

UKAEA-CCFE-PR(23)186

Prakhar Sharma, Llion Evans, Michelle Tindall,
Perumal Nithiarasu

Stiff **HP** PDEs and Physics **HP** Informed Neural Networks

Enquiries about copyright and reproduction should in the first instance be addressed to the UKAEA Publications Officer, Culham Science Centre, Building K1/O/83 Abingdon, Oxfordshire, OX14 3DB, UK. The United Kingdom Atomic Energy Authority is the copyright holder.

The contents of this document and all other UKAEA Preprints, Reports and Conference Papers are available to view online free at scientific-publications.ukaea.uk/

Stiff PDEs and Physics Informed Neural Networks

Prakhar Sharma, Llion Evans, Michelle Tindall, Perumal Nithiarasu



Stiff-PDEs and Physics-Informed Neural Networks

Prakhar Sharma^{1,4} · Llion Evans^{1,3,4} · Michelle Tindall² · Perumal Nithiarasu^{1,4}

Received: 28 October 2022 / Accepted: 19 January 2023 / Published online: 7 February 2023
© The Author(s) 2023

Abstract

In recent years, physics-informed neural networks (PINN) have been used to solve stiff-PDEs mostly in the 1D and 2D spatial domain. PINNs still experience issues solving 3D problems, especially, problems with conflicting boundary conditions at adjacent edges and corners. These problems have discontinuous solutions at edges and corners that are difficult to learn for neural networks with a continuous activation function. In this review paper, we have investigated various PINN frameworks that are designed to solve stiff-PDEs. We took two heat conduction problems (2D and 3D) with a discontinuous solution at corners as test cases. We investigated these problems with a number of PINN frameworks, discussed and analysed the results against the FEM solution. It appears that PINNs provide a more general platform for parameterisation compared to conventional solvers. Thus, we have investigated the 2D heat conduction problem with parametric conductivity and geometry separately. We also discuss the challenges associated with PINNs and identify areas for further investigation.

Acronyms

PINN	Physics-informed neural network
PDE	Partial differential equation
IC	Initial condition
BC	Boundary condition
NN	Neural network
ODE	Ordinary differential equation
DGM	Deep Galerkin method
LSTM	Long short-term memory

DNN	Deep neural network
FCNN	Fully connected neural network
SiReNs	Sinusoidal Representation Network
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
MSE	Mean squared error
SDF	Signed distance function
FEM	Finite element method
FDM	Finite difference method

Llion Evans, Michelle Tindall, and Perumal Nithiarasu have contributed equally to this work.

✉ Prakhar Sharma
prakhars962@gmail.com

Llion Evans
llion.evans@swansea.ac.uk

Michelle Tindall
michelle.tindall@ukaea.uk

Perumal Nithiarasu
p.nithiarasu@swansea.ac.uk

- ¹ Faculty of Science and Engineering, Swansea University, Bay Campus, Swansea SA1 8EN, UK
- ² Fusion Technology Facilities, United Kingdom Atomic Energy Authority, Unit 2a Lanchester Way, Rotherham S60 5FX, UK
- ³ Culham Science Centre, United Kingdom Atomic Energy Authority, Abingdon OX14 3DB, UK
- ⁴ Zienkiewicz Institute for Modelling, Data and AI, Swansea University, Bay Campus, Swansea SA1 8EN, UK

1 Introduction

For a variety of problems, conventional numerical techniques for solving partial differential equations (PDEs) remain difficult. The mesh generation is complicated, noisy experimental data can't be integrated with the existing algorithms and high-dimensional parametric PDEs often can't be solved. Ill-posed inverse problems are prohibitively expensive and require complicated algorithms. In an effort to address these issues, the development of powerful computers and machine learning libraries has taken research in physics-informed machine learning to new heights.

Recently, physics-informed neural networks (PINNs) have gained popularity due to the novel approach for solving forward [1–4] and inverse problems [5–7] involving PDEs using neural networks (NNs). Unlike conventional numerical techniques for solving PDEs, PINNs are non-data-driven meshless models that satisfy the prescribed initial (IC) and

boundary conditions (BC) as well as the governing PDE. The differential terms in the PDE are approximated using automatic differentiation.

Although, NNs have been present in literature for a long time, it was only in the last decade that they started gaining immense popularity due to the recent advancement in automatic differentiation and the availability of open-source machine learning libraries such as PyTorch and TensorFlow. It is automatic differentiation [8] producing highly accurate derivatives that enhances the accuracy of PINNs even with sparse data points in the spatio-temporal domain.

The numerical solution of ordinary differential equations (ODEs) and PDEs using NNs [9–15] has been a key area of interest for researchers. This is because the NNs behave like meshless solvers and can be scaled to higher dimensions without the need of additional algorithms.

In this review, we have investigated stiff-PDEs [16], specifically those with a discontinuous solution, such as conflicting BCs at adjacent edges or faces. The reason for this focus is due to the significant challenge posed by solving such problems with PINNs because it is not possible to fit a smooth curve such as a NN on these discontinuities.

The baseline PINN, a generalised NN framework, was proposed by Raissi et al. [17] for solving forward and inverse problems involving PDEs. The baseline PINN takes in the independent variables as the input and gives out the dependent variables as the output, and, depending on the type of problem we construct, the loss function. The baseline PINN accurately predicted the solution of forward problems such as the 1D Burgers equation, 1D Schrodinger equation, 1D Allen–Cahn equation and inverse problems such as recovering the 2D Navier–Stokes equation from the finite element generated flow past a cylinder simulation.

The baseline PINN has several limitations such as scalability to higher dimensions, imbalanced magnitude of individual loss terms in the multiple task loss function, gradient explosion etc. A brief description of these issues has been given in this section and are explored further in the subsequent section.

The baseline PINN can be scaled to higher dimensions by modifying the architecture of the NN. A general approach is to deploy more activated neurons (~ 500) with very few hidden layers (~ 4). This approach helps in fitting complex local variations in the solution. Wang et al. [18] proposed a fully connected NN (FCNN) with two transformer networks [19–21], which projects the input variables to a high-dimensional feature space. Sirignano and Spiliopoulos [22] proposed a deep Galerkin method (DGM) architecture influenced by the LSTM architecture [23]. Both these architectures attempt to encapsulate the complex local variations in the solution by using more complex NN architectures.

Spectral bias [24] is a learning bias of deep NNs (DNNs) towards low-frequency functions, which causes convergence

issues during training. Novel architectures such as Fourier networks [25], Modified Fourier networks [26, 27], Sinusoidal Representation networks, SiReNs [28] were proposed to remove the spectral bias from computer vision problems.

Imbalanced loss terms appear when the magnitude of a loss term is significantly larger than other loss terms, resulting in early convergence of other loss terms in the multi-task loss function. A predominant approach is to multiply a parameter λ to each loss term to balance out the contribution of each term to the overall loss [29]. Several frameworks such as self-adaptive PINNs [30] and the self-adaptive weight PINN [31]; algorithms such as learning rate annealing [18] and neural tangent kernel [32] were proposed to balance out the contribution of each term to the overall loss by introducing adaptive coefficients for each loss term.

Gradient explosion [33, 34] is a known issue while training NNs. The baseline PINN uses L-BFGS [35], a second order optimiser, which exhibits exploding gradients when it encounters sharp gradients [36].

Most of the PINN frameworks are readily available as open-source libraries. Libraries such as DeepXDE [37], SciANN [38], TensorDiffEq [39] and NeuralPDE [40] are easy to implement and can be used to solve simple problems in the 1D and 2D spatial domain. NVIDIA Modulus (formerly NVIDIA SimNet) [26, 27] is an advanced open-source library providing access to different PINN frameworks. NVIDIA Modulus has successfully simulated physical systems such as laminar flow over a field programmable gate arrays (FPGA) heatsink, turbulent flow over a simple 3D heatsink with parameterised fin dimensions, conjugate heat transfer over NVIDIA's NVSwitch heat sink using transfer learning.

This review paper has been divided into five major sections. Section 1 comprises a comprehensive literature review of PINNs. Section 2 discusses the functioning of a baseline PINN [17] and some advanced tools that enhance the performance of PINNs. Section 3 discusses different PINN frameworks that are available in DeepXDE [37] and NVIDIA Modulus [26]. Sections 4 and 5 discuss the solution of 2D and 3D heat conduction test case using various PINN frameworks. Section 6 discusses the results of a series of parametric PDEs. In Sect. 7, we discuss challenges related to implementing PINNs for stiff-PDEs.

We took two heat conduction problems (2D and 3D) with a discontinuous solution at corner points as test cases. In Sects. 4 and 5, we investigated these problems with a number of PINN frameworks from Table 1 and compared the results with the FEM solution. PINNs are also known for solving parametric PDEs [41–44]. In Sect. 6, we investigated the 2D test case with parameterised conductivity and geometry.

For the forward problems, we used the baseline PINN, DeepXDE's baseline PINN, NVIDIA Modulus's baseline

PINN, Fourier network, Modified Fourier network, SiReNs and DGM architecture. Whereas for the parametric PDE, we have only used the DeepXDE’s baseline PINN, Modified Fourier network and DGM architecture.

2 How PINNs Work

PINNs are DNNs that obey the physical constraints, such as, the BCs, the ICs and the governing PDE.

PINNs are fundamentally different from conventional PDE solvers such as finite difference or finite elements, in terms of the how the system of equations are represented. In the case of PINNs, the number of equations or the data points are more than the number of unknowns or the network parameters, resulting in a regression problem. On the other hand, the conventional PDE solvers, such as FEM, often solves the system of equations with the number of equations and unknowns being equal.

This section discusses the theoretical aspects of PINNs. Section 2.1 discusses the functioning of NNs. Section 2.2 discusses how the physics knowledge is embedded into the NN. Section 3 discusses the various PINN frameworks, that can solve stiff-PDEs. Finally, Sect. 2.3 discuss various tools that can help in speeding the convergence and improving the accuracy of PINNs.

2.1 Neural Networks for Regression

NNs are nonlinear and non-convex regression frameworks with exceptional predictive capability, widely known as universal approximators of continuous functions [45, 46]. They are known for their ability to learn and generalise very complicated information [47]. This section discusses the evolution of NNs.

One of the simplest regression models is the linear regression. In the case of linear regression, the hypothesis or the trial function is the linear combination of weights and features. Let us assume a dataset with n samples consisting of m features with a bias term $\mathbf{X} = \{1, \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \dots, \mathbf{x}_m\}$ and one label \mathbf{y} . The linear regression hypothesis for the dataset can be written as follows

$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + \dots + w_mx_m = \mathbf{W}^T \mathbf{X}, \tag{1}$$

where the trainable weights $\mathbf{W} = \{w_0, w_1, w_2, \dots, w_m\}$ are tweaked such that the mean squared error (MSE) or any loss metric of the ground truth \mathbf{y} against the predicted output \hat{y} is minimised.

In principle, we can introduce a hypothesis with a linear combination of weights with nonlinear features too. The

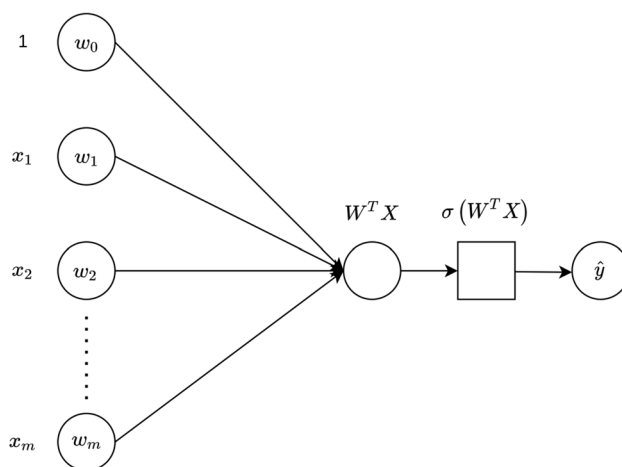


Fig. 1 Schematic diagram of an artificial neuron. Here, $\mathbf{X} = \{1, x_1, x_2, \dots, x_m\}$ is the input dataset, $\mathbf{W} = \{w_0, w_1, w_2, \dots, w_m\}$ is the vector of trainable weights/parameters, σ is the activation function and \hat{y} is the predicted output

main drawback of linear regression is underfitting because it essentially fits a linear equation onto the data points [48, 49].

A simple technique to prevent underfitting is to pass the output of linear regression hypothesis (Eq. 1) to specifically chosen nonlinear function, often referred to as the activation function (σ) [50]. These activation functions, for example, sigmoid, hyperbolic tangent, softmax are continuous and sometimes are infinitely differentiable in the domain of real numbers. The resulting hypothesis (Eq. 2) is referred to as an artificial neuron (Fig. 1) [51]. It transforms the linear regression hypothesis into a nonlinear feature space.

$$\hat{y} = \sigma(w_0 + w_1x_1 + w_2x_2 + \dots) = \sigma(\mathbf{W}^T \mathbf{X}). \tag{2}$$

The artificial neuron is the building block of a NN. DNNs have multiple hidden-layers, each hidden-layer consisting of multiple artificial neurons thus increasing the predictive capability even further. Each hidden layer transforms the feature space into a more complex feature space [52]. It is an open question in interpretable machine learning to explain mathematically, how these nonlinear transformations influence the output [53]. Figure 2 shows the schematic diagram of a FCNN with two hidden layers.

In Fig. 2, there are four layers, one input layer, two hidden layers and one output layer. Both the hidden layers consist of three artificial neurons, also known as the activated neurons or simply the nodes. In a FCNN, each node has its own set of weights or parameters.

Let us define some notation, $a_i^{(k)}$ is the i th activated neuron in the k th layer. w^k is the weight matrix controlling the nonlinear mapping between k th layer and $(k + 1)$ th layer. If the network has s_k activated neurons in k th layer and $s_{(k+1)}$

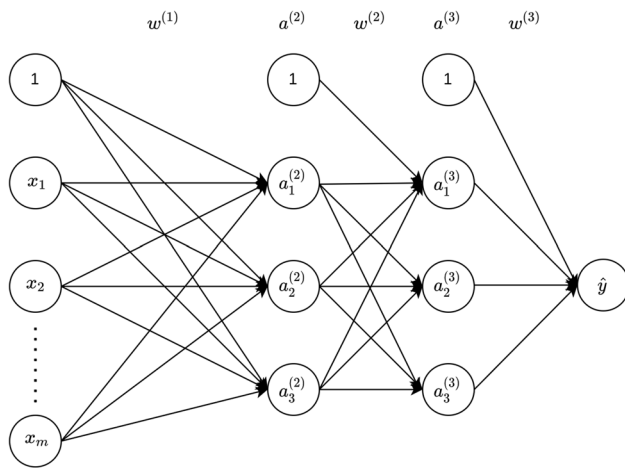


Fig. 2 Schematic diagram of a fully connected neural network (FCNN) with two hidden layers. Here, $X = \{1, x_1, x_2, \dots, x_m\}$ is the input dataset, $\{w^{(1)}, w^{(2)}, w^{(3)}\}$ is the vector of trainable weights/parameters for each layer, $\{a^{(2)}, a^{(3)}\}$ is the vector of activated layers and \hat{y} is the predicted output

activated neurons in $(k + 1)$ th layer then w^k will be of dimension $s_{(k+1)} \times s_k$.

2.1.1 Forward Pass

The first step in training of a NN is to forward pass the inputs X to obtain the predicted output \hat{y} . Let $a_i^{(k)} = \sigma(z_i^{(k)})$ such that z is the term without the activation. Now, the forward pass for the FCNN in Fig. 2 can be written as follows:

$$\begin{aligned} z^{(2)} &= w^{(1)}X, \\ a^{(2)} &= \sigma(z^{(2)}), \\ z^{(3)} &= w^{(2)}a^{(2)}, \\ a^{(3)} &= \sigma(z^{(3)}), \\ \hat{y} &= w^{(3)}a^{(3)}. \end{aligned} \quad (3)$$

In regression problems, the last layer is not activated because we want unbounded values. The hypothesis of a FCNN with two hidden layers can be written as follows:

$$\hat{y} = w^{(3)}\sigma(w^{(2)}\sigma(w^{(1)}X)). \quad (4)$$

2.1.2 Backpropagation and Automatic Gradient

Once we have obtained the predicted output \hat{y} from the forward pass, we can calculate the difference from the ground truth y using some loss metric [54]. In general, for a regression problem, the MSE (Eq. 5) is used as a loss metric.

$$J = \frac{1}{2n} \sum_{i=1}^n |y_i - \hat{y}_i|^2. \quad (5)$$

Now, the loss J is sent to the optimisers. Optimisers are algorithms that minimise the loss by updating the weights or parameters of the NN. There are many optimisers that are available in open-source libraries such as PyTorch and TensorFlow [55]. On the basis of order of the derivatives of the loss with respect to the weights, the optimisers can be divided mainly into two categories: first order and second order optimisers [56].

The first-order optimisers utilise the gradient of the loss with respect to weights of the NN. Similarly, second-order optimisers utilise the gradient as well as the Hessian of the loss with respect to the weights of the NN. Gradient-descent [57] and Adam [58] are the most popular first-order optimisers. Whereas, L-BFGS [35] is the only second-order optimiser that is still actively used in machine learning.

Now that the optimisers require derivatives, we need to compute the derivatives efficiently and accurately. There are three main techniques that have been successfully used in machine learning: numerical [59], symbolic [60] and automatic differentiations [8]. After the release of TensorFlow 1.0 in 2015, static computational graphs was the standard data structure for representing NNs, which was later substituted with dynamic computational graphs in PyTorch [61, 62]. These libraries use forward-mode or reverse-mode automatic differentiation to compute the derivatives within a computational graph [63]. The automatic differentiation computes the derivative of the loss with respect to weights using the chain rule [64] in differential calculus with a hard-coded expression for the derivative of simple functions [65].

Let us say that we are using the gradient-descent method (Eq. 6) for optimising the weight/parameters.

$$w^{(k)} := w^{(k)} - \alpha \frac{\partial J}{\partial w^{(k)}}, \quad (6)$$

where k is the k th layer and α is a constant often referred to as the learning rate, a hyperparameter. It tells the optimiser the amount of perturbation the weights should attain in each epoch [66]. An epoch is a complete cycle of forward pass and backpropagation on the whole dataset.

