Computer Methods in Applied Mechanics and Engineering 429 (2024) 117116

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

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

# Multilevel domain decomposition-based architectures for physics-informed neural networks

Victorita Dolean [a], Alexander Heinlein [b], Siddhartha Mishra [c], Ben Moseley [c,*]

[a] Centre for Analysis Scientific Computing and Applications (CASA), Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands
[b] Delft Institute of Applied Mathematics, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
[c] Seminar for Applied Mathematics (SAM) D-MATH, and ETH AI Center, ETH Zurich, Switzerland

A R T I C L E  I N F O

Dataset link: https://github.com/benmoseley/FBPINNs/tree/multilevel-paper/multilevel-paper

*Keywords:*
Physics-informed neural networks
Overlapping domain decomposition methods
Multilevel methods
Multi-scale modeling
Spectral bias
Forward modeling
Differential equations

A B S T R A C T

Physics-informed neural networks (PINNs) are a powerful approach for solving problems involving differential equations, yet they often struggle to solve problems with high frequency and/or multi-scale solutions. Finite basis physics-informed neural networks (FBPINNs) improve the performance of PINNs in this regime by combining them with an overlapping domain decomposition approach. In this work, FBPINNs are extended by adding multiple levels of domain decompositions to their solution ansatz, inspired by classical multilevel Schwarz domain decomposition methods (DDMs). Analogous to typical tests for classical DDMs, we assess how the accuracy of PINNs, FBPINNs and multilevel FBPINNs scale with respect to computational effort and solution complexity by carrying out strong and weak scaling tests. Our numerical results show that the proposed multilevel FBPINNs consistently and significantly outperform PINNs across a range of problems with high frequency and multi-scale solutions. Furthermore, as expected in classical DDMs, we show that multilevel FBPINNs improve the accuracy of FBPINNs when using large numbers of subdomains by aiding global communication between subdomains.

## 1. Introduction

Scientific machine learning (SciML) [1–5] is an emerging and rapidly growing field of research. The central goal of SciML is to provide accurate, efficient, and robust tools for carrying out scientific research by tightly combining scientific understanding with machine learning (ML). The field has provided many such tools which have enhanced traditional approaches, from accelerating simulation algorithms to discovering new scientific phenomena.

One popular SciML approach is physics-informed neural networks (PINNs) [6,7]. PINNs solve forward and inverse problems related to differential equations by using a neural network to directly approximate the solution to the differential equation. They are trained by using a loss function which minimizes the residual of the differential equation over a set of collocation points. The initial concepts behind PINNs were introduced by [6] and others, and later re-implemented and extended in [7]. One of the advantages of PINNs over traditional methods for solving differential equations such as finite difference (FD) and finite element methods (FEM) is that they provide a mesh-free approach, paving the way for the application of problems with complex geometry or in very high spatial dimensions; cf. [8]. Furthermore, they can easily be extended to solve inverse problems by incorporating observational data.

Since their invention, PINNs have been employed across a wide range of domains [3,9]. For example, they have been used to solve forward and inverse problems in geophysics [10], fluid dynamics [11–13], and optics [14]. Many extensions of PINNs

* Corresponding author.
*E-mail address:* benjamin.moseley@ai.ethz.ch (B. Moseley).

https://doi.org/10.1016/j.cma.2024.117116
Received 14 December 2023; Received in revised form 31 May 2024; Accepted 1 June 2024
Available online 20 June 2024

have also been proposed. For example, PINNs have been extended to carry out uncertainty quantification [15], learn fast surrogate models [16,17], and carry out equation discovery [18].

However, PINNs suffer from a number of limitations. One is that, compared to traditional methods, their convergence properties are poorly understood, although some work has started to explore this [19–21]. Another limitation is that, compared to traditional methods, the computational cost of training PINNs is relatively high, especially when they are only used for forward modeling [9]. Finally, a major limitation of PINNs is that they often struggle to solve problems with high frequency and/or multi-scale solutions [22,23]. Typically, as higher frequencies and multi-scale features are added to the solution, the accuracy of PINNs usually rapidly reduces and their computational cost rapidly increases in a super-linear fashion [22].

