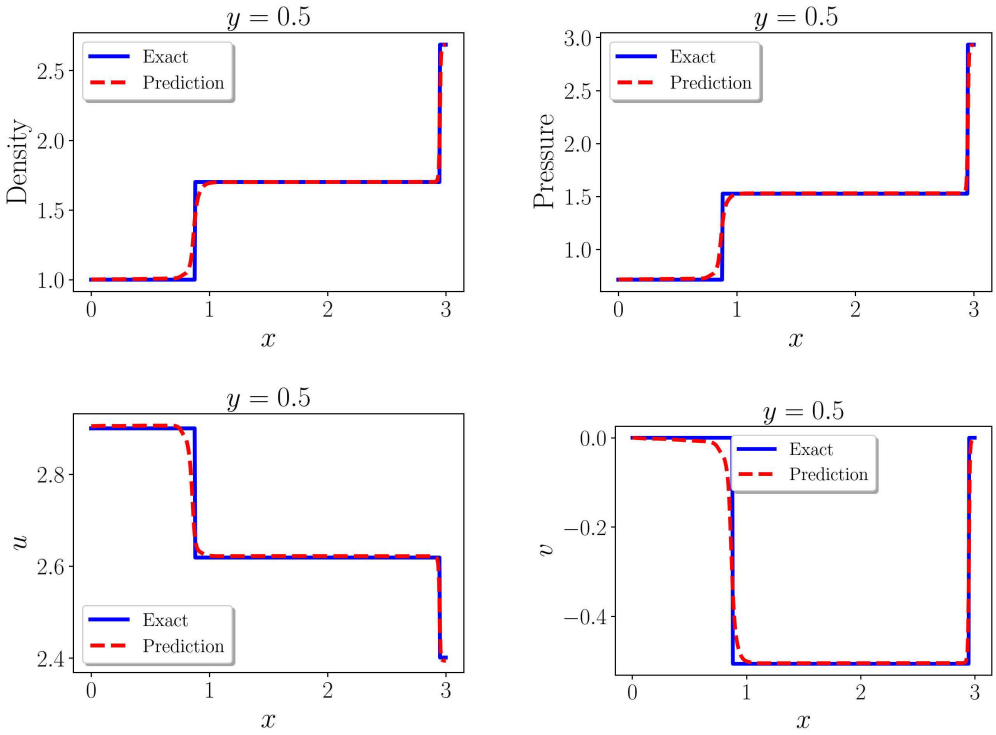


Figure 22: Compressible Euler equations: Exact vs NN solution comparison at $y=0.5$ location for all primitive variables.

Table 9: Neural network architecture in two subdomains for the two-dimensional compressible Euler equations.

| Subdomain number | 1 | 2 |
|---|---|---|
| # Layers | 6 | 6 |
| # Neurons | 20 | 25 |
| # Residual points | 12000 | 8000 |
| Adaptive activation function | sin | tanh |

Table 10: Compressible Euler equations: Relative $L_2$ errors in the two subdomains.

| | Density | $u$ | $v$ | Pressure |
|---|---|---|---|---|
| Subdomain 1 | 6.3287e-3 | 1.3674e-2 | 2.3101e-2 | 7.4893e-3 |
| Subdomain 2 | 9.2481e-3 | 3.1953e-2 | 3.2483e-2 | 9.3249e-3 |

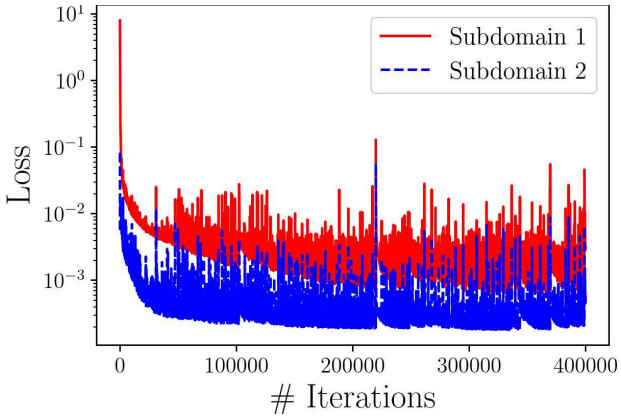


Figure 23: Compressible Euler equations: Loss history with number of iterations.

and pressure. The proposed method captures both the incident and the reflected shock waves accurately. Fig. 23 gives the loss variation for two sub-nets in the corresponding subdomains up to 400k iterations. Finally, Table 10 shows the relative $L_2$ errors in the two subdomains.