The gradient of the loss J with respect to the weights in each layer can be computed using the chain rule as follows:

$$\begin{aligned} \frac{\partial J}{\partial w^{(3)}} &= \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w^{(3)}}, \\ \frac{\partial J}{\partial w^{(2)}} &= \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial w^{(2)}}, \\ \frac{\partial J}{\partial w^{(1)}} &= \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}} \frac{\partial z^{(3)}}{\partial a^{(2)}} \frac{\partial a^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial w^{(1)}}. \end{aligned} \quad (7)$$

The analytical expression of the basic differential terms are hard-coded in machine learning libraries. The derivative of loss with respect to the weights is decomposed into these hard-coded basic differential terms using the chain rule. Once, these derivatives are calculated we plug them into Eq. 6 to update the weights/parameters. The process is repeated for a number of epochs until the loss J goes below some predefined tolerance.

2.2 Building a PINN

PINNs are supervised machine learning models that obey the prescribed BCs and the governing PDE. In PINNs, the inputs are the independent variables and the outputs are the dependent variables or the solution of the governing PDE.

As an example, for a 3D time-dependent heat conduction PDE, the number of inputs would be four, i.e., the x , y and z coordinates and the time t . Similarly, the number of outputs would be one, i.e. the temperature u .

2.2.1 Defining a Well-Posed PDE

Consider a well-posed PDE problem as follows:

$$\begin{aligned} u_t + \mathcal{N}_x[u] &= 0, & \mathbf{x} \in \Omega, & \quad t \in [0, T], \\ u(\mathbf{x}, 0) &= h(\mathbf{x}), & \mathbf{x} \in \Omega, & \\ u(\mathbf{x}, t) &= g(\mathbf{x}, t), & \mathbf{x} \in \partial\Omega, & \quad t \in [0, T], \end{aligned} \tag{8}$$

where $\mathcal{N}_x[u]$ is a differential operator, \mathbf{x} and t are the independent variables of the PDE, Ω and $\partial\Omega$ denotes the spatial domain and the boundary of the problem, $h(\mathbf{x})$ denotes the prescribed BC which is the solution to the PDE at all spatial points (Ω) at the initial time ($t = 0$), $g(\mathbf{x}, t)$ is the prescribed BC at the boundary of the domain ($\partial\Omega$).

2.2.2 Discrete Loss Function

PINNs have a multitask loss function with at least two components, the BC and the PDE losses (Eq. 9). The multitask loss function ensures that the inputs to the PINN satisfy a well-posed PDE problem.

The IC and BC losses are simply the MSE and the PDE loss is the residual of the governing PDE at randomly chosen collocation points. The total loss (Eq. 9) and the individual loss terms (Eq. 10) are as follows:

$$\mathcal{L} = \lambda_{PDE} \mathcal{L}_{PDE} + \lambda_{BC} \mathcal{L}_{BC} + \lambda_{IC} \mathcal{L}_{IC}, \tag{9}$$

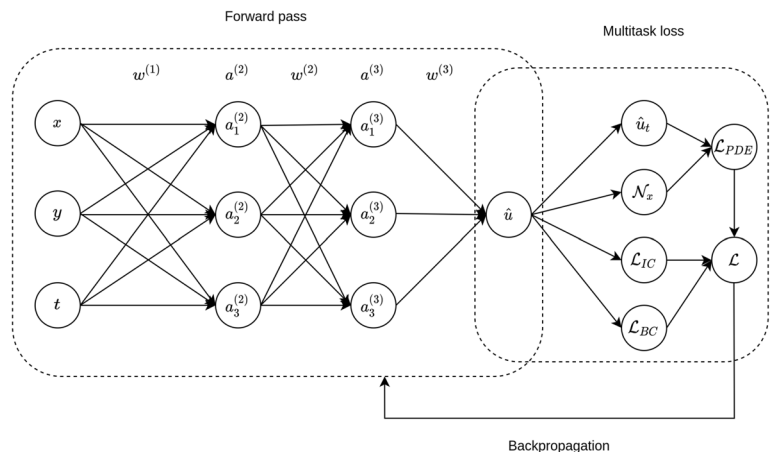
$$\begin{aligned} \mathcal{L}_{PDE} &= \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \hat{u}_t(\mathbf{x}_i, t_i) + \mathcal{N}_x[\hat{u}(\mathbf{x}_i, t_i)] \right|^2, \\ \mathcal{L}_{BC} &= \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \hat{u}(\mathbf{x}_i, t_i) - g(\mathbf{x}_i, t_i) \right|^2, \\ \mathcal{L}_{IC} &= \frac{1}{N_0} \sum_{i=1}^{N_0} \left| \hat{u}(\mathbf{x}_i, 0) - h(\mathbf{x}_i, 0) \right|^2, \end{aligned} \tag{10}$$

where N_r , N_b and N_0 are the number of data points that are sampled to satisfy the PDE loss, the BC loss and the IC loss. The coefficients λ_{PDE} , λ_{BC} and λ_{IC} in Eq. 9, help in achieving convergence and better accuracy, and are an active field of research. These weight coefficients can be scalar quantities [29] as well as vectors to apply weights to every single sample in the training dataset based on the pointwise error [27].

We can calculate the residual of the governing equation using automatic differentiation similar to Eq. 4. Figure 3, shows a schematic diagram of a baseline PINN with two hidden layers for a 2D spatio-temporal domain.

A PINN framework, referred to as sparse-regulated PINNs, uses the experimental or the simulation data to provide ground truth at sparse locations in the spatio-temporal domain [67, 68]. The addition of sparse ground truth helps the NN to accurately predict complex local

Fig. 3 Schematic diagram of a physics-informed neural network (PINN) with two hidden layers for a 2D spatio-temporal domain. Here, the inputs are $\{x, y, t\}$, $\{w^{(1)}, w^{(2)}, w^{(3)}\}$ is the vector of trainable weights/parameters for each layer, $\{a^{(2)}, a^{(3)}\}$ is the vector of activated neurons for each layers and \hat{y} is the predicted output



variations in the solution. This introduces one more loss term called the reconstruction loss (Eq. 11), because it points the optimiser towards the true solution.

$$\mathcal{L}_{reconstruct} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left| \hat{u}(\mathbf{x}_i, t_i) - u_i^{true} \right|^2, \quad (11)$$

where N_d is the number of true solutions being provided. Bajaj et al. [69] showed that the reconstruction loss is not very helpful in preventing overfitting. Instead, NVIDIA Modulus uses the Adam optimiser with exponential decay of the learning rate α , so that as we go closer to the minima of the loss, the weights do not attain large perturbations [58].

2.2.3 Integral Formulation of Loss

NVIDIA Modulus uses a slightly different version of the loss function [27, 70]. They define continuous/integral losses for the BC and PDE loss as follows:

$$\begin{aligned} \mathcal{L}_{PDE} &= \int_{\Omega} \left(\hat{u}_t(\mathbf{x}_i, t_i) + \mathcal{N}_x[\hat{u}(\mathbf{x}_i, t_i)] \right)^2 d\Omega, \\ \mathcal{L}_{BC} &= \int_{\partial\Omega} \left(\hat{u}(\mathbf{x}_i, t_i) - g(\mathbf{x}_i, t_i) \right)^2 d(\partial\Omega). \end{aligned} \quad (12)$$

Instead of approximating these integrals using deterministic numerical integration techniques, NVIDIA Modulus uses the Monte Carlo integration, a non-deterministic integration technique [71, 72], resulting in Eq. 13. This helps in keeping a specific loss term proportional to its length/area/volume. For example, it doesn't allow a specific BC applied over a relatively larger area to dominate other BCs.

$$\begin{aligned} \mathcal{L}_{PDE} &= \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \hat{u}_t(\mathbf{x}_i, t_i) + \mathcal{N}_x[\hat{u}(\mathbf{x}_i, t_i)] \right|^2 \int_{\Omega} d\Omega, \\ \mathcal{L}_{BC} &= \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \hat{u}(\mathbf{x}_i, t_i) - g(\mathbf{x}_i, t_i) \right|^2 \int_{\partial\Omega} d(\partial\Omega). \end{aligned} \quad (13)$$

2.2.4 Exact BC Imposition

The output of the PINN can be hard constrained to exactly satisfy the BCs [73]. A function is manually constructed to transform the network outputs to exactly satisfy the BCs. Generally, hard constrained BCs work with simple geometry, constant individual Dirichlet BCs without conflicting each other. This approach does not work well with stiff-PDEs [74]. In Sect. 4.4, we have discussed the difficulties with exact imposition of BCs in stiff-PDEs with a discontinuous solution.

2.2.5 Overfitted vs. Generalised Solution

The baseline PINN gained its popularity due to the notion that it can solve PDEs with sparse sampling in the spatio-temporal domain. Later on it was realised that, in the case of stiff-PDEs one must sample enough data points to capture the local variations in the solution.

Theoretically, one can obtain an overfitted model as well as a generalised model depending on the number of data points in the training dataset. An overfitted model exhibits low training loss but high validation and testing loss whereas a generalised model exhibits low training, validation and testing error. Thus overfitted model can only be used for inferring the solution from the training dataset, i.e., any data point of interest should be included in the training dataset. On the other hand, a generalised model can be obtained by sampling a significantly greater number of data points in the spatio-temporal domain. The solution can be predicted at new spatio-temporal locations within the domain.

The overfitted or generalised model is a qualitative aspect that depends on human judgement. In the case of stiff-PDEs, the overfitted model may converge with a considerable amount of pointwise error. Thus, it is important to sample more points around the boundary or the problematic region. In this paper, to be on the safer side, we have aimed for a generalised solution, i.e., we sampled a large number of points. Specific details can be found in Sect. 4.5.

2.2.6 Parametric PINNs

PINNs can predict the variation in the solution for a range of parameters such as density, geometry, conductivity etc. by introducing them as features in the training data set, compared to conventional numerical solvers where each parameter needs a separate simulation and may require complex algorithms. The idea is to add the parameter as another feature into the training data set such that each parameter has its own set of sampled points in the spatio-temporal domain. In other words, if there are n parameters for a 2D steady-state problem, than the training data set contains $n + 2$ features.

2.3 Additional Tools

In this section, we have discussed tools that enhance the accuracy and efficiency of PINNs. Outside the scope of this paper are tools such as gradient enhanced training [75], learning rate annealing [18], neural tangent kernel [32], integral continuity planes [27]. The gradient enhanced training does not always improve the results compared to the baseline PINN. They may even adversely affect the training convergence and accuracy [76]. The learning rate annealing and neural tangent kernel deals with imbalanced losses where there is a significant difference between the magnitude of

individual losses, which is not applicable to our test cases. The integral continuity planes are only applicable to problems involving fluid flow, where the mass flow rate or the volumetric flow rate is applied as an additional constraint, if known. This is particularly useful in the case of channel flow.

2.3.1 Adaptive Activation

An activation function transforms a feature space into a more complex feature space with the help of nonlinear functions. Without the activation function, the NN is just a sophisticated linear model with no performance improvement over linear regression. Jagtap et al. [77] proposed a DNN framework with trainable nonlinear transformations to improve the convergence as well as the accuracy of the NNs. They introduced an additional hyperparameter A in the nonlinear transformation of the hidden layers as follows:

$$\mathbf{a}^{(k)}(\mathbf{a}^{(k-1)}; \theta, A) = \sigma(A\mathbf{z}^{(k)}(\mathbf{a}^{(k-1)})), \tag{14}$$

where $\mathbf{a}^{(k)}$ is the nonlinear transformation at layer k , which is a function of $\mathbf{a}^{(k-1)}$, the output of the hidden layer $(k - 1)$, the weights and biases is denoted by θ , A is a trainable parameter and $\mathbf{z}^{(k)}$ is the linear transformation at layer k . As A is a trainable parameter, it gets updated in each epoch based on the total loss \mathcal{L} . Thus the activation function \mathbf{a} adapts itself to minimise the total loss \mathcal{L} .

2.3.2 Signed Distance Function

While there have been several efforts to solve stiff-PDEs with discontinuity inside the spatial domain, so far, there is only one viable technique to solve stiff-PDEs with conflicting BCs at the adjacent edges and corners. The difficulty lies in the fact that the activation functions are smooth, i.e., they are differentiable and can't capture discontinuous BCs. Thus, currently the only way to alleviate the issue is to exclude the points with a discontinuity from the training data set.

In DeepXDE, these corner points are excluded by default, as the normal vector at those corners are not defined, or in other words the derivative at those points are not defined, so any BC that includes a derivative for example, a Neumann BC is not defined at the corners.

NVIDIA Modulus and several other literature takes this one step further by using signed distance function (SDF) weights [74, 78, 79]. SDF weights are used to assign minuscule weights around the region with conflicting BCs. This way a region with a discontinuous solution gets a lower priority compared to the regions where the solution is smooth. The application of SDF weights on complex geometry is an active field of research.

2.3.3 Importance Sampling

Nabian et al. [70] proposed a sampling strategy for efficient training of PINNs based on an approximation method called importance sampling [80], which is often used in reinforcement learning for approximating the expected reward based on the older policy [81, 82]. In optimisation, the optimal parameters θ^* is defined such that

$$\begin{aligned} \theta^* &= \operatorname{argmin}_{\theta} \mathbb{E}_f[\mathcal{L}(\theta)] \\ &\approx \operatorname{argmin}_{\theta} \frac{1}{N} \sum_{j=1}^N \mathcal{L}(\theta; \mathbf{x}_j), \quad \mathbf{x}_j \sim f(\mathbf{x}), \end{aligned} \tag{15}$$

where $\mathbb{E}_f[\mathcal{L}(\theta)]$ is the expected value of the total loss \mathcal{L} , when the collocation points are sampled from f the sampling distribution in the physical domain $\mathbf{x}_j \in \Omega$.

Typically, we use a uniform distribution for sampling the collocation points. In importance sampling, the collocation points are drawn from an alternative sampling distribution $q(x)$, and the NN parameters are approximated as per Eq. 16 instead of Eqs. 5 and 6.

$$\theta^* \approx \operatorname{argmin}_{\theta} \frac{1}{N} \sum_{j=1}^N \frac{f(\mathbf{x}_j)}{q(\mathbf{x}_j)} \mathcal{L}(\theta; \mathbf{x}_j), \quad \mathbf{x}_j \sim q(\mathbf{x}). \tag{16}$$

Sampling the collocation points in each epoch according to $q(x)$ (Eq. 17), i.e. a distribution proportional to the loss function \mathcal{L} improves the efficiency of PINNs without introducing a hyperparameter.

$$q_j^{(i)} \approx \frac{\mathcal{L}_j^{(i)}}{\sum_{j=1}^N \mathcal{L}_j^{(i)}}, \quad \forall j \in 1, \dots, N, \tag{17}$$

where $\mathcal{L}_j^{(i)}$ is the total loss of the j th sample in the training dataset in i th epoch. Areas with higher $q^{(i)}$ are sampled more frequently in the i th epoch.

2.3.4 Low-Discrepancy Spatio-temporal Sampling

The collocation points may be sampled according to a uniform distribution, or using the Latin hypercube sampling [83] approach. Alternatively, one can choose low-discrepancy sequence generators such as the quasi-random sampling [84, 85], the Halton sequence [86, 87] the Sobol sequence [88, 89] which is used by DeepXDE's baseline PINN and Hammersley sets [90, 91].

3 Modified Baseline PINNs

Numerous modified frameworks of the baseline PINN have been proposed so far. We have specifically chosen PINN frameworks that not only scale to higher dimensions but can also deal with discontinuous BCs.

3.1 DeepXDE

DeepXDE [37] is a popular Python based physics-informed machine learning library for solving forward and inverse problems involving PDEs. DeepXDE features its own version of the baseline PINN, that not only improves the accuracy, but helps in faster convergence as discussed in Sect. 2.3.2.

Other than baseline PINNs, DeepXDE can also solve forward and inverse integro-differential equations, (IDEs) [92], fractional PDEs, (fPDEs) [92], stochastic PDEs, (sPDEs) [93], topology optimization with hard constraints, (hPINN) [7], PINN with multi-scale Fourier features [94] and multifidelity NN, (MFNN) [95, 96].

DeepXDE also features NNs for nonlinear operator learning such as DeepONet [97], POD-DeepONet [98], MIONet [99], DeepM&Mnet [100, 101] and multifidelity DeepONet [102].

It supports tensor libraries such as TensorFlow, PyTorch, JAX, and PaddlePaddle. There are two notable features in DeepXDE. The first one is that it chooses the model with least training loss. The second one is that it does not include those corner points no matter what the problem is, with the justification that normal vectors are not defined at those corners to be able to apply the Neumann BCs.

3.2 NVIDIA Modulus

NVIDIA Modulus 22.03, formerly NVIDIA SimNet, is an advanced physics-informed machine learning package. It redefines the loss function (Sect. 2.2.3), it also uses SDF weights to avoid problematic edges and corners (Sect. 2.3.2). The SDF weights result in increased convergence speed and improved accuracy.

NVIDIA Modulus employs the SDF weights on the collocation points (they refer to these as interior points) and the boundary points separately. We have discussed the variation of SDF weights in the spatial domain in Sects. 4 and 5.

NVIDIA Modulus features forward models such as Fourier networks [25], Modified Fourier networks [26, 27], Sinusoidal Representation networks, SiReNs [28], Highway Fourier network [103], multi-scale Fourier feature

network [94], spatial–temporal Fourier Feature network [97], DGM architecture [22] and multiplicative filter network [27].

NVIDIA Modulus also features NNs for nonlinear operator learning such as Fourier neural operator, (FNO) [103], adaptive FNO, (AFNO) [104], physics-informed neural operator, (PINO) [105], DeepONet [97], Pix2Pix net [106, 107] and super-resolution net [108].

3.2.1 Fourier Network

Spectral bias is a learning bias of DNNs towards low-frequency functions, i.e., functions that vary globally rather than locally. It ignores the high-frequency functions, such as sharp variation in the solution of the PDE. This can adversely affect the training convergence as well as the accuracy.

One approach to alleviate this issue is to perform input encoding, for example, a transformer [109], to transform the inputs to a higher-dimensional feature space with the help of high-frequency functions. In the Fourier network, the input features are encoded into Fourier space using sinusoidal activation functions (Eq. 18).

The trainable input encoding layer is as follows:

$$\phi_E = [\sin(2\pi\mathbf{f} \times \mathbf{X}); \cos(2\pi\mathbf{f} \times \mathbf{X})]^T, \quad (18)$$

where $\mathbf{f} \in \mathbb{R}^{n_f \times d_0}$ is the trainable frequency matrix, d_0 is the number of features and n_f is the number of frequency sets which we can choose and \mathbf{X} is the input dataset. These frequencies, similar to network weights, can be sampled from a Gaussian distribution or from a spectral space created from combinations of all entries from a user-defined list of frequencies.