There are multiple reasons for this behavior. One is the spectral bias of neural networks, which is the well-studied property that neural networks tend to learn high frequencies much slower than low frequencies [24–27]. Another is that, as higher frequencies and more multi-scale features are added, more collocation points and a larger neural network with significantly more free parameters are typically required to accurately approximate the solution. This creates a significantly more complex optimization problem when training the PINN.

Recently, [22] proposed finite basis physics-informed neural networks (FBPINNs), which aim to improve the performance of PINNs in this regime by using an overlapping domain decomposition (DD) approach. In particular, instead of using a single neural network to approximate the solution to the differential equation, many smaller neural networks were placed in overlapping subdomains and summed together to represent the solution. On the one hand, FBPINNs can be seen as a DD-based network architecture for PINNs. On the other hand, by taking this "divide and conquer" approach, the global PINN optimization problem is transformed into many smaller local optimization problems, which are coupled implicitly due to the overlap of the subdomains and their globally defined loss function. The results in [22] show that this approach significantly improves the accuracy and reduces the training cost of PINNs when solving differential equations with high frequency and multi-scale solutions.

In this work, we significantly extend FBPINNs by incorporating *multilevel modeling* into their design. In particular, instead of using a single DD in their solution ansatz, we add multiple levels of overlapping DDs. This idea is inspired by classical DDMs, where coarse levels are required for numerical scalability when using large numbers of subdomains. Furthermore, to assess the performance of multilevel FBPINNs, we define strong and weak scaling tests for measuring how the accuracy of PINNs and FBPINNs scale with computational effort and solution complexity, analogous to the strong and weak scaling tests commonly used in classical DDMs.

Given these extensions, the performance of PINNs, (one-level) FBPINNs, and multilevel FBPINNs across a range of high frequency and multi-scale problems is investigated. We also compare multilevel FBPINNs to PINNs with Fourier input features [23,28] and self-adaptive PINNs (SA-PINNs) [29], which have both been shown to improve the accuracy of PINNs when solving multi-scale problems. Across these tests, we find that multilevel FBPINNs significantly outperform PINNs in terms of accuracy and computational cost. Furthermore, as expected in classical DDMs, we show that multilevel FBPINNs improve the accuracy of FBPINNs when using large numbers of subdomains by aiding global communication between subdomains.

The remainder of this work is structured as follows. In Section 1.1 we discuss related work on combining ML, PINNs, and DD, and in Sections 1.2 and 1.3 we give a brief overview of neural networks and PINNs. Then we define FBPINNs and extend them to multilevel FBPINNs in Section 2. Our strong and weak scaling tests and corresponding numerical results on the performance of PINNs, FBPINNs, and multilevel FBPINNs across a range of high frequency and multi-scale problems are discussed in Section 3. Finally, in Section 4, we discuss the implications and limitations of our work and further research directions.

### 1.1. Related work

In general, the idea of combining ML with classical DDMs is not new; for early works on using ML to predict the geometrical location of constraints in adaptive finite element tearing and interconnecting (FETI) and balancing DD by constraints (BDDC) methods; see [30]. An overview of the first attempts on combining DD and ML can be found in [31], a more recent overview is given in [32].

For specifically combining PINNs with DD, some of the first methods in this area were the deep domain decomposition method (D3M) [33], the deep-learning-based domain decomposition method (DeepDDM) [34,35], and its two-level variant [36], which use PINNs to solve local problems and overlapping Schwarz steps to iteratively connect them based on Lions' parallel Schwarz algorithm [37]. At the same time, a series of other extensions, like conservative physics-informed neural network (cPINN) and extended physics-informed neural networks (XPINNs) [38] were proposed, which similarly divide the domain and use PINNs to solve each local problem; here, typically a non-overlapping DD is used. A detailed comparison of these methods to FBPINNs is given in Section 2.1.3.