## 4.7 Inverse problem: One-dimensional viscous Burgers equation

In this section, we consider a problem of data-driven discovery of partial differential equations, which is an inverse problem. The parameterized viscous Burgers equation is written as

$$u_t + uu_x = \lambda u_{xx}, \quad x \in \Omega \subset \mathbb{R}, \quad t > 0,$$

where $u(x,t)$ is the solution. Given sparse observations of the solution $u(x,t)$, what is the parameter $\lambda$ that accurately describes the observed data? In this example, $\lambda = \nu =$

Table 11: One-dimensional viscous Burgers equation: Neural network architecture in both subdomains.

| Subdomain number | 1 | 2 |
|---|---|---|
| # Layers | 6 | 6 |
| # Neurons | 20 | 20 |
| # Residual points | 7000 | 3000 |
| Adaptive Activation function | tanh | sin |

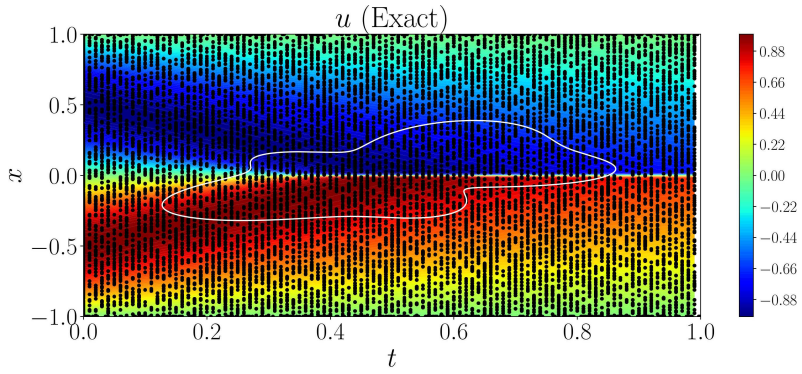


Figure 24: Training data points shown in black dots from the two subdomains.

0.003184713 is the viscosity coefficient. The computational space-time domain is divided into two subdomains as shown in Fig. 24, where the internal subdomain boundary is given by $r=0.3+0.1\sin(2\theta)+0.05\cos(5\theta)$, and the spatio-temporal interface points are obtained as $(t,x)=(0.5+r\cos(\theta),r\sin(\theta))$. The architecture of the network is given in Table 11.

Instead of providing the exact differential equation, we provide a library of differential terms given in the nonlinear operator $\mathcal{N}$ of the following differential equation

$$u_t=\mathcal{N}\left(x,t,u,u^2,\cdots,u_x,uu_x,u^2u_x,\cdots,u_{xx},uu_{xx},\cdots\right),$$

where

$$\mathcal{N}=c_0+c_1u+c_2u^2+\cdots+c_ku_x+c_{k+1}uu_x+c_{k+2}u^2u_x+\cdots+c_{k+m}u_{xx}+c_{k+m+1}uu_{xx}+\cdots.$$

Here, the network's job is to identify all the coefficients $c_i$'s present in the given differential equation. As discussed in [41], we could use sparse regression technique to determine these coefficients. For the viscous Burgers equation, we provide the additional dispersive term, i.e.,

$$u_t+uu_x=\nu u_{xx}+\alpha u_{xxx},\quad x\in\Omega\subset\mathbb{R},\quad t>0,$$

and now the aim is to identify the terms $\alpha,\nu$, which represent diffusion and dispersion coefficients, respectively in the given differential equation. Fig. 24 shows the number of

Table 12: Exact and predicted coefficients of viscous Burgers equation.

| | Exact $\nu$ | Predicted $\nu$ | Exact $\alpha$ | Predicted $\alpha$ |
|---|---|---|---|---|
| Subdomain 1 | 0.003184713 | 0.003221345 | 0.0 | 4.3245e-7 |
| Subdomain 2 | 0.003184713 | 0.003229753 | 0.0 | 3.1653e-7 |

training points in the entire domain. The total number of training points in each subdomain is [5500,3600]. The initial values of $\nu$,$\alpha$ in the two subdomains are $[2,\ 6]e-3$, $[1.0,\ -0.5]e-3$, respectively. Table 12 shows the exact and predicted values of diffusion and dispersion coefficients after 300k iterations, which are quite accurate.