Finally, the encoded inputs are $\phi_E x$, which results in a training data with n_f number of features, thus, transforming the input features to a higher dimensional feature space. NVIDIA Modulus recommends $n_f = 10$, however, for our test cases we used $n_f = 35$, which is computationally expensive, but is reasonable for PDEs with a discontinuous solution.

3.2.2 Modified Fourier Network

In a Fourier network (Sect. 3.2.1), a FCNN is used as the nonlinear mapping between the Fourier features and the model output. The modified Fourier network uses a modified version of the fully-connected network, similar to the one proposed in [18]. The authors were inspired by the neural attention mechanism, which is employed in natural language processing to enhance the hidden states with transformer networks [109].

In Eq. 19, \mathbf{U} and \mathbf{V} are two transformer layers that help in projecting the Fourier features ϕ_E to another feature space, and forward passed through the hidden layers (\mathbf{H}) using the Hadamard product, similar to its standard fully connected counterpart. The Modified Fourier network takes the following forward propagation rule:

$$\begin{aligned} \mathbf{U} &= \sigma(\mathbf{W}^1 \phi_E), \quad \mathbf{V} = \sigma(\mathbf{W}^2 \phi_E), \\ \mathbf{H}^{(1)} &= \sigma(\mathbf{W}^{z,1} \mathbf{X}), \\ \mathbf{Z}^{(k)} &= \sigma(\mathbf{W}^{z,k} \mathbf{H}^{z,1}), \quad k = 1, \dots, L, \\ \mathbf{H}^{(k+1)} &= \sigma(1 - \mathbf{Z}^{(k)}) \odot \mathbf{U} + \mathbf{Z}^{(k)} \odot \mathbf{V}, \quad k = 1, \dots, L, \\ \hat{y} &= \mathbf{W} \mathbf{H}^{(L+1)}, \end{aligned} \tag{19}$$

where \mathbf{W}^1 , \mathbf{W}^2 , $\mathbf{W}^{z,k}$ and \mathbf{W} are four different sets of weight matrices associated with \mathbf{U} , \mathbf{V} , $\mathbf{H}^{(k)}$ and $\mathbf{H}^{(L+1)}$ with $k = 1, \dots, L$, where L is the number of hidden layers. Figure 4 shows the structure of a modified Fourier network.

3.2.3 Sinusoidal Representation Networks (SiReNs)

Sitzmann et al. [28] proposed a FCNN which uses the sine trigonometric function as the activation function. This network has some similarities to Fourier networks (Sect. 3.2.1) as both uses a sine activation function, manifesting the same effect as the input encoding for the first layer of the network.

A key component of this network architecture is the weight initialisation scheme. The weights of the NN are sampled from a uniform distribution $W \sim U\left(-\sqrt{\frac{6}{f_{in}}}, \sqrt{\frac{6}{f_{in}}}\right)$ where f_{in} is the input size to that layer.

The input of each Sine activation has a Gauss-Normal distribution and the output of each Sine activation, a Sine inverse distribution. This preserves the distribution of activations allowing deep architectures to be constructed and trained effectively.

3.2.4 DGM Architecture

The DGM architecture [22] (Eq. 20), consists of several hidden layers, which are referred to as DGM layers, similar to LSTM gates [23], where each layer produces weights based on the last layer, determining how much of the information gets passed to the next layer.

$$\begin{aligned} \mathbf{S}^{(1)} &= \sigma(\mathbf{W}^1 \mathbf{X}), \\ \mathbf{Z}^{(k)} &= \sigma(\mathbf{V}_z^{(k)} \mathbf{X} + \mathbf{W}_z^{(k)} \mathbf{S}^{(k)}), \quad k = 1, \dots, L, \\ \mathbf{G}^{(k)} &= \sigma(\mathbf{V}_g^{(k)} \mathbf{X} + \mathbf{W}_g^{(k)} \mathbf{S}^{(k)}), \quad k = 1, \dots, L, \\ \mathbf{R}^{(k)} &= \sigma(\mathbf{V}_r^{(k)} \mathbf{X} + \mathbf{W}_r^{(k)} \mathbf{S}^{(k)}), \quad k = 1, \dots, L, \\ \mathbf{H}^{(k)} &= \sigma(\mathbf{V}_h^{(k)} \mathbf{X} + \mathbf{W}_h^{(k)} (\mathbf{S}^{(k)} \odot \mathbf{R}^{(k)})), \quad k = 1, \dots, L, \\ \mathbf{S}^{(k+1)} &= (1 - \mathbf{G}^{(k)}) \odot \mathbf{H}^{(k)} + \mathbf{Z}^{(k)} \odot \mathbf{S}^{(k)}, \quad k = 1, \dots, L, \\ \hat{y} &= \mathbf{W} \mathbf{S}^{(L+1)}. \end{aligned} \tag{20}$$

The DGM architecture consists of multiple nonlinear transformations of the input: \mathbf{Z} , \mathbf{G} , \mathbf{R} and \mathbf{H} , that helps with learning complicated functions such as discontinuous functions. Figure 5 shows the structure of DGM architecture and the DGM layers.

A DGM layer includes \mathbf{Z} , \mathbf{G} , \mathbf{R} and \mathbf{H} with their sets of weights \mathbf{V} and \mathbf{W} . Thus, a DGM layer consists of eight weight matrices. Additionally, the DGM architecture consists of two more weight matrices: \mathbf{W}^1 and \mathbf{W} .

4 Problem 1: 2D Steady-State Heat Conduction

For Problem 1, we chose a 2D heat conduction problem with conflicting BCs at the corners. We solved the problem with different PINN frameworks and compared with the FEM solution. The 2D heat conduction problem can be stated as follows:

$$\begin{aligned} u_{xx} + u_{yy} &= 0, \quad x \in [-0.5, 0.5], \quad y \in [0, 1], \\ u(x, 0) &= u(x, 1) = 0, \quad u(-0.5, y) = u(0.5, y) = 1. \end{aligned} \tag{21}$$

We used MATLAB Partial Differential Equation Toolbox [110] to solve the 2D steady-state heat conduction problem (Eq. 21) with quadratic triangles (Fig. 6).

We assigned a model number to each of the PINN frameworks (see Table 1). Table 2 summarises the network parameters and relative L^2 error for various PINN frameworks. Figures 9 and 7 show the solution of 2D steady-state heat conduction problem for different PINN frameworks.

We have not completed a full exploration of the best parameters for each PINN such as the number of hidden

Fig. 4 Structure of modified Fourier network as per Eq. 19

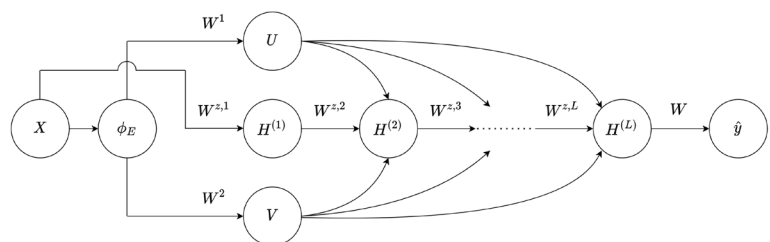
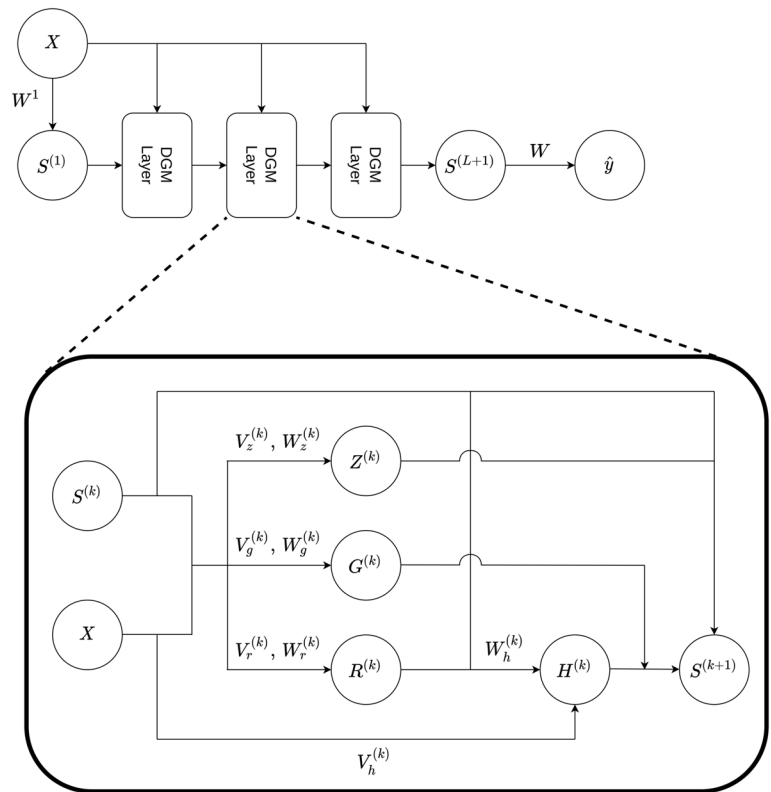


Fig. 5 Structure of structure of DGM architecture and the DGM layers as per Eq. 20



layers, the number of neurons in each layers, number of boundary and collocation points to be sampled etc. to use in these models. We started with the default values and retained the results if they were acceptable. We observed early convergence in several models, so we stopped training those models. Unless otherwise specified, the same applies to other upcoming problems. As the PINN reaches maturity we will hopefully come up with the best practices to benchmark different PINN frameworks.

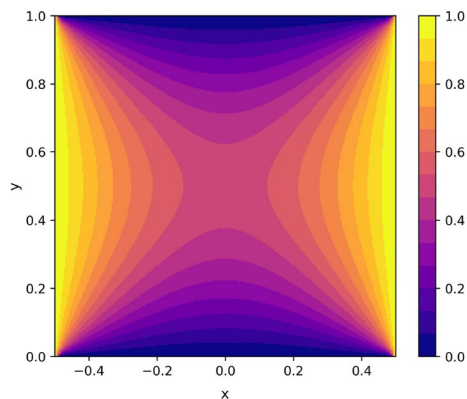


Fig. 6 FEM solution of 2D steady-state heat conduction problem (Eq. 21) using MATLAB Partial Differential Equation Toolbox

4.1 Model 1: Baseline PINN (See Also Table 1)

Firstly, we trained a baseline PINN with 8 hidden layers and 20 neurons in each layer using the L-BFGS optimiser with hyperbolic tangent activation. We used the nodal coordinates from the mesh of FEM solution consisting of 612 boundary points and 5800 collocation points. We observed that the gradient explodes while training with L-BFGS. That is why we switched to the Adam optimiser with default parameters, unless otherwise stated.

Figure 7 shows the predicted temperature distribution and pointwise absolute error at 5k, 10k and 30k epochs. Initially, i.e., around 5k epochs, the pointwise absolute error around the corners is much higher compared to the interior points. If we continue the training past 5k epochs, the optimiser tries to reduce the loss around the corners at the cost of high errors at the interior points, which can be clearly seen in the pointwise absolute error plot at 30k epochs. This is a result of the fact that we did not use a learning rate scheduler [111]. A learning rate scheduler adjusts the learning rate α between epochs during the training which helps convergence.

Figure 8 shows the training and validation losses. We purposely used the predicted temperature distribution from the training dataset against the FEM solution to calculate the validation loss. In the validation loss plot, we can clearly see that the least validation error occurs somewhere between 15k and 20k. However, the total training loss continues to

Table 1 Models considered in the numerical study

Model number	Model name
Model 1	Baseline PINN
Model 2	DeepXDE's baseline PINN
Model 3	NVIDIA Modulus's baseline PINN (no SDF weights)
Model 4	NVIDIA Modulus's baseline PINN (Interior SDF weights)
Model 5	NVIDIA Modulus's baseline PINN (Full SDF weights)
Model 6	NVIDIA Modulus's baseline PINN (Full SDF weights with importance sampling)
Model 7	NVIDIA Modulus's baseline PINN (Full SDF weights with adaptive activation, importance sampling and quasi-random sampling*)
Model 8	Fourier network (Full SDF weights with adaptive activation, importance sampling and quasi-random sampling*)
Model 9	SiReNs network (Full SDF weights with importance sampling and quasi-random sampling*)
Model 10	Modified Fourier network (Full SDF weights with adaptive activation, importance sampling and quasi-random sampling*)
Model 11	DGM architecture (Full SDF weights with adaptive activation, importance sampling and quasi-random sampling*)

*Sampling drawn from a Halton sequence

Table 2 Problem 1: summary of training parameters and relative L^2 error for different PINN frameworks

	Layers	Nodes	Boundary points	Collocation points	α	Epochs	Relative L^2 error
Model 1	8	20	612	5800*	1e-3	30k	0.13599
Model 2	8	20	800	2500**	1e-3	30k	0.05285#
Model 3	6	512	4000	4000	1e-3	20k	0.08931
Model 4	6	512	4000	4000	1e-3	10k	0.10124
Model 5	6	512	4000	4000	1e-3	10k	0.09557
Model 6	6	512	4000	4000	1e-3	20k	0.06448
Model 7	6	512	4000	4000	1e-3	20k	0.06433
Model 8	6	512	4000	4000	1e-3	20k	0.18741
Model 9	6	512	4000	4000	2e-5	11k	0.12640
Model 10	6	512	4000	4000	1e-3	20k	0.17577
Model 11	6	512	4000	4000	1e-3	30k	0.08229

*Model 1 uses the nodal coordinates from the FEM mesh

**Model 2 uses the Sobol sequence

#Relative L^2 error not calculated at corner points

decrease even after 20k epochs resulting in increased validation loss in the interior points as discussed earlier.

The FCNNs are continuous and differentiable functions and can't predict discontinuities such as the corners with conflicting BCs. Also, it is very challenging to obtain a trained PINN model with minimum validation error because we are not using the ground truth to calculate the training loss.

4.2 Model 2: DeepXDE's Baseline PINN

We trained a FCNN with 8 hidden layers and 20 neurons in each layer using the Adam optimiser with hyperbolic tangent activation and exponential decay of the learning rate. We used DeepXDE's default sampling method, i.e., the Sobol sequence to sample 800 boundary points and

2500 collocation points, i.e. only about half the collocation points compared to Model 1.

Although the training loss converged in 5k epochs, we continued to train the model until 30k epochs to demonstrate the benefits of exponential decay of the learning rate. As the training progresses, the amount of perturbation in the weights (as per Eq. 6) decreases. Thus, we don't see much difference in the predicted temperature distribution after 5k epochs, which was not possible with Model 1.

As discussed in Sect. 2.3.2, DeepXDE does not sample the corner points, so we ignored these points from the FEM solution to compute the relative L^2 error. Thus we obtain very low relative L^2 error as most of the error occurs around the corners.

Fig. 7 Solution of 2D steady-state heat conduction problem using Model 1. The predicted temperature distribution (on the top) and the absolute pointwise error (on the bottom) is shown at 5k, 10k and 30k epochs respectively. Temperature predicted outside the expected bound, i.e., when $u \notin [0, 1]$, is shown using the white and grey colour

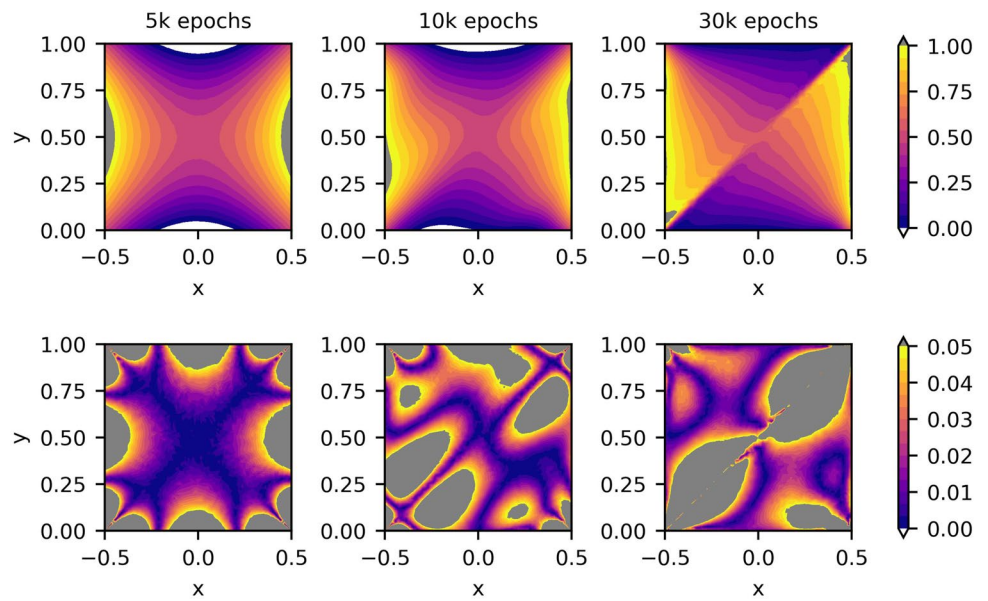


Figure 9a shows the predicted temperature distribution and the pointwise absolute error for the 2D steady-state heat conduction problem.

4.3 Models 3–11: NVIDIA Modulus

We used NVIDIA Modulus to train PINN frameworks from Models 3 to 11. We started with NVIDIA Modulus’s baseline PINN with the integral form of the loss function (Model 3). Then we applied the SDF weights on interior points (Model 4) and on both interior and boundary points (Model 5). We used additional tools such as importance sampling (Model 6) and combined the importance sampling with adaptive activation and quasi-random sampling (Model 7). We also used the Fourier network (Model 8), SiReNs network (Model 9), Modified-Fourier network (Model 10) and DGM architectures (Model 11) with adaptive activation, importance sampling, quasi-random sampling and full SDF

weights, i.e., SDF weights on both interior and boundary points with an exception for SiReNs network which does not has an adaptive activation implemented in NVIDIA Modulus 22.03. Furthermore, NVIDIA Modulus 22.03 only implements the Halton sequence to generate a quasi-random sample.