In [39], partition of unity functions, similar to the window functions used in the FBPINN method, are learned. However, this is done in a pure function approximation setting rather than in the solution of PDE-based problems with PINNs.

### 1.2. Neural networks

We first provide a basic definition of a neural network. For the purpose of this work, we simply consider a neural network to be a mathematical function with some learnable parameters. More precisely, the network is defined as $u(\mathbf{x}, \boldsymbol{\theta}) : \mathbb{R}^{d_x} \times \mathbb{R}^{d_\theta} \to \mathbb{R}^{d_u}$, where $\mathbf{x}$ are some inputs to the network, $\boldsymbol{\theta}$ are a set of learnable parameters, and $d_x$, $d_\theta$, and $d_u$ are the dimensionality of the network's inputs, parameters, and outputs. In a traditional supervised learning setting, learning typically consists of fitting the network function to some training data containing example inputs and outputs, by minimizing a loss function with respect to $\theta$ which penalizes the difference between the network's outputs and the training data.

The exact form of the network function is determined by the neural network's architecture. In this work, we solely use feedforward fully connected networks (FCNs) [40]. In this case, the network function is given by

$$u(\mathbf{x},\boldsymbol{\theta}) = f_n \circ \ldots \circ f_i \circ \ldots \circ f_1(\mathbf{x},\boldsymbol{\theta}) \tag{1}$$

where now $\mathbf{x} \in \mathbb{R}^{d_0}$ is the input to the FCN, $u \in \mathbb{R}^{d_n}$ is the output of the FCN, $n$ is the number of layers (depth) of the FCN, and $f_i(\mathbf{x},\boldsymbol{\theta}) = \sigma_i(W_i\mathbf{x} + \mathbf{b}_i)$ where $\boldsymbol{\theta}_i = (W_i, \mathbf{b}_i)$, $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ are known as weight matrices, $\mathbf{b}_i \in \mathbb{R}^{d_i}$ are known as bias vectors, $\sigma_i$ are element-wise activation functions commonly chosen as rectified linear unit (ReLU), hyperbolic tangent, or identity functions, and $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_i, \ldots, \boldsymbol{\theta}_n)$ are the set of learnable parameters of the network. Note that only the nonlinear activation functions $\sigma_i$ facilitate nonlinearity of the network function.

## 1.3. Physics-informed neural networks

Physics-informed neural networks (PINNs) [6,7] use neural networks to solve problems related to differential equations. In particular, PINNs focus on solving boundary value problems of the form

$$\begin{aligned} \mathcal{N}[u](\mathbf{x}) &= f(\mathbf{x}), & \mathbf{x} \in \Omega \subset \mathbb{R}^d, \\ \mathcal{B}_k[u](\mathbf{x}) &= g_k(\mathbf{x}), & \mathbf{x} \in \Gamma_k \subset \partial\Omega \end{aligned} \tag{2}$$

where $\mathcal{N}[u](\mathbf{x})$ is some differential operator, $u(\mathbf{x})$ is the solution, and $\mathcal{B}_k(\cdot)$ are a set of boundary conditions (BCs) which ensure uniqueness of the solution. For the sake of simplicity, we consider BCs in a broad sense; we do not explicitly distinguish between initial and boundary conditions, and the $\mathbf{x}$ variable can include time. Eq. (2) can describe many different differential equation problems, including linear and non-linear problems, time-dependent and time-independent problems, and those with irregular, higher-order, and cyclic boundary conditions.