# 5 Conclusions

We have proposed a generalized domain decomposition approach namely, the eXtended PINN (XPINN) method. Like the PINN method, the proposed method can be employed to solve any differential equation. This is achieved by enforcing the residual continuity condition along the common interfaces of neighboring subdomains. The residual continuity condition can theoretically enforce the solution regularity across the interface in the classical sense, i.e., the solution across the interface is sufficiently smooth such that it satisfies its governing PDE. We also enforce the average solution ($C^0$ solution continuity) given by two different neural networks along the common interface between two subdomains, which can increase the convergence rate. The XPINN has all the advantages of its predecessor, the conservative PINN (cPINN) method like deployment of separate neural network in each subdomain, efficient hyper-parameter adjustment (like depth, width, activation function, penalizing points, optimization method etc) for all networks, parallelization capacity, large representation capacity etc. Moreover, a key factor in the XPINN formulation is the flexibility in the division of subdomains for any type of differential equations. In particular, any irregularly shaped convex/non-convex space-time domain decomposition with $C^0$ or more regular boundary can be employed in the proposed approach. Due to residual continuity property, the interface conditions are easy to implement. Moreover, the interface conditions are very simple and in particular, universal, i.e, it can be applied to large class of problems in the field of mathematical physics. To verify our claim, various computational experiments show that the proposed method can easily and efficiently handle the highly irregular domains/subdomains for various differential equations. The major advantage of the XPINN is that it can be easily employed for any complex simulations involving complex domains, especially in higher dimensions. Overall, the proposed XPINN method is the generalization of PINN and cPINN approaches, both in terms of applicability as well as domain decomposition technique, which efficiently lends itself to parallelized computation. In ongoing work, we aim to parallelize XPINNs and measure the speed-up factor on multi-GPU platforms, which we expect to accelerate the training of XPINN by orders of magnitude.

## Acknowledgments

This work was supported by the Department of Energy PhILMs grant DE-SC0019453, the DARPA-AIRA grant HR00111990025, and the DARPA CompMods grant HR00112090062.

### References

[1] H. Lee and I. S. Kang, Neural algorithm for solving differential equations, Journal of Computational Physics, 91(1) (1990) 110-131.
[2] I.E. Lagaris, A. Likas, and D.I. Fotiadis, Artificial neural networks for solving ordinaryand partial differential equations, IEEE transactions on neural networks, 9(5) (1998) 987-1000.
[3] I.E. Lagaris, A.C. Likas, and D.G. Papageorgiou, Neural-network methods for boundary-value problems with irregular boundaries, IEEE Transactions on Neural Networks, 11(5) (2000) 1041-1049.
[4] D.C. Psichogios and L.H. Ungar, A hybrid neural network-first principles approach to process modeling, AIChE J., 38, 1499 (1992).
[5] R. S. Beidokhti and A. Malek, Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques, J. Franklin Inst., 346, 898 (2009).
[6] M. Hirn, S. Mallat, and N. Poilvert, Wavelet scattering regression of quantum chemical energies, Multiscale Model. Simul., 15, 827 (2017).
[7] S. Mallat, Understanding deep convolutional networks, Phil. Trans. R. Soc. A, 374, 20150203 (2016).
[8] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, and J.M. Siskind, Automatic differentiation in machine learning: A survey, Journal of Machine Learning Research, 18 (2018) 1-43.
[9] S. Wang, Y. Teng and P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, arxiv:2001.04536v1, 2020.
[10] C. Basdevant et al., Spectral and finite difference solution of the Burgers equation, Comput. Fluids, 14 (1986) 23-41.
[11] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deeplearning. Proceedings of the National Academy of Sciences, 115(34):8505-8510, 2018.
[12] D. Zhang, L. Guo, and G.E. Karniadakis. Learning in modal space: Solving time-dependentstochastic PDEs using physics-informed neural networks. SIAM Journal on Scientific Computing, 42(2):A639-A665, 2020.
[13] G. Pang, L. Lu, and G.E. Karniadakis. fpinns: Fractional physics-informed neural networks. SIAM Journal on Scientific Computing, 41(4):A2603-A2626, 2019.
[14] E. Kharazmi, Z. Zhang, G.E. Karniadakis, Variational Physics-Informed Neural Networks for Solving Partial Differential Equations, arXiv:1912.00873.
[15] Y. Yang and P. Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. Journal of Computational Physics, 394:136-152, 2019.
[16] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deeplearning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics, 394:56-81, 2019.
[17] S. Mishra, R. Molinaro, Estimates on the generalization error of Physics Informed Neural-Networks (PINNs) for approximating PDEs, arXiv:2006.16144v1, 2020.