The SDF weights are manually adjusted depending on the problem and is an active field of research. We formulated the SDF weights on boundary points such that the weights on the corner points is zero and increased as we move away from the corner (see Eq. 22). The SDF weights on the interior points depends on the shape of the spatial domain. We used the default SDF weights on collocation/interior points. Figure 10 shows the magnitude of the SDF weights on the boundary and interior points of the square domain for Problem 1. In the case of a full set of SDF weights, it is worth noting that the maximum magnitude of the SDF weights on the interior points is 0.5. So, We are giving 50% less

Fig. 8 The training loss (top) and the validation loss (bottom) for the 2D steady-state heat conduction problem. The training loss plot shows the BC loss (BC loss), the residual loss (PDE loss) and the sum of both, i.e., the total loss. The validation loss plot shows the mean squared error between the temperature predicted from the training dataset and the FEM solution. The black dot denotes the least validation loss

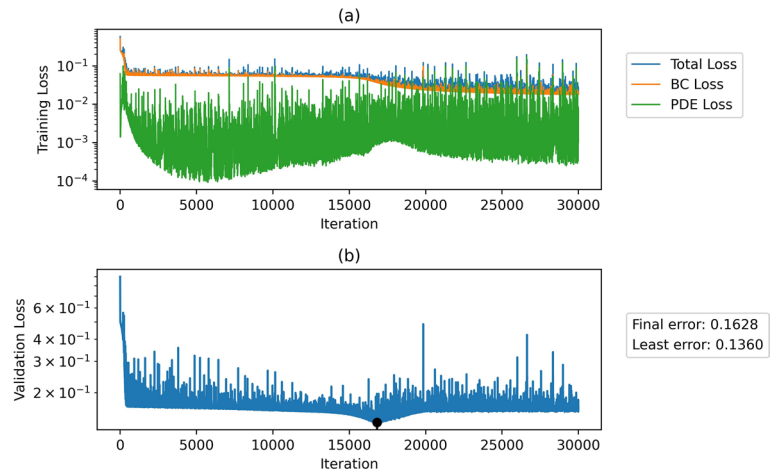


Fig. 9 PINN predicted solution of 2D steady-state heat conduction problem is shown on the left side. The absolute pointwise error between the FEM solution and PINN predicted solution is shown on the right side. Temperature predicted outside the expected bound (if any), i.e., when $u \notin [0, 1]$ is shown using the white and grey colour

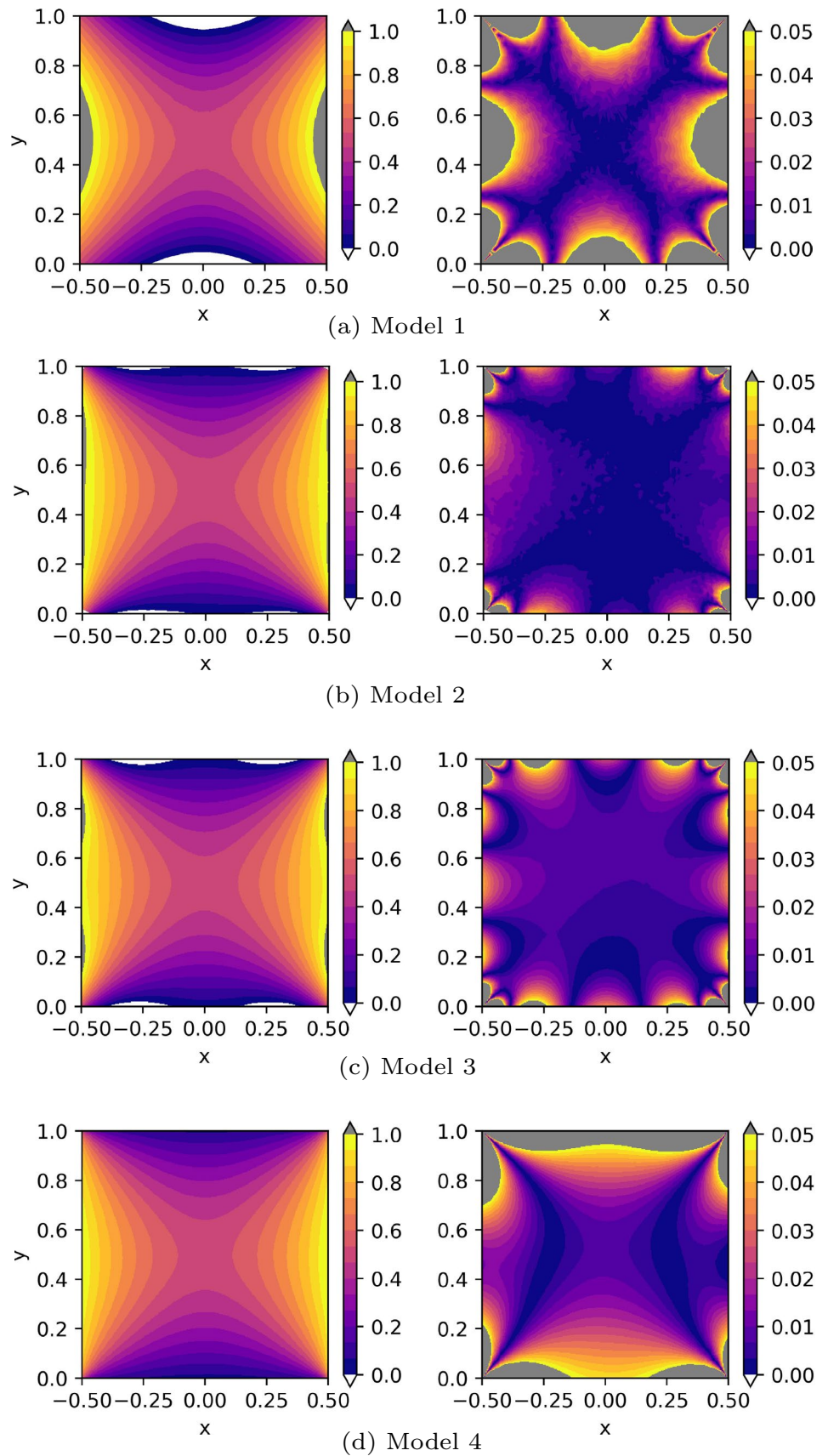


Fig. 9 (continued)

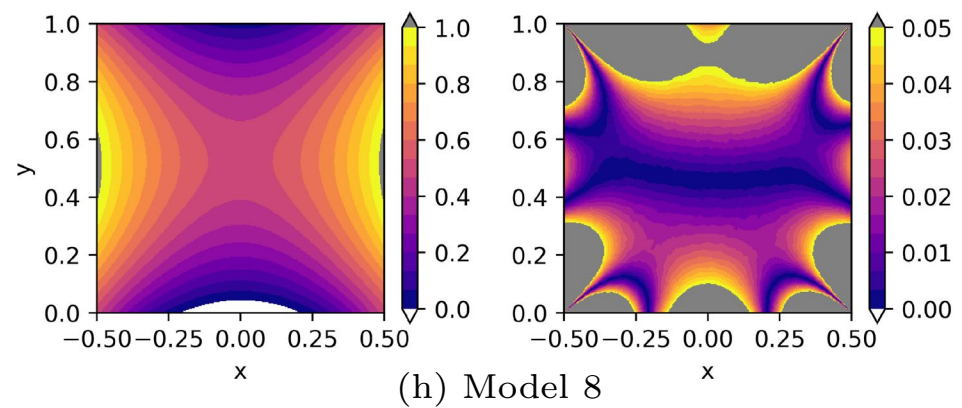
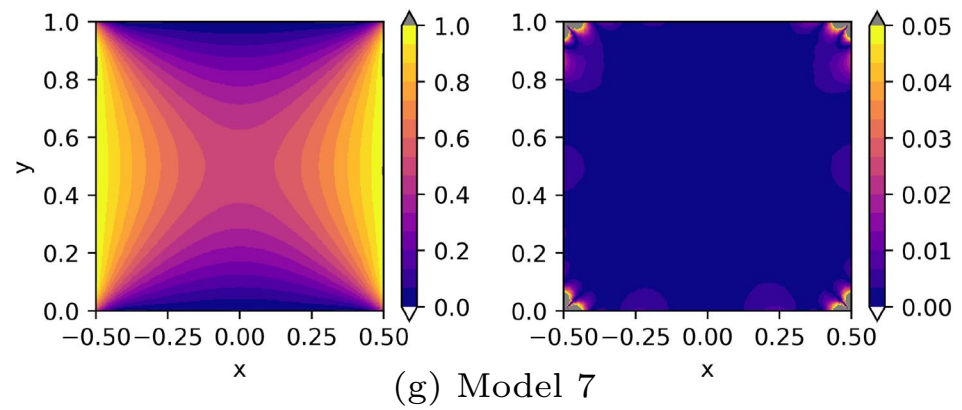
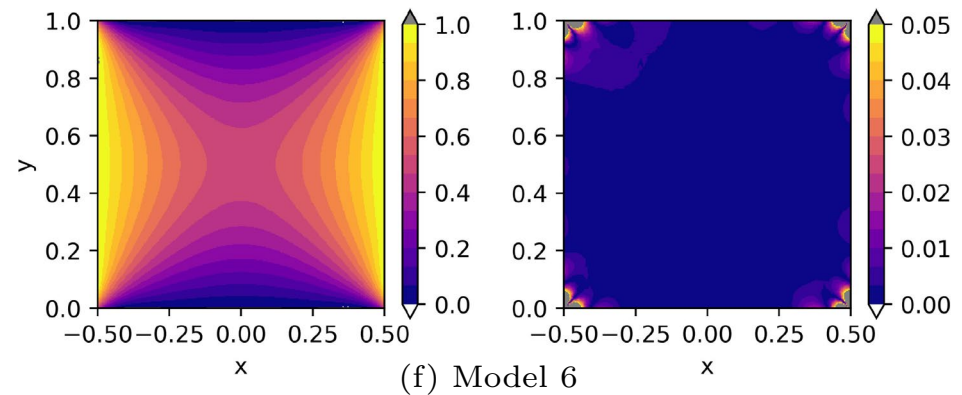
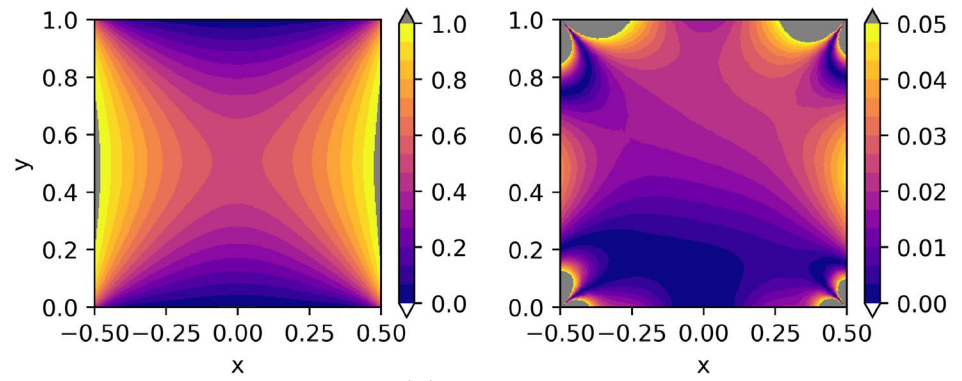
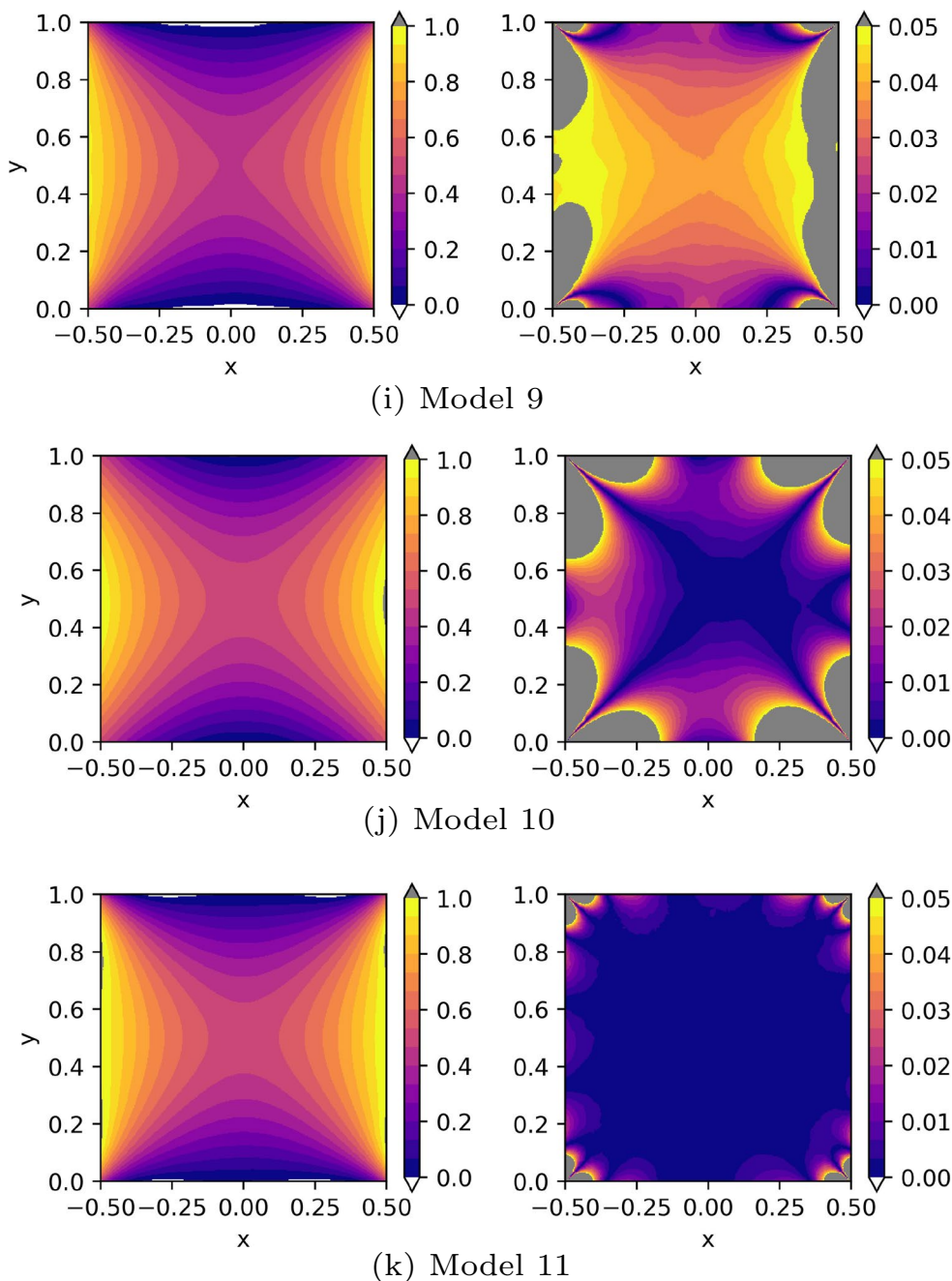


Fig. 9 (continued)



importance to the interior points compared to the boundary points, because we want the optimiser to obtain more information from the BC loss, as it does not converge in the case of Model 1 (as shown in Fig. 8).

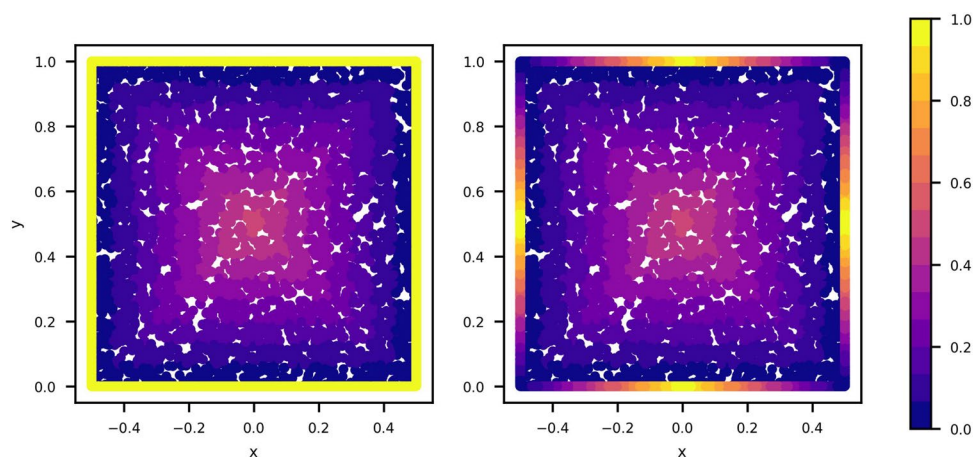
$$\begin{aligned} \text{Top and bottom boundary: } & y = 1.0 - 2|x|, \\ \text{Left and right boundary: } & x = 1.0 - 2|y - 0.5|. \end{aligned} \tag{22}$$

In Models 3, 4 and 5, we trained NVIDIA Modulus’s baseline PINN with default values, i.e., with 6 hidden

layers and 512 neurons in each layer using the Adam optimiser with exponential decay in the learning rate and SiLU activation (Sigmoid Linear Unit) [112] for 20k epochs. We used 4000 boundary points and 4000 collocation points sampled from a uniform distribution. We used the same number of boundary points and collocation points in models from Models 3 to 11. We obtained a relative L^2 error of 0.08931, 0.10124, 0.09557 for Models 3, 4 and 5, respectively.

In Model 6, we replaced the uniform sampling with importance sampling to resample the interior points in

Fig. 10 The magnitude of SDF weights on interior and boundary points of a square domain. The SDF weights for Model 4, i.e., with only interior points is shown on the left side. Whereas, the SDF weights with interior and boundary points is shown on the right side (for Models 5 to 11)



Model 5. We observed a significant improvement around the corners, resulting in a relative L^2 error of 0.06448. In the case of Model 7, we used adaptive activation with quasi-random sampling for the initial sampling in each batch and importance sampling for resampling of the interior points. We obtained a relative L^2 error of 0.06433, not a significant improvement over Model 6.

In Model 8, we trained a Fourier network with 10 Fourier features (see Sect. 3.2.1) using 6 hidden layers and 512 neurons in each layer using the Adam optimiser with SiLU adaptive activation for 20k epochs, with full SDF weights, quasi-random sampling and importance sampling. We used 10, 15, 25 and 35 Fourier features for the input encoding. We observed that the Fourier network is only working for 10 Fourier features and the absolute pointwise error is 0.18741, which is even higher than Model 1. For 15, 25 and 35 Fourier features the predicted temperature distribution were close to 0.5 over the entire domain, i.e., the average of upper and lower bound temperature in the domain.