To solve Eq. (2), PINNs use a neural network to directly approximate the solution, i.e., $u(\mathbf{x},\boldsymbol{\theta}) \approx u(\mathbf{x})$. Note, for simplicity throughout this work, we use the same notation for the true solution and the neural network. It is important to note that PINNs provide a functional approximation to the solution, and not a discretized solution similar to that provided by traditional methods such as finite difference methods, and as such PINNs are a mesh-free approach for solving differential equations. Following the approach proposed by [7], the following loss function is minimized to train the PINN,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{\lambda_I}{N_I}\sum_{i=1}^{N_I}\underbrace{\left(\mathcal{N}[u](\mathbf{x}_i,\boldsymbol{\theta}) - f(\mathbf{x}_i)\right)^2}_{\text{PDE residual}} + \sum_{k=1}^{N_k}\frac{\lambda_B^k}{N_B^k}\sum_{i=1}^{N_B^k}\underbrace{\left(\mathcal{B}_k[u](\mathbf{x}_i^k,\boldsymbol{\theta}) - g_k(\mathbf{x}_i^k)\right)^2}_{\text{BC residual}}. \tag{3}$$

where $\{\mathbf{x}_i\}_{i=1}^{N_I}$ is a set of collocation points sampled in the interior of the domain, $\{\mathbf{x}_j^k\}_{j=1}^{N_B^k}$ is a set of points sampled along each boundary condition, and $\lambda_I$ and $\lambda_B^k$ are well-chosen scalar weights that ensure the terms in the loss function are well balanced. Intuitively, one can see that by minimizing the PDE residual, the method tries to ensure that the solution learned by the network obeys the underlying PDE, and by minimizing the BC residual, the method tries to ensure that the learned solution is unique by matching it to the BCs. Importantly, a sufficient number of collocation and boundary points must be chosen such that the PINN is able to learn a consistent solution across the domain.

Iterative schemes are typically used to optimize this loss function. Usually, variants of the gradient descent (GD) method, such as the Adam optimizer [41], or quasi-Newton methods, such as the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm [42] are employed. These methods require the computation of the gradient of the loss function with respect to the network parameters, which can be computed easily and efficiently using automatic differentiation [43] provided in modern deep learning libraries [44–46]. Note that gradients of the network output with respect to its inputs are also typically required to evaluate the PDE residual in the loss function, and can similarly be obtained and further differentiated through to update the network's parameters using automatic differentiation.

### 1.3.1. Hard constrained PINNs

A downside of training PINNs with the loss function given by Eq. (3) is that the BCs are *softly* enforced. This means the learned solution may deviate from the BCs because the BC term may not be fully minimized. Furthermore, it can be challenging to balance the different objectives of the PDE and BC terms in the loss function, which can lead to poor convergence and solution accuracy [21,47]. An alternative approach, as originally proposed by [6], is to enforce BCs in a *hard* fashion by using the neural network as part of a solution ansatz. More precisely, the solution to the differential equation is instead approximated by $[\mathcal{C}u](\mathbf{x},\boldsymbol{\theta}) \approx u(\mathbf{x})$ where $\mathcal{C}$ is an appropriately selected constraining operator which analytically enforces the BCs [22,48].

To give a simple example, suppose we want to enforce $u(x=0)=0$ when solving a one-dimensional ordinary differential equation (ODE). The constraining operator and solution ansatz could be chosen as $[\mathcal{C}u](x,\boldsymbol{\theta}) = \tanh(x)u(x,\boldsymbol{\theta}) \approx u(\mathbf{x})$. The rationale behind this is that the function $\tanh(x)$ is zero at $0$, forcing the BC to always be obeyed, but non-zero away from $0$, allowing the network to learn the solution away from the BC.

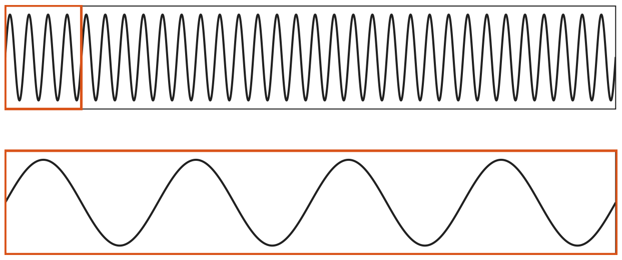

**Fig. 1.** Scaling high frequency problems to low frequency problems using domain decomposition. FBPINNs decompose the domain into many subdomains, and use neural networks within each subdomain to learn the local solution. The input coordinates to each network are normalized to the range [−1, 1] over their individual subdomains. When solving problems with high frequency solutions, this effectively scales each local problem from a high frequency problem to a lower frequency problem, and helps reduce the network's spectral bias.