[18] S. Mishra, R. Molinaro, Estimates on the generalization error of Physics Informed-Neural Networks (PINNs) for approximating PDEs II: A class of inverse problems, arXiv:2007.01138v1, 2020.
[19] K. Shukla, P.C.D. Leoni, J. Blackshire, D. Sparkman, G.E. Karniadakis, Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks, arXiv:2005.03596v1, 2020.
[20] D.P. Kingma, J.L. Ba, ADAM: A method for stochastic optimization, arXiv:1412.6980v9, 2017.
[21] A.D. Jagtap, K. Kawaguchi and G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys., 404 (2020) 109136.
[22] A.D. Jagtap, K. Kawaguchi and G.E. Karniadakis, Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks, Proc. R. Soc. A, 476: 20200334 (2020). http://dx.doi.org/10.1098/rspa.2020.0334
[23] Z. Mao, A.D. Jagtap and G.E. Karniadakis, Physics-informed neural network for high-speed flows, Computer Methods in Applied Mechanics and Engineering, 360 (2020) 112789.
[24] K. Li, K. Tang, T. Wu, Q. Liao, D3M: A Deep Domain Decomposition Method for Partial Differential Equations, in IEEE Access, vol. 8, pp. 5283-5294, 2020.
[25] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pp. 1097-1105, 2012.
[26] M. Raissi, H. Babaee, and P. Givi, Deep learning of turbulent scalar mixing, Physical Review Fluids, 4(12) (2019) 124501.
[27] X. Jin, S. Cai, H. Li, G.E. Karniadakis, NSFnets (Navier-Stokes Flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, arXiv preprint arXiv:2003.06496.
[28] C. Rao, H. Sun, Y. Liu, Physics informed deep learning for computational elastodynamics without labeled data, arXiv preprint arXiv:2006.08472, 2020.
[29] J.-X. Wang, J. Wu, J. Ling, G. Iaccarino, and H. Xiao, A comprehensive physics-informed machine learning framework for predictive turbulence modeling, Phys. Rev. Fluids, 3, 074602 (2018).
[30] A.D. Jagtap, Y. Shin, and G.E. Karniadakis, Kronecker neural networks: A general framework for neural networks with adaptive activation functions (in preparation).
[31] Y. Shin, J. Darbon, and G.E. Karniadakis, On the convergence and generalization of physics informed neural networks, arXiv:2004.01806v1, 2020.
[32] L.D. Landau, and E.M. Lifshits, Theory of Elasticity. Pergamon Press, Oxford, 1986.
[33] H. Owhadi, Bayesian numerical homogenization, Multiscale Model. Simul., 13, 812-828, 2015.
[34] S.L. Brunton, J.L. Proctor, and J.N. Kutz. Discovering governing equations from data by sparseidentification of nonlinear dynamical systems. Proceedings of the National Academy of Sciences, 113, 3932-3937, 2016.
[35] M. Abadi et al., Tensorflow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 16, 2016, pp. 265-283.
[36] Q. Zheng, L. Zeng, and G.E. Karniadakis, Physics-informed semantic inpainting: Application to geostatistical modeling, arXiv preprint arXiv:1909.09459.
[37] M. Raissi, P. Perdikaris, and G.E. Karniadakis, Physics-informed neural network: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys., 378, 686-707, 2019.

[38] J. Sirignano and K. Spiliopoulos, DGM, A deep learning algorithm for solving partial differential equations, J. Comput. Physics, 375 (2018) 1339-1364.
[39] J. Ling, A. Kurzawski, and J. Tompson, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, J. Fluid Mech., 807 (2016) 155-166.
[40] J. Tompson et al., Accelerating Eulerian fluid simulation wtih convolutional networks, in: Proceedings of Machine learning research, vol. 70, 2017, pp 3424-3433.
[41] S. Ruder, An overview of gradient descent optimization algorithms, arXiv:1609.04747v2, 2017.
[42] M. Raissi, A. Yazdani, G. E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science, 367 (6481) (2020) 1026-1030.
[43] G. Kissas, Y. Yang, E. Hwuang, W. R. Witschey, J. A. Detre, and P. Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering, 358 (2020) 112623.
[44] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, and E. Kuhl, Physics-informed neural networks for cardiac activation mapping, Frontiers in Physics, 8 (2020) 42.
[45] M. Raissi, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis, Deep learning of vortex-induced vibrations, Journal of Fluid Mechanics, 861 (2019) 119-137.
[46] A. D. Jagtap, E. Kharazmi, and G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering, 365 (2020) 113028.
[47] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. InAistats, volume 9, pages 249-256, 2010.
[48] E. Kharazmi, Z. Zhang, and G.E. Karniadakis, hp-VPINNs: Variational Physics-Informed Neural Networks with Domain Decomposition, arXiv:2003.05385.
[49] H.C. Yee, R.F. Warming, and A. Harten, A high-resolution numerical technique for inviscid gas-dynamics problems with weak solutions, in: Proceedings in Eight International Conference on Numerical Methods in Fluid Dynamics, in: Lecture notes in Physics, vol. 170, Springer, New York/Berkin, 1982, pp. 546-552.