In Model 9, we trained a SiReNs network (Sect. 3.2.3) with 6 hidden layers and 512 neurons in each layer using the Adam optimiser with Sine activation for 20k epochs, with full SDF weights, quasi-random sampling and importance sampling. The network was continuously experiencing exploding gradients until we reduced the learning rate to $(2e-5)$. Still after 12k epochs the training loss would abruptly increase, leading to prohibitively large absolute pointwise error. Hence, we forced the training to stop around 11k epochs and the relative L^2 error was found to be 0.12640.

In Model 10, we trained a modified Fourier network (Sect. 3.2.2) with 6 hidden layers and 512 neurons in each layer using the Adam optimiser with SiLU activation for 20k epochs, with full SDF weights, quasi-random sampling and importance sampling. Similar to Model 10, we used 10 Fourier features for the input encoding. The predicted

temperature distribution looks similar to Model 8 and the relative L^2 error was found to be 0.17577.

In Model 11, we trained a DGM architecture (see Sect. 3.2.4), with 6 hidden layers and 512 neurons in each layer using the Adam optimiser with SiLU activation for 20k epochs, with full SDF weights, quasi-random sampling and importance sampling. We obtained a relative L^2 error of 0.08229.

In Fig. 9, Models 6, 7 and 11 resulted in less than 1% absolute pointwise error at most of the points in the domain. This indicates that the SDF weights and the importance sampling plays an important role in solving stiff-PDEs with a discontinuous solution. In Table 2, Models 3, 5, 6, 7 and 11 resulted in less than 10% relative L^2 error. Further investigation is required to determine whether the DGM architecture is advantageous compared to the baseline PINNs in higher dimensions.

4.4 Hard Constrained BCs

In this section, we discuss the exact imposition of BCs in PINNs. Hard constrained BCs involves the construction of a continuous and differentiable function through which we pass the output of the NN. Problem 1, involves discontinuous BCs which can't be exactly satisfied with a continuous and differentiable function. However, it is possible to satisfy the BCs on two opposite walls, here we chose to exactly satisfy the top and bottom walls using the following output transform function.

$$u := y(y - 1)u. \quad (23)$$

We trained the Model 2 (see Table 1) with 3500 collocation points and 400 boundary points with Adam for 30k epochs. Figure 11 shows the DeepXDE predicted solution to Problem 1 with the hard constrained BCs at the top and bottom walls. Due to hard constrained BCs at the top and bottom

walls, we observe significant errors around the left and right walls resulting in a relative L^2 error of 0.183119. Thus, it is not recommended to apply hard constraints to the BCs in stiff-PDEs.

4.5 Overfitted Solution

In this section, we discuss the behaviour of an overfitted PINN model while predicting the solution of the PDE at new spatio-temporal locations in the domain. We trained the Model 2 (see Table 1) with 2500 collocation points and 200 boundary points with Adam for 30k epochs. Figure 12 shows the DeepXDE predicted solution to Problem 1 from both the training dataset and on new spatio-temporal locations within the domain. Given that the trained model’s accuracy during the validation is low, we will categorise the model as an overfitted model. The validation loss can be decreased by adding more points in the training data set, for instance, NVIDIA Modulus samples different points for the training in each epoch. Intuitively, the overfitted model is computationally cheaper than a generalised model as it requires very few collocation points, which is evident with this example.

5 Problem 2: 3D Steady-State Heat Conduction

The next problem we address is a 3D steady-state heat conduction problem with conflicting BCs at the edges and the corners. We solved the problem with different PINN frameworks and compared the results to the FEM solution. The 3D heat conduction problem is described as follows:

$$\begin{aligned}
 u_{xx} + u_{yy} + u_{zz} &= 0, & x \in [-0.5, 0.5], & y \in [-0.5, 0.5], & z \in [0, 1], \\
 u(x, y, 0) = u(x, y, 1) = u(0.5, y, z) &= 0, \\
 u(-0.5, y, z) = u(x, -0.5, z) = u(x, 0.5, z) &= 1.
 \end{aligned}
 \tag{24}$$

We used MATLAB Partial Differential Equation Toolbox [110] to solve the 3D steady-state heat conduction problem (Eq. 24) with quadratic triangles (Fig. 13).

We again used the same PINN models (see Table 1) to solve the 3D steady-state heat conduction problem. We did not use the full SDF weights because the boundary walls are planes instead of lines in 2D. This is where we potentially require an algorithm, instead of manually constructing the SDF weights for the boundary points. Therefore, for this problem, we refer to the SDF weights on interior points as the full SDF weights and we exclude Model 4. Table 3

Fig. 11 DeepXDE predicted solution of 2D steady-state heat conduction problem (Eq. 21) with hard constrained BCs at the top and bottom walls on the left and the absolute point-wise error between the FEM and PINN predicted solutions is shown on the right side. Temperature predicted outside the expected bound (if any), i.e., when $u \notin [0, 1]$ is shown using the white and grey colour

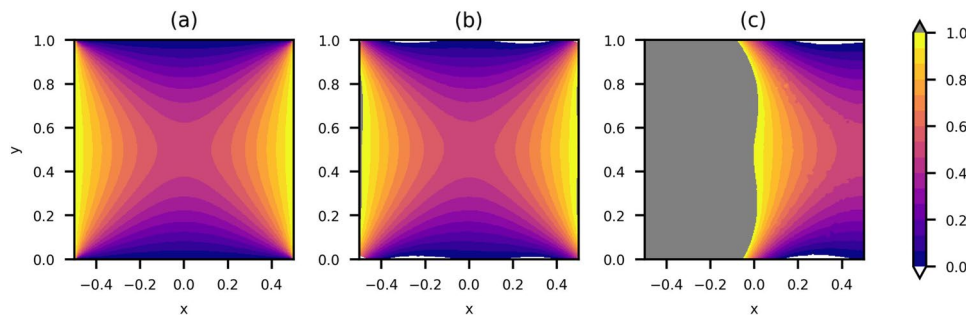
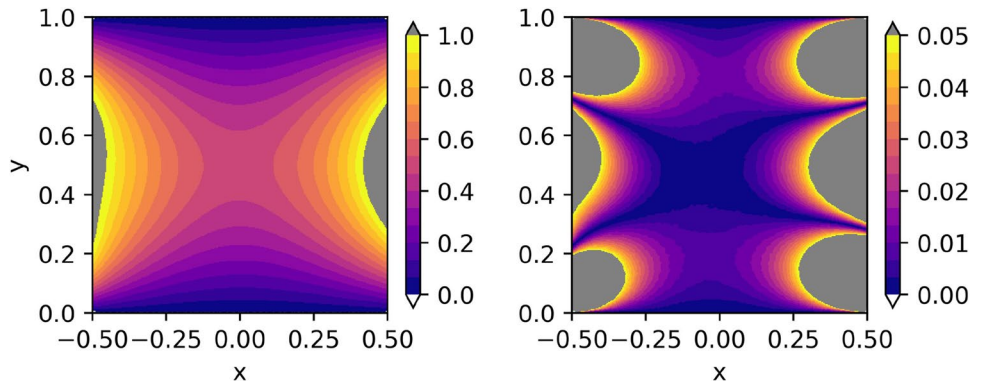


Fig. 12 Comparison of PINN predicted solution from the training dataset and at new data-points. **a** The FEM solution of Problem 1 (Eq. 21), **b** the DeepXDE predicted solution of the same problem from the training dataset, and **c** the DeepXDE predicted solution of

the same problem at new spatio-temporal locations in the domain. Temperature predicted outside the expected bound (if any), i.e., when $u \notin [0, 1]$ is shown using the white and grey colour

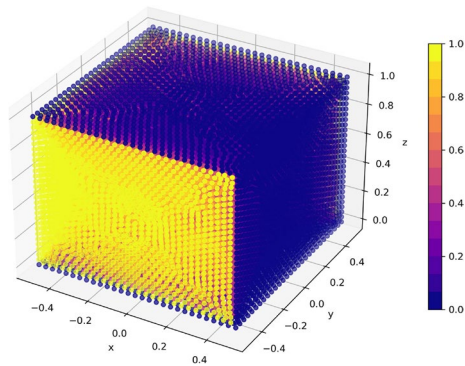


Fig. 13 FEM solution for the 3D steady-state heat conduction problem (Eq. 24) using MATLAB Partial Differential Equation Toolbox

summarises the network parameters and relative L^2 error for various PINN frameworks. Figure 14 shows the solution of the 3D steady-state heat conduction problem for different PINN frameworks.

In Problem 1 (Sect. 4), the discontinuities occurred at the corners of the square domain, which affected only a few training points. Whereas, in Problem 2, the discontinuities affected not only the vertices but also the edges. We can sample a large number of points along these edges. Thus, Problem 2 is more suitable for testing the robustness of different PINN frameworks.

In Problem 2, all the models had the same number of layers, nodes per layer and the learning rate as in Problem 1. However, we increased the number of boundary points and collocation/ interior points in Models 1 and 2.

In Model 1, we observed that the interior points had significant absolute pointwise error, meaning the PDE loss didn't converge. We obtained a relative L^2 error of 0.23778. In Model 2, the DeepXDE network didn't sample the points

at the edges and vertices (see Sect. 2.3.2). We obtained a relative L^2 error of 3.15547.

In the case of models trained within NVIDIA Modulus, the SDF weights on the interior nodes from Problem 1 was extended to three dimension such that interior points close to the boundary were assigned negligible weights. As we move away from the boundary, the SDF weights on the interior points increases until it reaches close to 0.5 around the centroid of the cubical domain (see Fig. 15).

NVIDIA Modulus's baseline PINN (Model 3) predicts better temperature distribution compared to Models 1 and 2 with a relative L^2 error of 0.12031. The addition of SDF weights on the interior points (Model 5) dramatically reduces the relative L^2 error to 0.07455.

To obtain accurate results, the nodal coordinates or the training data needs special treatment either by transforming the coordinates to higher dimensions using the Fourier network or by adding more transformer layers using the DGM architecture or both using the modified Fourier network. From Table 3, it is clear that the baseline PINN without SDF weights is not suitable for solving stiff-PDEs with discontinuities in 3D.

In summary, Models 5, 6, 7, 8, 10 and 11 resulted in less than 10% relative L^2 error and less than 5% absolute pointwise error at most of the points in the domain. It is worth mentioning that the SiReNs network (Model 9) experiences difficulty while minimising the loss, resulting in a constant temperature over the entire domain with a relative L^2 error of 0.56448 (see Fig. 14).

A summary of the total training time (s) and training time (s) per epoch in Problems 1 and 2 for each model is presented in Table 4. The training time does not include the time taken in pre-processing of the data and initialisation of the NN. In Models 1 and 2, the number of data points were different in Problems 1 and 2, which influences the training time. Thus, we could not draw a concrete conclusion.

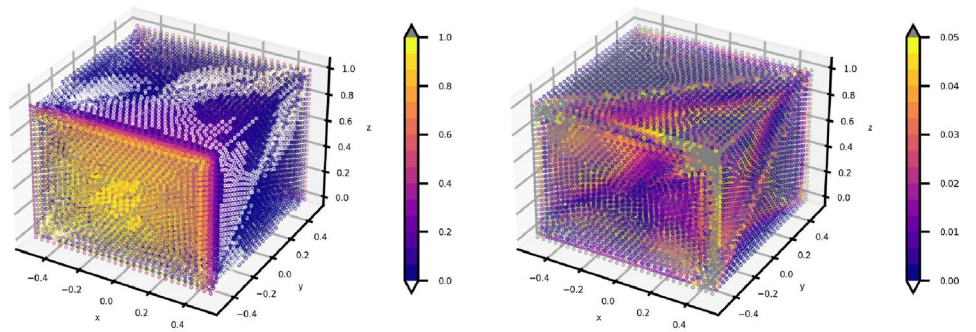
Table 3 Problem 2: summary of training parameters and relative L^2 error for different PINN frameworks

	Layers	Nodes	Boundary points	Collocation points	α	Epochs	Relative L^2 error
Model 1	8	150	5814	34,071*	1e-3	20k	0.23778
Model 2	10	150	5000	15,000**	1e-3	15k	3.15547
Model 3	6	512	4000	4000	1e-3	20k	0.12031
Model 5	6	512	4000	4000	1e-3	10k	0.07455
Model 6	6	512	4000	4000	1e-3	20k	0.07163
Model 7	6	512	4000	4000	1e-3	20k	0.08557
Model 8	6	512	4000	4000	1e-3	20k	0.07302
Model 9	6	512	4000	4000	2e-5	20k	0.56448
Model 10	6	512	4000	4000	1e-3	20k	0.09498
Model 11	6	512	4000	4000	1e-3	30k	0.07213

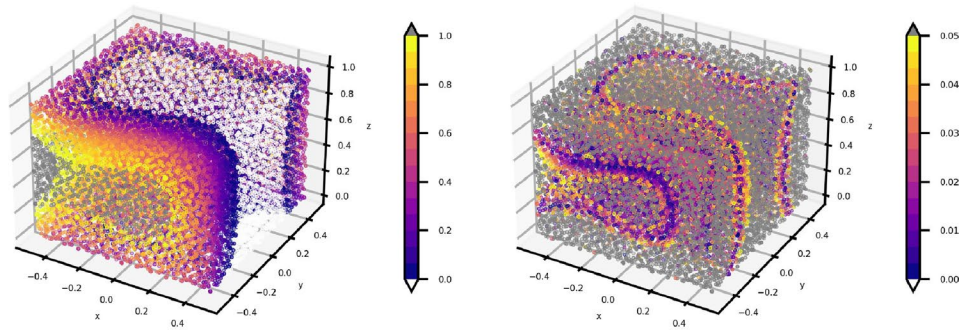
*Model 1 uses the nodal coordinates from the FEM mesh

**Model 2 uses the Sobol sequence

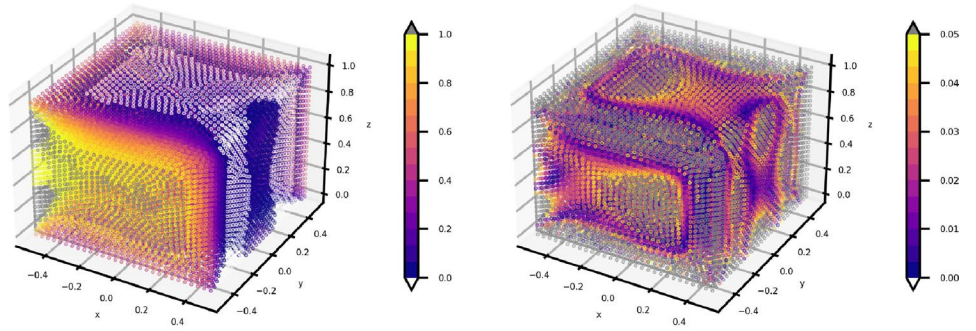
Fig. 14 PINN predicted solution of 3D steady-state heat conduction problem is shown on the left side. The absolute pointwise error between the FEM solution and PINN predicted solution is shown on the right side. Temperature predicted outside the expected bound (if any), i.e., when $u \notin [0, 1]$ is shown using the white and grey colour



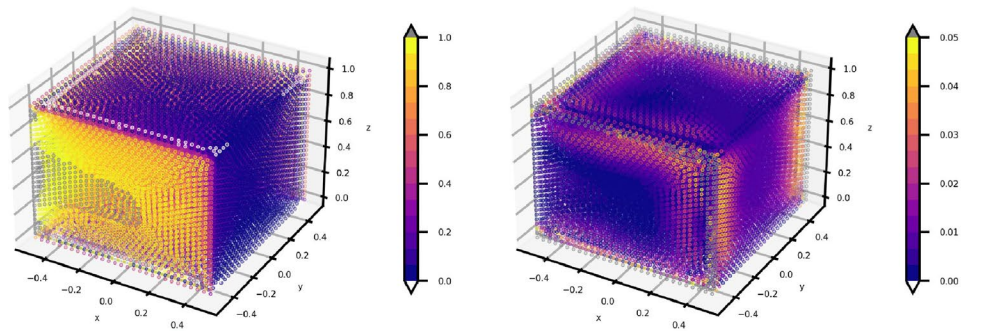
(a) Model 1



(b) Model 2

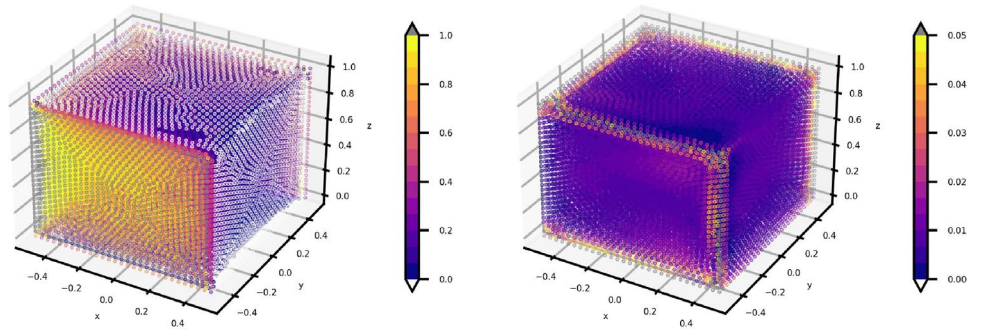


(c) Model 3

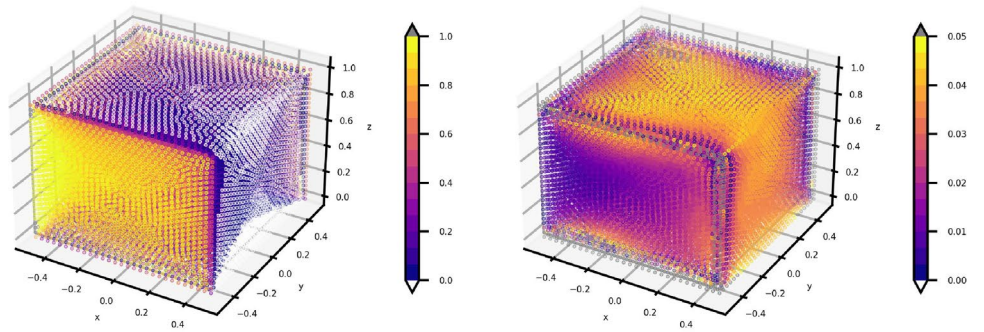


(d) Model 5

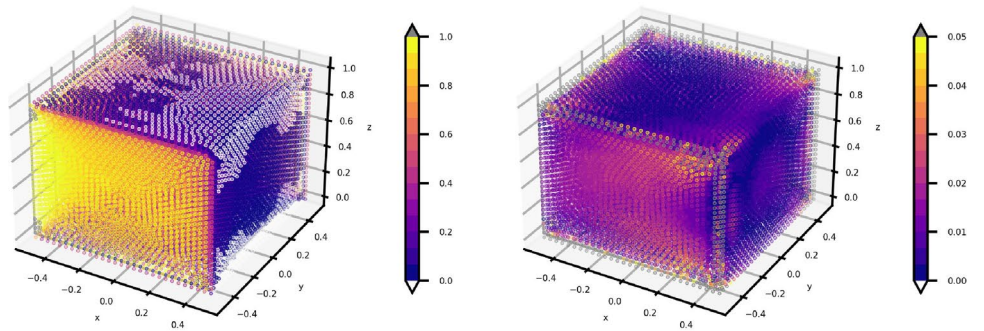
Fig. 14 (continued)