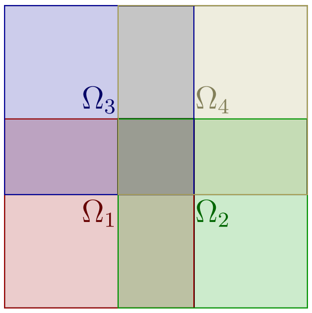

**Fig. 2.** Plot of a square domain $\Omega$ decomposed into four overlapping subdomains, using a uniform rectangular decomposition.

In this approach, the BCs are always satisfied and therefore the BC term in the loss function Eq. (3) can be removed, meaning that the PINN can be trained using the simpler unconstrained loss function,

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\mathcal{N}[\mathcal{C}u](\mathbf{x}_i,\boldsymbol{\theta}) - f(\mathbf{x}_i))^2. \tag{4}$$

where $\{\mathbf{x}_i\}_{i=1}^{N}$ is a set of collocation points sampled in the interior of the domain. Note that, in general, there is no unique way of choosing the constraining operator, and the definition of a suitable constraining operator for complex geometries and/or complex BCs may be difficult or sometimes even impossible, i.e., this strategy is problem dependent; in this case, one may resort to the soft enforcement of boundary conditions Eq. (3) instead.

## 2. Methods

In this section, we define FBPINNs (Section 2.1) and extend them to multilevel FBPINNs (Section 2.2). We also discuss the similarities and differences of FBPINNs and multilevel FBPINNs to classical DDMs (Section 2.2.2).

### 2.1. Finite basis physics-informed neural networks

As discussed in Section 1, a major challenge when training PINNs is that, when higher frequencies and multi-scale features are added to the solution, the accuracy of PINNs usually rapidly reduces and their computational cost rapidly increases in a super-linear fashion [22,23].

In the FBPINN approach [22], instead of using a single neural network to represent the solution, many smaller neural networks are confined in overlapping subdomains and summed together to represent the solution. By taking this "divide and conquer" approach, the global PINN optimization problem is transformed into many smaller coupled local optimization problems.

Furthermore, FBPINNs ensure that the inputs to each subdomain network are normalized over their individual subdomain. When solving problems with high frequency solutions, this effectively scales each local problem from a high frequency problem to a lower frequency problem, and helps limit the effect of spectral bias; Fig. 1 explains this effect further.

#### 2.1.1. Mathematical definition

We now provide a mathematical definition of FBPINNs. First, the global solution domain $\Omega$ is decomposed into $J$ overlapping subdomains $\left\{\Omega_j\right\}_{j=1}^{J}$; cf. Fig. 2. Then, for each subdomain $\Omega_j$, a space of network functions is defined,

$$\mathcal{V}_j = \left\{v_j(\mathbf{x},\boldsymbol{\theta}_j) \mid \mathbf{x}\in\Omega_j, \boldsymbol{\theta}_j\in\Theta_j\right\},$$

where $v_j(\mathbf{x}, \boldsymbol{\theta}_j)$ is a neural network placed in each subdomain and $\Theta_j = \mathbb{R}^{K_j}$ is the linear space of all possible network parameters. Here, $K_j$ is the number of local network parameters which is determined by the network architecture.

Next, each subdomain network is confined to its subdomain by multiplying each network with a window function $\omega_j(\mathbf{x})$, where $\operatorname{supp}(\omega_j) \subset \Omega_j$. Note the neural network functions used in $\mathcal{V}_j$ generally can have global support, and the window functions are used to restrict them to their individual subdomains. Furthermore, we impose that the window functions form a partition of unity, i.e.,

$$\sum_{j=1}^{J} \omega_j \equiv 1 \quad \text{on } \Omega.$$

Given the space of network functions and the window functions, we define a global space decomposition given by $\mathcal{V}$ as

$$\mathcal{V} = \sum_{j=1}^{J} \omega_j \mathcal{V}_j.$$

This space decomposition allows for decomposing any given function $u \in \mathcal{V}$ as follows

$$u = \sum_{j=1}^{J} \omega_j v_j \quad \text{or} \quad u(\mathbf{x}, \boldsymbol{\theta}) = \sum_{j=1}^{J} \omega_j v_j(\mathbf{x}, \boldsymbol{\theta}_j), \tag{5}$$

respectively.

FBPINNs solve the boundary value problem Eq. (2) by using equation Eq. (5) to approximate the solution, and we refer to Eq. (5) as the FBPINN solution. From a PINN perspective, the FBPINN solution can simply be thought of as a specific type of neural network architecture for the PINN which sums together many locally-confined networks to generate the output solution. The same scheme for training PINNs is used to train the FBPINN. More specifically, the FBPINN solution Eq. (5) is substituted into the PINN loss function Eq. (3) and the same iterative optimization scheme is used to learn the parameters $\{\boldsymbol{\theta}_j\}_{j=1}^{J}$ of each subdomain network. FBPINNs can also be trained with hard BCs by using the same constraining operator approach described in Section 1.3.1. In particular, substituting the FBPINN solution Eq. (5) into the hard-constrained loss function Eq. (4) yields the loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} (\mathcal{N}[\mathcal{C} \sum_{j=1}^{J} \omega_j v_j(\mathbf{x}_i, \boldsymbol{\theta}_j)] - f(\mathbf{x}_i))^2. \tag{6}$$

### 2.1.2. Computational efficiency of FBPINNs versus PINNs

Assuming the same size network in each subdomain, naively computing the FBPINN solution Eq. (5) has a time complexity of $\mathcal{O}(NJ\tilde{S})$, where $\tilde{S}$ is the cost of computing the output of a single subdomain network for a single collocation point. This becomes very expensive as more subdomains are added ($J$ increases). However, because the output of each subdomain network is zero outside of the overlapping subdomain after applying the window function, only collocation points within the subdomain need to be included in the summations in Eq. (6). This reduces the computational cost to $\mathcal{O}(NC\tilde{S})$, where $C$ is the average number of subdomains a collocation point belongs to. PINNs have a computational cost of $\mathcal{O}(NS)$, where $S$ is the cost of computing the output of the single global PINN network. Importantly, as the problem complexity increases, the size of the PINN network must typically be increased (increasing $S$), whilst for FBPINNs, we can typically keep the subdomain network size fixed, and increase $J$ instead — thus, often $\tilde{S} \ll S$ and FBPINNs are often orders of magnitude more efficient than PINNs. More details on our efficient software implementation are provided in Appendix A.

### 2.1.3. FBPINNs versus other methods for combining PINNs with DD

Multiple other approaches exist which combine PINNs with DD; a recent overview can be found in [32]. Similar to FBPINNs, XPINNs [38] divide the domain into subdomains and use separate neural networks to solve each subdomain problem. However, XPINNs use a non-overlapping DD. A downside of this approach is that the global solution contains discontinuities at the subdomain interfaces and additional loss terms are required to enforce coupling between subdomain networks. In contrast, FBPINNs do not require additional loss terms and their solution is continuous across subdomain interfaces. XPINNs and FBPINNs are comparable in terms of their computational cost of training, and both are able to use irregular DDs with different types of subdomain neural networks. [49,50] propose a similar approach to XPINNs, but they use extreme learning machines [51] as subdomain networks, where only the parameters of the last layer of the network are learnt. This approach has the advantage of being much faster to train, but the capacity of the subdomain networks is limited, and the performance strongly depends on the initialization.

Other approaches attempt to learn suitable DDs for PINNs. For example, Gated-PINNs [52] use several neural networks, called experts, to propose a solution to a PDE, whilst a gating network is used to weight-average the expert solutions given a coordinate in the domain. Augmented-PINNs (APINNs) [53] build upon this strategy by introducing parameter sharing between experts to capture solution similarities between subdomains. These approaches are more flexible than FBPINNs in that they can adaptively learn DDs, but they are also significantly more computationally expensive to train as they require all experts to be evaluated at each input coordinate and therefore do not scale well with problem size; cf. Section 2.1.2.