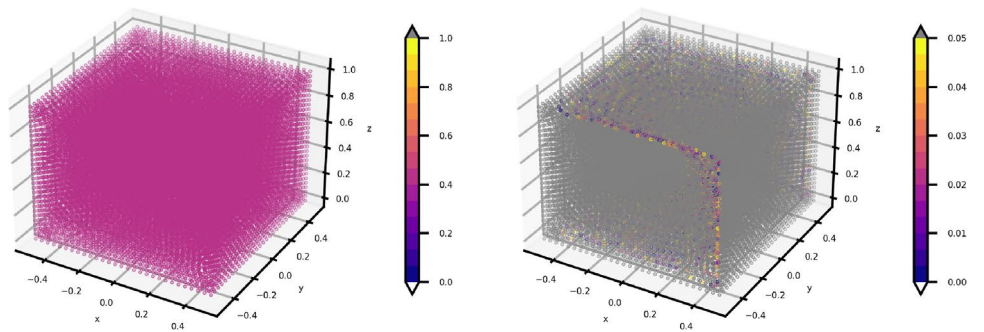
(e) Model 6



(f) Model 7



(g) Model 8



(h) Model 9

Fig. 14 (continued)

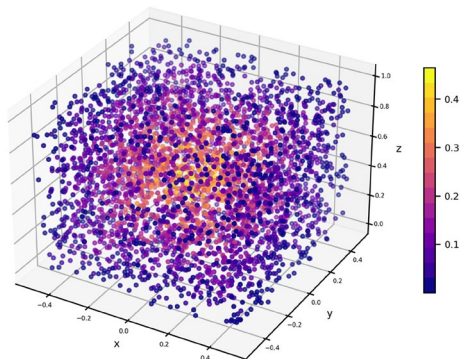
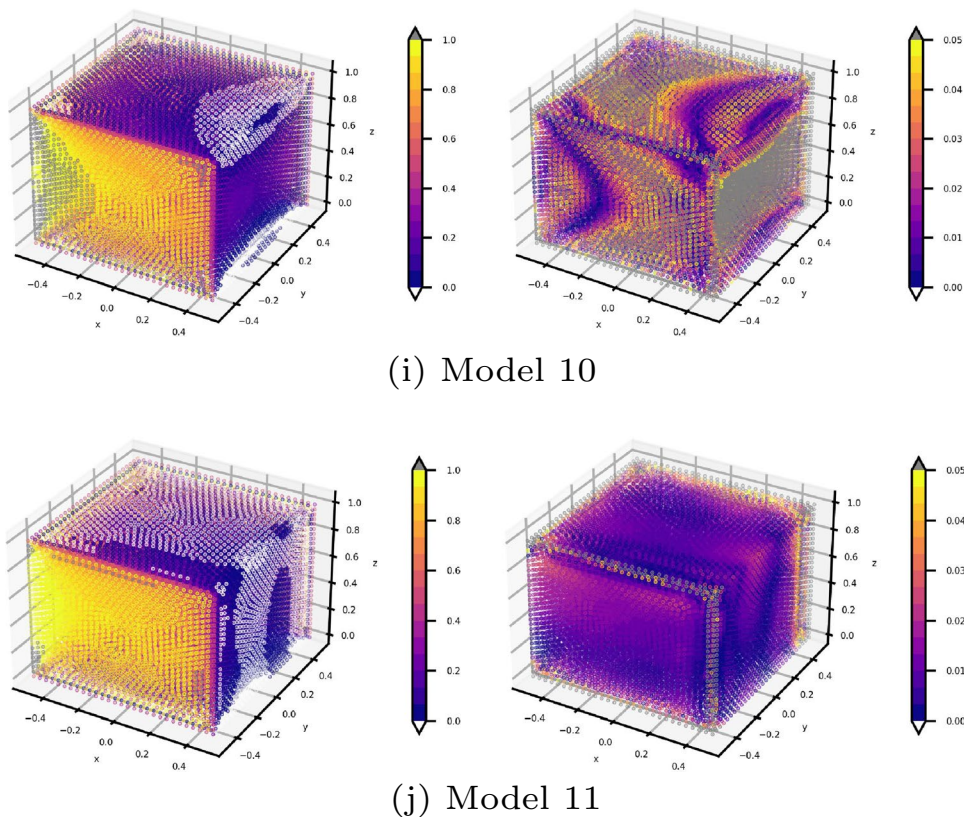


Fig. 15 The magnitude of SDF weights on interior points of the cubical domain of Problem 2

In Models 3–10, even though the number of data points were same there was no noticeable increase in the training time per epoch. However, in Model 11, the training time per epoch was almost 3 times for Problem 2, a 3D problem, compared to Problem 1, a 2D problem. This is because one DGM layers contains eight weight matrices and introducing one more feature into the training dataset increases the number of operations exponentially.

(i) Model 10

(j) Model 11

6 Parametric Heat Conduction Problem

As discussed in Sect. 2.2.6, PINNs can be parameterised by adding the parameter of interest as another feature in the training dataset. We solved two parametric 2D steady-state heat conduction problems with parameterised conductivity and parameterised geometry (see Table 5).

6.1 Problem 3: Parameterised Conductivity

We formulated the 2D steady-state heat conduction problem such that the conductivity κ is varying from 0 to 1 in Eq. 25.

$$\begin{aligned}
 u_{xx} + \kappa u_{yy} &= 0, \quad x \in [-0.5, 0.5], \quad y \in [0, 1], \quad \kappa \in [0, 1], \\
 u(x, 0) = u(x, 1) &= 0, \quad u(-0.5, y) = u(0.5, y) = 1.
 \end{aligned}
 \tag{25}$$

We trained Problem 3, with various models from Table 1 and observed that the predicted temperature distribution was fairly accurate except for Model 1 because we did not use a learning rate scheduler (see Sect. 4.1). The absolute pointwise error is less than 5% on the interior points with some errors around the corners, similar to Problems 1 and 2. Figure 16 shows the solution to Problem 3 using Model 2. Model 2 predicted a reasonably accurate temperature

Table 4 Summary of the total training time (s) and training time (s) per epoch in Problems 1 and 2 for each model

	Problem 1		Problem 2	
	Training time (s)	Training time/epoch (10^{-3} s)	Training time (s)	Training time/epoch (10^{-3} s)
Model 1	455	15.61	2259	112.95
Model 2	482	16.06	845	56.33
Model 3	632	31.60	592	29.6
Model 4	527	52.71	–	–
Model 5	530	53.00	585	58.5
Model 6	2759	137.95	1931	96.55
Model 7	1843	92.15	2193	109.65
Model 8	2031	101.55	2416	120.80
Model 9	769	69.91	1144	57.2
Model 10	2792	139.60	3481	174.05
Model 11	2670	89.00	8174	272.46

Table 5 Summary of parametric heat conduction problems

	Parametric conductivity	Parametric geometry
PINN framework	Model 2	Models 7, 10, 11
Layers	8	6
Nodes per layer	20	512
Boundary points	10,000	4000
Collocation points	25,000	4000
α	$1e-3$	$1e-3$
Epochs	30k	20k
Average relative L^2 error	0.10305	0.05460, 0.10407, 0.08192

distribution in just 523 s. Model 3 to 11 took 2–3 times the time to train the model with very little improvement.

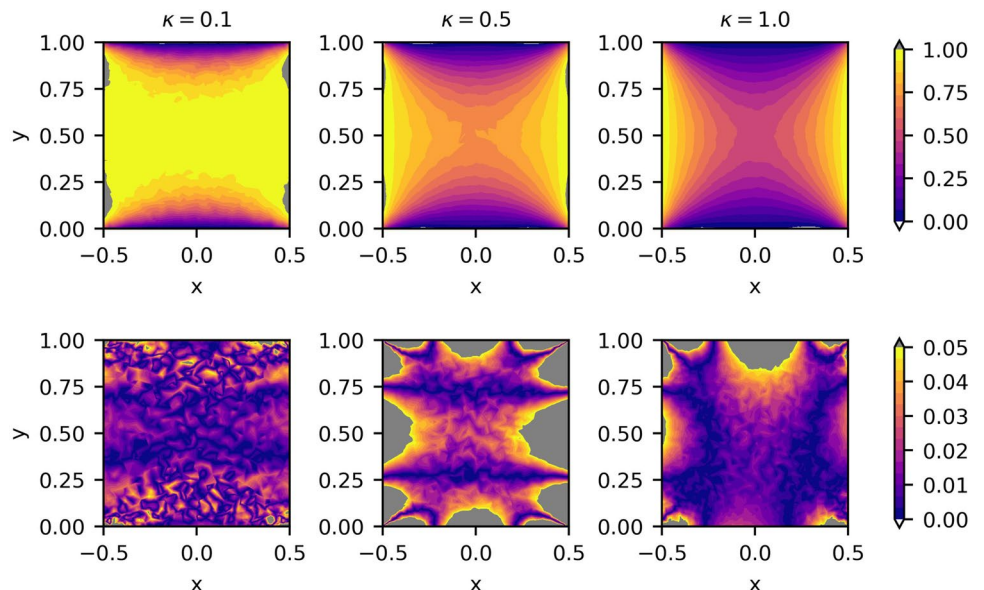
6.2 Problem 4: Parameterised Geometry

Similar to conductivity the geometry can also be parameterised. We used the y-dimension of the 2D steady-state heat conduction problem as the parameter (Eq. 26).

$$\begin{aligned}
 u_{xx} + u_{yy} &= 0, \quad x \in [-0.5, 0.5], \quad y \in [0, L], \quad L \in [1, 10], \\
 u(x, 0) = u(x, L) &= 0, \quad u(-0.5, y) = u(0.5, y) = 1.
 \end{aligned}
 \tag{26}$$

Implementing the parametric geometry is not straightforward, care should be taken that for each geometry parameter the sampled points along the geometry parameter is within the spatio-temporal domain. For example, as per Eq. 26,

Fig. 16 Solution to Problem 3 using Model 2. The predicted temperature distribution (on the top) and the absolute pointwise error (on the bottom) for different values of κ (see Eq. 25). Temperature predicted outside the expected bound, i.e., when $u \notin [0, 1]$, is shown in white and grey colour



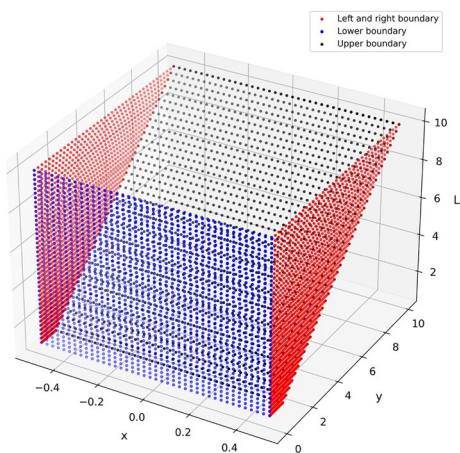


Fig. 17 The boundary points sampling of the parameterised geometry in Problem 4

$y \in [0, L]$ and $L \in [1, 10]$, i.e., $y \leq L$ for each sample in the training dataset.

Figure 17 shows the boundary points sampling of Problem 4 to illustrate the implementation of parametric geometry. Here, x and y axes shows the 2D spatial domain for each geometry parameter along the L axis. As discussed earlier, for $y \leq L$, this results in moving the upper boundary on the y axis as L increases.

We observed that none of the models could predict a temperature distribution that visually looks similar to the FEM solution. Some of the models such as Models 1, 3, 4, 5, 6 and 7 could not converge the loss. Whereas, Models 10 and 11 resulted in gradient explosion. We think that none of the PINN frameworks in Table 1 does have enough features and parameters to predict multiple discontinuous temperature distribution around the moving upper boundary. We need more sophisticated PINN frameworks to solve these types of problems.

Thus, we decreased the range of the parameter L until we could predict a temperature distribution that visually resembles the FEM solution. The parameter’s range was eventually narrowed to 1.05, i.e. $L \in [1, 1.05]$.

We trained Models 7, 10 and 11 with the redefined L parameter. For Models 10 and 11, we have also shown the predicted temperature distribution without the SDF weights to highlight how important they are while solving problems with discontinuities. Figure 18 shows the predicted temperature distribution for the redefined problem with Models 7, 10 and 11 with and without the SDF weights.

In Model 7, the predicted temperature distribution around the left and right boundary does not change with the moving upper boundary, especially, around the upper-left and upper-right corners of the square domain. In Model 10 without the SDF weights, the predicted temperature distribution does not visually resemble the FEM solution. Whereas, when we

use the SDF weights, the predicted temperature distribution improves dramatically.

In Model 11, the predicted temperature distribution without the SDF weights was much better than any other Model without the SDF weights. Also, Model 11 with SDF weights predicts better temperature distribution compared to Models 7 and 10 with SDF weights.

7 Conclusion

In this paper, we have reviewed application of PINNs to stiff-PDEs, specifically for simple steady-state heat conduction problems with discontinuous BCs at the corners. We defined a list of PINN frameworks in Table 1 which included the baseline PINN and models from open-source libraries such as DeepXDE and NVIDIA Modulus.

We started with a 2D steady-state heat conduction problem (Problem 1). We observed that both the baseline PINN and DGM architecture predicted temperature distribution with 5–10% absolute pointwise error in the corner regions for most models. The results are summarised in Table 2.

Next, we solved Problem 1 with hard constrained BCs. We showed that we can’t satisfy all the BCs exactly when they conflict with each other. Problem 1 contains discontinuous BCs, which can’t be satisfied exactly with a differentiable function such as a NN. Thus, we do not recommend applying hard constrained BCs on a stiff-PDE.

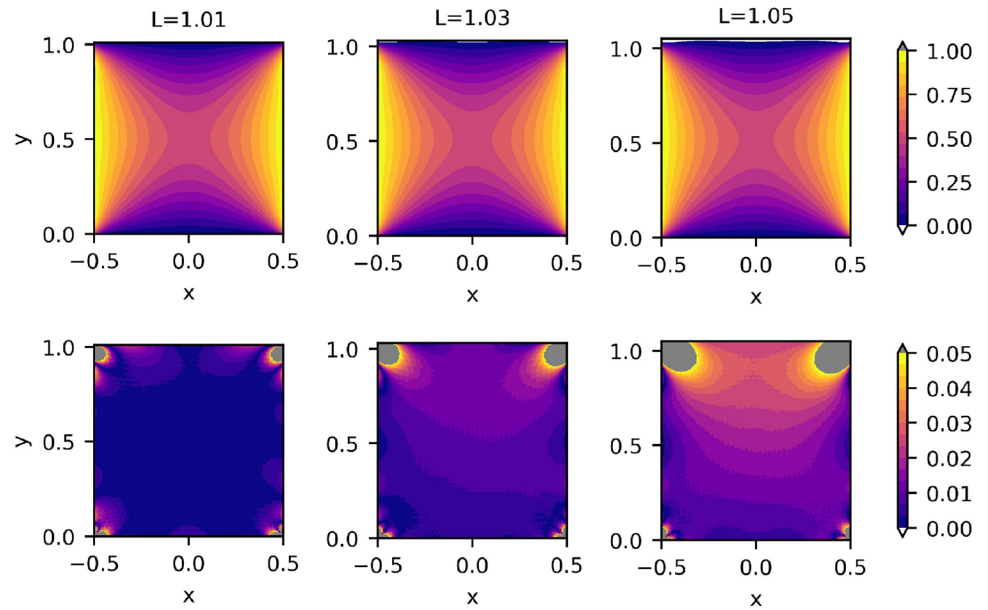
We also demonstrated the inability of an overfitted PINN to predict the temperature distribution at new spatio-temporal locations. However, an overfitted PINN proves to be computationally cheap when the solution is to be inferred on the training dataset only.

We then solved a 3D steady-state heat conduction problem (Problem 2), an equivalent of Problem 1 in 3D space. This is where modified PINN frameworks such as Fourier network, Modified Fourier network, DGM architecture proves to be more accurate than the baseline PINN frameworks (Models 1–7) in terms of the relative L^2 error. Furthermore, the SDF weights significantly improved the accuracy. This is primarily because the modified PINN frameworks are inherently evolutionary over the baseline PINN. The Fourier network addresses the spectral bias in the baseline PINNs, the DGM architecture is useful for solving problems in higher dimensions and the Modified Fourier network is a combination of both.