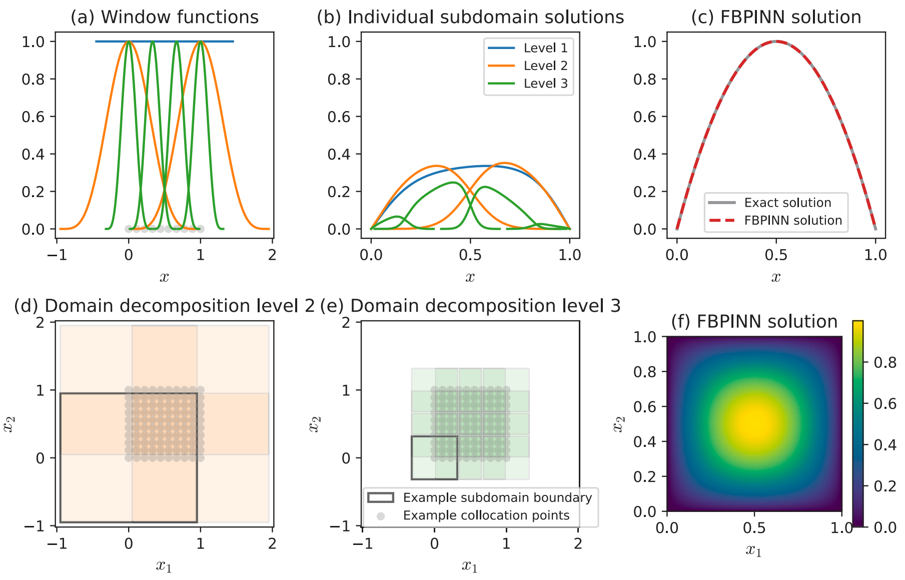

**Fig. 3.** Example of a multilevel FBPINN solving Laplace's equation in one and two dimensions. For the 1D problem, the multilevel FBPINN uses $L = 3$ levels, where each level has 1, 2 and 4 subdomains respectively. The window functions, $\hat{\omega}_j^{(l)}(x)$, used for each level are shown in (a), the individual solutions learned by each subdomain network are shown in (b), and the multilevel FBPINN solution is shown in (c). For the 2D problem, the multilevel FBPINN uses $L = 3$ levels, where each level has $1 \times 1$, $2 \times 2$ and $4 \times 4$ subdomains respectively, using a uniform rectangular DD. The DDs for level 2 and level 3 are plotted in (d) and (e), and the multilevel FBPINN solution is shown in (f). Note the subdomain boundaries and window functions extend past the problem domain (in this case, $[0,1]^d$). Example collocation points used to train the multilevel FBPINN are plotted in (a), (d) and (e).

### 2.2. *Multilevel FBPINNs*

In this work we propose multilevel FBPINNs, which extend FBPINNs by adding multiple levels of DDs to their solution ansatz. They are inspired by classical multilevel DD methods, where coarse levels are generally required for numerical scalability when using large numbers of subdomains, and multilevel approaches may significantly improve performance; see, for instance, [54,55]. Our hypothesis is that adding multilevel modeling to FBPINNs similarly improves their performance. The generalization of FBPINNs to two levels was briefly discussed in [56] and we fully introduce the concept here.

A multilevel FBPINN is defined as follows. First, we define $L$ levels of DDs, where each level, $l$, defines an overlapping DD of $\Omega$ with $J^{(l)}$ subdomains, i.e.,

$$D^{(l)} = \left\{ \Omega_j^{(l)} \right\}_{j=1}^{J^{(l)}},$$

for $j = 1, \dots, J^{(l)}$. Without loss of generality, let $J^{(1)} = 1$, that is, on the first level, we only have one subdomain $\Omega_j^{(1)} = \Omega$. Moreover, we let $J^{(1)} < J^{(2)} < \cdots < J^{(l)}$, meaning that the number of subdomains increases from one to the next level.