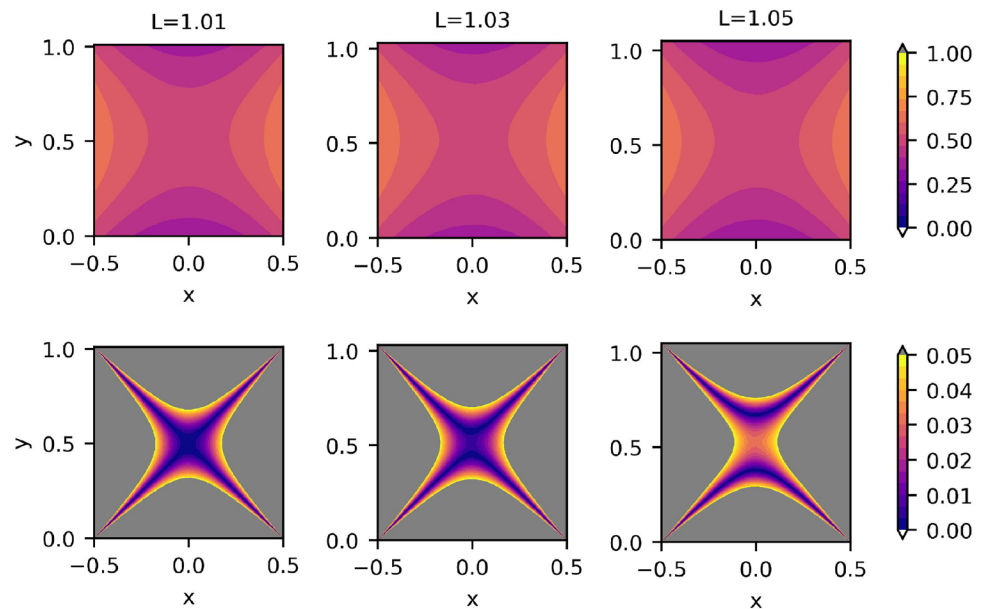
Then we solved Problem 1 with parametric conductivity (Problem 3) using DeepXDE’s baseline PINN. DeepXDE’s baseline PINN accurately predicted the temperature distribution while using fewer resources compared to Models 3–11.

We also solved Problem 1 with parametric geometry by extending the y -dimension (Problem 4). We observed that solving stiff-PDEs with parametric geometry can be very

Fig. 18 Solution of 2D steady-state heat conduction problem with parameterised upper boundary. Each plot shows the predicted temperature distribution for a specific value of parameter $L \in [1, 1.05]$. Temperature predicted outside the expected bound, i.e., when $u \notin [0, 1]$, is shown using white and grey colour



(a) Model 7



(b) Model 10 without SDF weights

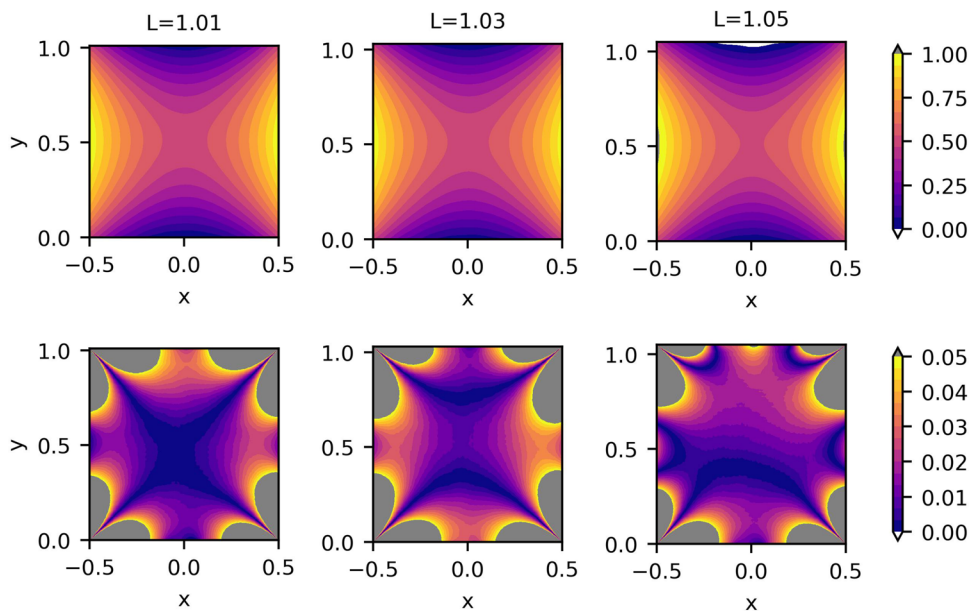
challenging for PINNs. Even robust PINN frameworks such as the Modified Fourier Network resulted in exploding gradients. In Problem 4, we also showed that the predicted temperature distribution with SDF weights significantly outperforms those without the SDF weights. Table 5 summarises the network parameters and the accuracy (relative L^2 error) of various models for Problems 3 and 4.

Among the PINN frameworks (see Sect. 1), baseline PINNs are not useful in the case of 3D problems. On the other hand, PINN frameworks such as Modified Fourier

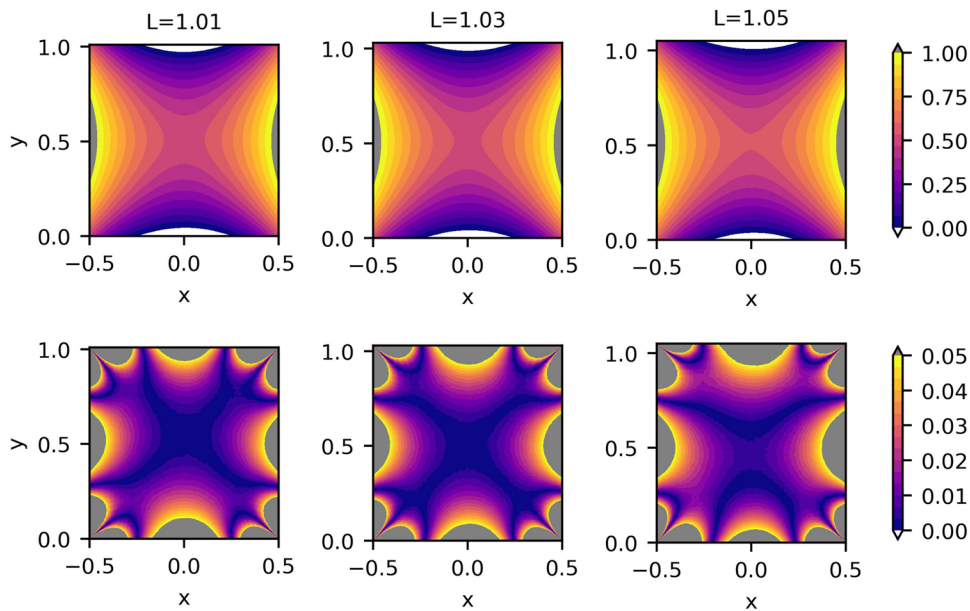
network and DGM architecture are robust even on 3D problems. The SiReNs network is very unstable and often results in exploding gradients even with exponential decay of the learning rate. The SiReNs network fails to predict the temperature distribution in Problem 2 even with a reduced learning rate. A summary of the best models for each problem is shown in Table 6.

In Table 6, Models 7 and 11 can be categorised as the best performing models in general for the four problems we considered. Model 10 also works in most of the cases. Model

Fig. 18 (continued)



(c) Model 10



(d) Model 11 without SDF weights

7 is computationally cheaper than Model 11 (see Table 4). However, Model 11 seems to work without SDF weights in Problem 4 which is an advantage for solving more complicated problems.

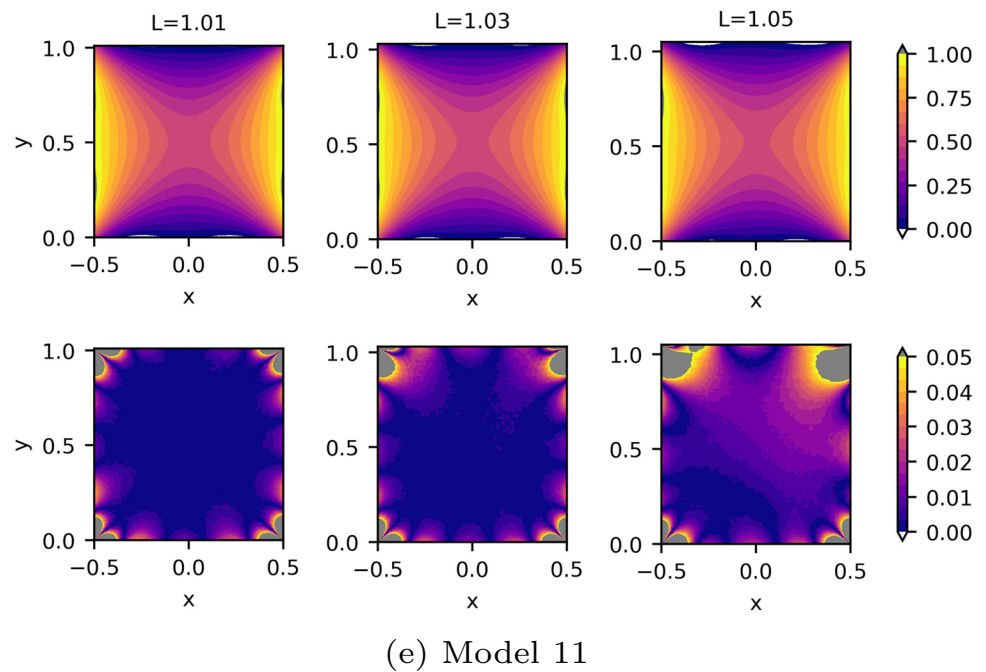
Among the additional tools (see Sect. 2.3), the SDF weights are an important asset for solving problems with discontinuous BCs. The importance sampling is a loss dependent resampling strategy of the collocation points. It samples points proportional to the absolute pointwise error, which is particularly useful in stiff-PDEs where we need more points to capture the sharp gradients. The adaptive activation and

quasi-random sampling improves the convergence and the accuracy respectively.

We also tried to solve 2D steady-state heat conduction problem with a temperature dependent conductivity along the y-direction. However, we could not converge the training loss with any of the models in Table 1. We were also interested in solving Problem 1 with parametric prescribed BCs, but it seems, currently, this is not possible with any PINN framework.

The development of FEM took us almost five decades. In contrast, PINNs are evolving rather quickly. As of now,

Fig. 18 (continued)

**Table 6** Summary of models with least relative L^2 error

	Models
Problem 1	3, 5, 6, 7 and 11
Problem 2	5, 6, 7, 8, 10 and 11
Problem 3	All except Model 1
Problem 4	7, 10 and 11

PINNs are able to solve simple discontinuous problems in 2D and 3D without any problem-specific tuning and transfer learning. The same framework can be used to solve the parametric PDEs which is difficult to achieve with conventional PDE solvers. We believe that in 2–3 years, we will see PINNs solving complex benchmark problems.

There is a need to automate the SDF for problems with discontinuous solutions. Also, there is a lack of benchmarking in PINNs. We can not use the same parameters for all the Models in Table 1. To compare different PINN frameworks, an improvement to this field would be for the research community to collectively agree upon a standard set of benchmarks.

Acknowledgements This work is funded by the United Kingdom Atomic Energy Authority (UKAEA) and the Engineering and Physical Sciences Research Council (EPSRC) under the Grant Agreement Numbers EP/T517987/1 and EP/R012091/1. We acknowledge the support of Supercomputing Wales and AccelerateAI projects, which is part-funded by the European Regional Development Fund (ERDF) via the Welsh Government for giving us access to NVIDIA A100 40 GB GPUs for batch training. We also acknowledge the support of NVIDIA for donating us a NVIDIA RTX A5000 24 GB for local testing.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest. The code we used to train and evaluate our models is available at <https://doi.org/10.5281/zenodo.7361856>.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. De Florio M, Schiassi E, Ganapol BD, Furfaro R (2022a) Physics-Informed Neural Networks for rarefied-gas dynamics: Poiseuille flow in the BGK approximation. *Z angew Math Phys* 73(3):126. ISSN 1420-9039. <https://doi.org/10.1007/s00033-022-01767-z>
2. De Florio M, Schiassi E, Furfaro R (2022b) Physics-informed neural networks and functional interpolation for stiff chemical kinetics. *Chaos Interdiscip J Nonlinear Sci* 32(6):063107. ISSN 1054-1500. <https://doi.org/10.1063/5.0086649>
3. Aliakbari M, Mahmoudi M, Vadasz P, Arzani A (2022) Predicting high-fidelity multiphysics data from low-fidelity fluid flow and transport solvers using physics-informed neural networks.

- Int J Heat Fluid Flow 96:109002. ISSN 0142-727X. <https://doi.org/10.1016/j.ijheatfluidflow.2022.109002>
4. Abueidda DW, Koric S, Guleryuz E, Sobh NA (2022) Enhanced physics-informed neural networks for hyperelasticity. Technical Report. [arXiv:2205.14148](https://arxiv.org/abs/2205.14148)
 5. Xu C, Cao TB, Yuan Y, Meschke G (2022) Transfer learning based physics-informed neural networks for solving inverse problems in tunneling. Technical Report. [arXiv:2205.07731](https://arxiv.org/abs/2205.07731)
 6. Zapf B, Haubner J, Kuchta M, Ringstad G, Eide PK, Mardal K-A (2022) Investigating molecular transport in the human brain from MRI with physics-informed neural networks. Technical Report. [arXiv:2205.02592](https://arxiv.org/abs/2205.02592)
 7. Lu L, Pestourie R, Yao W, Wang Z, Verdugo F, Johnson SG (2021a) Physics-informed neural networks with hard constraints for inverse design. *SIAM J Sci Comput* 43(6):B1105–B1132. ISSN 1064-8275. <https://doi.org/10.1137/21M1397908>
 8. Margossian CC (2019) A review of automatic differentiation and its efficient implementation. *WIREs Data Min Knowl Discov* 9(4):e1305. ISSN 1942-4795. <https://doi.org/10.1002/widm.1305>
 9. Lee H, Kang IS (1990) Neural algorithm for solving differential equations. *J Comput Phys* 91(1):110–131. ISSN 0021-9991. [https://doi.org/10.1016/0021-9991\(90\)90007-N](https://doi.org/10.1016/0021-9991(90)90007-N)
 10. Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans Neural Netw* 9(5): 987–1000. ISSN 1941-0093. <https://doi.org/10.1109/72.712178>
 11. Lagaris IE, Likas AC, Papageorgiou DG (2000) Neural-network methods for boundary value problems with irregular boundaries. *IEEE Trans Neural Netw* 11(5):1041–1049. ISSN 1941-0093. <https://doi.org/10.1109/72.870037>
 12. Malek A, Shekari Beidokhti R (2006) Numerical solution for high order differential equations using a hybrid neural network-optimization method. *Appl Math Comput* 183(1):260–271. ISSN 0096-3003. <https://doi.org/10.1016/j.amc.2006.05.068>
 13. Rudd K, Ferrari S (2015) A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. *Neurocomputing* 155:277–285. ISSN 0925-2312. <https://doi.org/10.1016/j.neucom.2014.11.058>
 14. Raissi M, Perdikaris P, Karniadakis GE (2018) Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM J Sci Comput*. <https://doi.org/10.1137/17M1120762>
 15. Raissi M, Wang Z, Triantafyllou MS, Karniadakis GE (2019a) Deep learning of vortex-induced vibrations. *J Fluid Mech* 861:119–137. ISSN 0022-1120, 1469-7645. <https://doi.org/10.1017/jfm.2018.872>
 16. Stiff differential equations. <https://uk.mathworks.com/company/newsletters/articles/stiff-differential-equations.html>
 17. Raissi M, Perdikaris P, Karniadakis GE (2019b) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J Comput Phys* 378:686–707. ISSN 0021-9991. <https://doi.org/10.1016/j.jcp.2018.10.045>
 18. Wang S, Teng Y, Perdikaris P (2021a) Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM J Sci Comput* 43(5):A3055–A3081. ISSN 1064-8275. <https://doi.org/10.1137/20M1318043>
 19. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017a) Attention is all you need. In: *Advances in neural information processing systems*, 2017, vol 30
 20. Cao S (2021) Choose a transformer: Fourier or Galerkin. In: *Advances in neural information processing systems*, 2021, vol 34. Curran Associates, Inc., pp 24924–24940. <https://proceedings.neurips.cc/paper/2021/hash/d0921d442ee91b896ad95059d13df618-Abstract.html>
 21. Gao H, Zahr MJ, Wang J-X (2022) Physics-informed graph neural Galerkin networks: a unified framework for solving PDE-governed forward and inverse problems. *Comput Methods Appl Mech Eng* 390:114502. ISSN 0045-7825. <https://doi.org/10.1016/j.cma.2021.114502>
 22. Sirignano J, Spiliopoulos K (2018) DGM: a deep learning algorithm for solving partial differential equations. *J Comput Phys* 375:1339–1364. ISSN 0021-9991. <https://doi.org/10.1016/j.jcp.2018.08.029>
 23. Yu Y, Si X, Hu C, Zhang J (2019) A review of recurrent neural networks: LSTM cells and network architectures. *Neural Comput* 31(7):1235–1270. ISSN 0899-7667. https://doi.org/10.1162/neco_a_01199
 24. Rahaman N, Baratin A, Arpit D, Draxler F, Lin M, Hamprecht F, Bengio Y, Courville A (2019) On the spectral bias of neural networks. In: *Proceedings of the 36th international conference on machine learning*, 2019. PMLR, pp 5301–5310. ISSN 2640-3498. <https://proceedings.mlr.press/v97/rahaman19a.html>
 25. Tancik M, Srinivasan P, Mildenhall B, Fridovich-Keil S, Raghavan N, Singhal U, Ramamoorthi R, Barron J, Ng R (2020) Fourier features let networks learn high frequency functions in low dimensional domains. In: *Advances in neural information processing systems*, 2020, vol 33. Curran Associates, Inc., pp 7537–7547. <https://proceedings.neurips.cc/paper/2020/hash/55053683268957697aa39fba6f231c68-Abstract.html>
 26. Modulus user guide, release v21.06 (2021). <https://developer.nvidia.com/modulus-user-guide-v2106>
 27. Hennigh O, Narasimhan S, Nabian MA, Subramaniam A, Tangsali K, Fang Z, Rietmann M, Byeon W, Choudhry S (2021) NVIDIA SimNetTM: an AI-accelerated multi-physics simulation framework. In: Paszynski M, Kranzlmüller D, Krzhizhanovskaya VV, Dongarra JJ, Sloot PMA (eds) *Computational science—ICCS 2021*, Lecture notes in computer science, 2021. Springer, Cham, pp 447–461. ISBN 978-3-030-77977-1. https://doi.org/10.1007/978-3-030-77977-1_36
 28. Sitzmann V, Martel J, Bergman A, Lindell D, Wetzstein G (2020) Implicit neural representations with periodic activation functions. In: *Advances in neural information processing systems*, 2020, vol 33. Curran Associates, Inc., pp 7462–7473. <https://proceedings.neurips.cc/paper/2020/hash/53c04118df112c13a8c34b38343b9c10-Abstract.html>
 29. Zhao CL (2020) Solving Allen–Cahn and Cahn–Hilliard equations using the adaptive physics informed neural networks. *Commun Comput Phys* 29(3). <https://doi.org/10.4208/cicp.OA-2020-0086>
 30. McClenny L, Braga-Neto U (2019) Self-adaptive physics-informed neural networks using a soft attention mechanism. Technical Report 68. http://ceur-ws.org/Vol-2964/article_68.pdf
 31. Shi S, Liu D, Zhao Z (2021) Non-Fourier heat conduction based on self-adaptive weight physics-informed neural networks. In: 2021 40th Chinese control conference (CCC), pp 8451–8456. ISSN 1934-1768. <https://doi.org/10.23919/CCC52363.2021.9550487>
 32. Wang S, Yu X, Perdikaris P (2022a) When and why PINNs fail to train: a neural tangent kernel perspective. *J Comput Phys* 449:110768. ISSN 0021-9991. <https://doi.org/10.1016/j.jcp.2021.110768>
 33. Sun R-Y (2020) Optimization for deep learning: an overview. *J Oper Res Soc China* 8(2):249–294. ISSN 2194-6698. <https://doi.org/10.1007/s40305-020-00309-6>
 34. Pascanu R, Mikolov T, Bengio Y (2013) On the difficulty of training recurrent neural networks. In: *Proceedings of the 30th international conference on machine learning*. PMLR, pp 1310–1318. ISSN 1938-7228. <https://proceedings.mlr.press/v28/pascanu13.html>

35. Fletcher R (1994) An overview of unconstrained optimization. In: Spedicato E (ed) Algorithms for continuous optimization: the state of the art, NATO ASI series. Springer, Dordrecht, pp 109–143. ISBN 978-94-009-0369-2. https://doi.org/10.1007/978-94-009-0369-2_5
36. Tan HH, Lim KH (2019) Review of second-order optimization techniques in artificial neural networks backpropagation. IOP Conf Ser Mater Sci Eng 495:012003. ISSN 1757-899X. <https://doi.org/10.1088/1757-899X/495/1/012003>
37. Lu L, Meng X, Mao Z, Karniadakis GE (2021b) DeepXDE: a deep learning library for solving differential equations. SIAM Rev 63(1):208–228. ISSN 0036-1445. <https://doi.org/10.1137/19M1274067>
38. Haghghat E, Juanes R (2021) SciANN: a Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. Comput Methods Appl Mech Eng 373:113552. ISSN 0045-7825. <https://doi.org/10.1016/j.cma.2020.113552>
39. McClenny LD, Haile MA, Braga-Neto UM (2021) TensorDiffEq: scalable multi-GPU forward and inverse solvers for physics informed neural networks. Technical Report. arXiv [arXiv:2103.16034](https://arxiv.org/abs/2103.16034)
40. Zubov K, McCarthy Z, Ma Y, Calisto F, Pagliarino V, Azeglio S, Bottero L, Luján E, Sulzer V, Bharambe A, Vinchhi N, Balakrishnan K, Upadhyay D, Rackauckas C (2021) NeuralPDE: automating physics-informed neural networks (PINNs) with error approximations. Technical Report. arXiv [arXiv:2107.09443](https://arxiv.org/abs/2107.09443)
41. Schiassi E, Leake C, De Florio M, Johnston H, Furfaro R, Mortari D (2020) Extreme theory of functional connections: a physics-informed neural network method for solving parametric differential equations. Technical Report. arXiv [arXiv:2005.10632](https://arxiv.org/abs/2005.10632)
42. Demo N, Strazzullo M, Rozza G (2021) An extended physics informed neural network for preliminary analysis of parametric optimal control problems. Technical Report. arXiv [arXiv:2110.13530](https://arxiv.org/abs/2110.13530)
43. Raj M, Kumbhar P, Annabattula RK (2022) Physics-informed neural networks for solving thermo-mechanics problems of functionally graded material. Technical Report. arXiv [arXiv:2111.10751](https://arxiv.org/abs/2111.10751)
44. Heger P, Full M, Hilger D, Hosters N (2022) Investigation of physics-informed deep learning for the prediction of parametric, three-dimensional flow based on boundary data. Technical Report. arXiv [arXiv:2203.09204](https://arxiv.org/abs/2203.09204)
45. Wu Y, Feng J (2018) Development and application of artificial neural network. Wirel Pers Commun 102(2):1645–1656. ISSN 1572-834X. <https://doi.org/10.1007/s11277-017-5224-x>
46. Trehan D (2020) Non-convex optimization: a review. In: 2020 4th International conference on intelligent computing and control systems (ICICCS), pp 418–423. <https://doi.org/10.1109/ICICCS48265.2020.9120874>
47. Chen T, Chen H (1995) Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Trans Neural Netw 6(4):911–917. ISSN 1941-0093. <https://doi.org/10.1109/72.392253>
48. Freedman D (2009) Statistical models: theory and practice. Cambridge University Press. ISBN 978-0-521-11243-7. Google-Books-ID 4N3KOEitRe8C
49. Thacker WC (1989) The role of the Hessian matrix in fitting models to measurements. J Geophys Res Oceans 94(C5):6177–6196. ISSN 2156-2202. <https://doi.org/10.1029/JC094iC05p06177>
50. Diaconis P, Shahshahani M (1984) On nonlinear functions of linear combinations. SIAM J Sci Stat Comput 5(1):175–191. ISSN 0196-5204. <https://doi.org/10.1137/0905013>
51. Werbos P (1974) Beyond regression: new tools for prediction and analysis in the behavior science. Doctoral Dissertation, Harvard University. <https://ci.nii.ac.jp/naid/10012540025/>
52. Minsky M, Papert SA (2017) Perceptrons: an introduction to computational geometry. The MIT Press. ISBN 978-0-262-34393-0. <https://doi.org/10.7551/mitpress/11301.001.0001>
53. Molnar C (2020) Interpretable machine learning. Lulu.com. ISBN 0-244-76852-8
54. Wang Q, Ma Y, Zhao K, Tian Y (2022b) A comprehensive survey of loss functions in machine learning. Ann Data Sci 9(2):187–212. ISSN 2198-5812. <https://doi.org/10.1007/s40745-020-00253-5>
55. Elshawi R, Wahab A, Barnawi A, Sakr S (2021) DLBench: a comprehensive experimental evaluation of deep learning frameworks. Clust Comput 24(3):2017–2038. ISSN 1573-7543. <https://doi.org/10.1007/s10586-021-03240-4>
56. Ghoghogh B, Ghodsi A, Karray F, Crowley M (2021) KKT conditions, first-order and second-order optimization, and distributed optimization: tutorial and survey. [arXiv:2110.01858](https://arxiv.org/abs/2110.01858) [cs, math]
57. Ketkar N (2017) Stochastic gradient descent. In: Ketkar N (ed) Deep learning with Python: a hands-on introduction. Apress, Berkeley, pp 113–132. ISBN 978-1-4842-2766-4. https://doi.org/10.1007/978-1-4842-2766-4_8
58. Kingma DP, Ba J (2017) Adam: a method for stochastic optimization. Technical Report. arXiv [arXiv:1412.6980](https://arxiv.org/abs/1412.6980)
59. Ramm A, Smirnova A (2001) On stable numerical differentiation. Math Comput 70(235):1131–1153. ISSN 0025-5718, 1088-6842. <https://doi.org/10.1090/S0025-5718-01-01307-2>
60. Davenport JH, Siret Y, Tournier É (1993) Computer algebra systems and algorithms for algebraic computation. Academic Press Professional, Inc. ISBN 0-12-204232-8
61. Barros CDT, Mendonça MRF, Vieira AB, Ziviani A (2021) A survey on embedding dynamic graphs. ACM Comput Surv 55(1):1–37. ISSN: 0360-0300
62. Fang B, Yang E, Xie F (2020) Symbolic techniques for deep learning: challenges and opportunities. arXiv preprint [arXiv:2010.02727](https://arxiv.org/abs/2010.02727)
63. Giles M (2008) An extended collection of matrix derivative results for forward and reverse mode automatic differentiation. Report
64. Mathias R (1996) A chain rule for matrix functions and applications. SIAM J Matrix Anal Appl 17(3):610–620. ISBN: 0895-4798
65. Raschka S, Patterson J, Nolet C (2020) Machine learning in Python: main developments and technology trends in data science, machine learning, and artificial intelligence. Information 11(4):193. ISSN 2078-2489. <https://doi.org/10.3390/info11040193>
66. Andonie R (2019) Hyperparameter optimization in learning systems. J Membr Comput 1(4):279–291. ISSN 2523-8914. <https://doi.org/10.1007/s41965-019-00023-0>
67. Gopakumar V, Pamela S, Samaddar D (2022) Loss landscape engineering via data regulation on PINNs. Technical Report. arXiv [arXiv:2205.07843](https://arxiv.org/abs/2205.07843)
68. Arzani A, Wang J-X, D’Souza RM (2021) Uncovering near-wall blood flow from sparse data with physics-informed neural networks. Phys Fluids 33(7):071905. ISSN 1070-6631. <https://doi.org/10.1063/5.0055600>
69. Bajaj C, McLennan L, Andeen T, Roy A (2021) Robust learning of physics informed neural networks. Technical Report. arXiv [arXiv:2110.13330](https://arxiv.org/abs/2110.13330)
70. Nabian MA, Gladstone RJ, Meidani H (2021) Efficient training of physics-informed neural networks via importance sampling. Comput Aided Civ Infrastruct Eng 36(8):962–977. ISSN 1467-8667. <https://doi.org/10.1111/mice.12685>

71. Robert CP, Casella G (1999) Monte Carlo integration. In: Robert CP, Casella G (eds) Monte Carlo statistical methods, Springer texts in statistics. Springer, New York, pp 71–138. ISBN 978-1-4757-3071-5. https://doi.org/10.1007/978-1-4757-3071-5_3
72. Morokoff WJ, Caffisch RE (1995) Quasi-Monte Carlo integration. *J Comput Phys* 122(2):218–230. ISSN 0021-9991. <https://doi.org/10.1006/jcph.1995.1209>
73. Berrada I, Ferland JA, Michelon P (1996) A multi-objective approach to nurse scheduling with both hard and soft constraints. *Socio-Econ Plan Sci* 30(3):183–193. ISSN 0038-0121. [https://doi.org/10.1016/0038-0121\(96\)00010-9](https://doi.org/10.1016/0038-0121(96)00010-9)
74. Sukumar N, Srivastava A (2022) Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Comput Methods Appl Mech Eng* 389:114333. ISSN 0045-7825. <https://doi.org/10.1016/j.cma.2021.114333>
75. Son H, Jang JW, Han WJ, Hwang HJ (2021) Sobolev training for physics informed neural networks. Technical Report. arXiv [arXiv:2101.08932](https://arxiv.org/abs/2101.08932)
76. Yu J, Lu L, Meng X, Karniadakis GE (2022) Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems. *Comput Methods Appl Mech Eng* 393:114823. ISSN 0045-7825. <https://doi.org/10.1016/j.cma.2022.114823>
77. Jagtap AD, Kawaguchi K, Karniadakis GE (2020) Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *J Comput Phys* 404:109136. ISSN 0021-9991. <https://doi.org/10.1016/j.jcp.2019.109136>
78. Chan T, Zhu W (2005) Level set based shape prior segmentation. In: 2005 IEEE Computer Society conference on computer vision and pattern recognition (CVPR'05), vol 2, pp 1164–1170. ISSN 1063-6919. <https://doi.org/10.1109/CVPR.2005.212>
79. Xiang Z, Peng W, Zhou W, Yao W (2022) Hybrid finite difference with the physics-informed neural network for solving PDE in complex geometries. [arXiv:2202.07926](https://arxiv.org/abs/2202.07926) [physics]
80. Martino L, Elvira V, Louzada F (2017) Effective sample size for importance sampling based on discrepancy measures. *Signal Process* 131:386–401. ISSN 0165-1684. <https://doi.org/10.1016/j.sigpro.2016.08.025>
81. Samsami MR, Alimadad H (2020) Distributed deep reinforcement learning: an overview. Technical Report. arXiv [arXiv:2011.11012](https://arxiv.org/abs/2011.11012)
82. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) Deep reinforcement learning: a brief survey. *IEEE Signal Process Mag* 34(6):26–38. ISSN 1558-0792. <https://doi.org/10.1109/MSP.2017.2743240>
83. Viana FAC (2016) A tutorial on Latin hypercube design of experiments. *Qual Reliab Eng Int* 32(5):1975–1985. ISSN 1099-1638. <https://doi.org/10.1002/qre.1924>
84. Shaw JEH (1988) A quasirandom approach to integration in Bayesian statistics. *Ann Stat* 16(2):895–914. ISSN 0090-5364. <https://www.jstor.org/stable/2241763>
85. Lemieux C (2006) Chapter 12: quasi-random number techniques. In: Henderson SG, Nelson BL (eds) Handbooks in operations research and management science. Simulation, vol 13. Elsevier, pp 351–379. [https://doi.org/10.1016/S0927-0507\(06\)13012-1](https://doi.org/10.1016/S0927-0507(06)13012-1)
86. Halton JH (1960) On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numer Math* 2(1):84–90. ISSN 0945-3245. <https://doi.org/10.1007/BF01386213>
87. Faure H, Lemieux C (2009) Generalized Halton sequences in 2008: a comparative study. *ACM Trans Model Comput Simul* 19(4):15:1–15:31. ISSN 1049-3301. <https://doi.org/10.1145/1596519.1596520>
88. Sobol' IM (1967) On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Comput Math Math Phys* 7(4):86–112. ISSN 00415553. [https://doi.org/10.1016/0041-5553\(67\)90144-9](https://doi.org/10.1016/0041-5553(67)90144-9)
89. Joe S, Kuo FY (2008) Constructing Sobol sequences with better two-dimensional projections. *SIAM J Sci Comput* 30(5):2635–2654. ISSN 1064-8275. <https://doi.org/10.1137/070709359>
90. Hammersley JM (1960) Monte Carlo methods for solving multivariable problems. *Ann NY Acad Sci* 86(3):844–874. ISSN 1749-6632. <https://doi.org/10.1111/j.1749-6632.1960.tb42846.x>
91. Hammersley J (2013) Monte Carlo methods. Springer, Singapore
92. Pang G, Lu L, Karniadakis GE (2019) fPINNs: fractional physics-informed neural networks. *SIAM J Sci Comput* 41(4):A2603–A2626. ISSN 1064-8275. <https://doi.org/10.1137/18M1229845>
93. Zhang D, Guo L, Karniadakis GE (2020) Learning in modal space: solving time-dependent stochastic PDEs using physics-informed neural networks. *SIAM J Sci Comput* 42(2):A639–A665. ISSN 1064-8275. <https://doi.org/10.1137/19M1260141>
94. Wang S, Wang H, Perdikaris P (2021) Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Sci Adv* 7(40):eabi8605. <https://doi.org/10.1126/sciadv.abi8605>
95. Meng X, Karniadakis GE (2020) A composite neural network that learns from multi-fidelity data: application to function approximation and inverse PDE problems. *J Comput Phys* 401:109020. ISSN 0021-9991. <https://doi.org/10.1016/j.jcp.2019.109020>
96. Lu L, Dao M, Kumar P, Ramamurthy U, Karniadakis GE, Suresh S (2020) Extraction of mechanical properties of materials through deep learning from instrumented indentation. *Proc Natl Acad Sci USA* 117(13):7052–7062. <https://doi.org/10.1073/pnas.1922210117>
97. Wang S, Wang H, Perdikaris P (2021c) On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Comput Methods Appl Mech Eng* 384:113938. ISSN 0045-7825. <https://doi.org/10.1016/j.cma.2021.113938>
98. Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE (2021c) Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat Mach Intell* 3(3):218–229. ISSN 2522-5839. <https://doi.org/10.1038/s42256-021-00302-5>
99. Jin P, Meng S, Lu L (2022) MIONet: learning multiple-input operators via tensor product. Technical Report. [arXiv:2202.06137](https://arxiv.org/abs/2202.06137)
100. Cai S, Wang Z, Lu L, Zaki TA, Karniadakis GE (2021) DeepM&Mnet: inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *J Comput Phys* 436:110296. ISSN 0021-9991. <https://doi.org/10.1016/j.jcp.2021.110296>
101. Mao Z, Lu L, Marxen O, Zaki TA, Karniadakis GE (2021) DeepM&Mnet for hypersonics: predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *J Comput Phys* 447:110698. ISSN 0021-9991. <https://doi.org/10.1016/j.jcp.2021.110698>
102. Lu L, Pestourie R, Johnson SG, Romano G (2022) Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. Technical Report [arXiv:2204.06684](https://arxiv.org/abs/2204.06684)
103. Srivastava RK, Greff K, Schmidhuber J (2015) Training very deep networks. In: Advances in neural information processing systems, vol 8. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2015/hash/215a71a12769b056c3c32e7299f1c5ed-Abstract.html>
104. Guibas J, Mardani M, Li Z, Tao A, Anandkumar A, Catanzaro B (2022) Adaptive Fourier neural operators: efficient token mixers for transformers. Technical Report. [arXiv:2111.13587](https://arxiv.org/abs/2111.13587)
105. Li Z, Zheng H, Kovachki N, Jin D, Chen H, Liu B, Azizzadehsheli K, Anandkumar A (2021) Physics-informed neural

- operator for learning partial differential equations. arXiv preprint [arXiv:2111.03794](https://arxiv.org/abs/2111.03794)
106. Isola P, Zhu J-Y, Zhou T, Efros AA (2017) Image-to-image translation with conditional adversarial networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp 1125–1134
 107. Wang T-C, Liu M-Y, Zhu J-Y, Tao A, Kautz J, Catanzaro B (2018) High-resolution image synthesis and semantic manipulation with conditional GANs. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp 8798–8807
 108. Ledig C, Theis L, Huszár F, Caballero J, Cunningham A, Acosta A, Aitken A, Tejani A, Totz J, Wang Z, Shi W (2017) Photo-realistic single image super-resolution using a generative adversarial network. In: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp 4681–4690
 109. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017b) Attention is all you need. In: Advances in neural information processing systems, vol 30. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fd053c1c4a845aa-Abstract.html>
 110. Partial differential equation toolbox (R2022a). <https://uk.mathworks.com/products/pde.html>
 111. Lewkowycz A (2021) How to decay your learning rate. [arXiv:2103.12682](https://arxiv.org/abs/2103.12682) [cs]
 112. Elfving S, Uchibe E, Doya K (2017) Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Technical Report. arXiv [arXiv:1702.03118](https://arxiv.org/abs/1702.03118)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.