Next, we define spaces of network functions for each level,

$$\mathcal{V}_j^{(l)} = \left\{ v_j^{(l)}(\mathbf{x}, \boldsymbol{\theta}_j^{(l)}) \mid \mathbf{x} \in \Omega_j^{(l)}, \boldsymbol{\theta}_j^{(l)} \in \Theta_j^{(l)} \right\}, \quad j = 1, \dots, J^{(l)},\ l = 1, \dots, L,$$

as well as a partition of unity for each level using window functions, $\omega_j^{(l)}$, with

$$\operatorname{supp}\left(\omega_j^{(l)}\right) \subset \Omega_j^{(l)} \quad \text{and} \quad \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \equiv 1 \quad \text{on } \Omega \quad \forall l.$$

We can then define a global space decomposition,

$$\mathcal{V} = \frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \mathcal{V}_j^{(l)},$$

and use this space decomposition to decompose any given function $u \in \mathcal{V}$ as follows,

$$u = \frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} v_j^{(l)} \quad \text{or} \quad u(\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} v_j^{(l)}(\mathbf{x}, \boldsymbol{\theta}_j^{(l)}). \tag{7}$$

We refer to Eq. (7) as the multilevel FBPINN solution. Note, the original FBPINN solution described in Section 2.1 can be obtained by simply setting $L = 1$; we refer to these as one-level FBPINNs going forward.

Analogous to FBPINNs, we can train multilevel FBPINNs by using the same training scheme as PINNs and inserting Eq. (7) into the PINN loss function. When using the hard-constrained PINN loss function Eq. (4), this yields the corresponding multilevel FBPINN loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}(\mathcal{N}[\mathcal{C}\frac{1}{L}\sum_{l=1}^{L}\sum_{j=1}^{J^{(l)}}\omega_j^{(l)} v_j^{(l)}(\mathbf{x}_i,\boldsymbol{\theta}_j^{(l)})] - f(\mathbf{x}_i))^2. \tag{8}$$

### 2.2.1. Example of a multilevel FBPINN

We now show a simple example of a multilevel FBPINN to aid understanding. In particular, we use a multilevel FBPINN to solve the Laplacian boundary value problem,

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega = [0,1]^d, \\ u &= 0 \quad \text{on } \partial\Omega. \end{aligned}$$

First we consider the 1D case ($d = 1$), and set $f = 8$. Then the exact solution is given by $u(x) = 4x(1 - x)$.

We create an $L = 3$ level FBPINN to solve this problem, with $J^{(1)} = 1$, $J^{(2)} = 2$, and $J^{(3)} = 4$. Each level uses a uniform DD given by

$$\Omega_j^{(l)} = \begin{cases} \left[0.5 - \delta/2, 0.5 + \delta/2\right] & l = 1, \\ \left[\frac{(j-1)-\delta/2}{J^{(l)}-1}, \frac{(j-1)+\delta/2}{J^{(l)}-1}\right] & l > 1, \end{cases} \tag{9}$$

where $\delta$ is defined as the *overlap ratio* and is fixed at a value of $\delta = 1.9$. Note that an overlap ratio of less than 1 means that the subdomains are no longer overlapping. The subdomain window functions form a partition of unity for each level and are given by

$$\omega_j^{(l)} = \frac{\hat{\omega}_j^{(l)}}{\sum_{j=1}^{J^{(l)}}\hat{\omega}_j^{(l)}} \quad \text{where} \quad \hat{\omega}_j^{(l)}(x) = \begin{cases} 1 & l = 1 \\ [1 + \cos(\pi(x - \mu_j^{(l)})/\sigma_j^{(l)})]^2 & l > 1, \end{cases}$$

where $\mu_j^{(l)} = (j-1)/(J^{(l)} - 1)$ and $\sigma_j^{(l)} = (\delta/2)/(J^{(l)} - 1)$ represent the center and half-width of each subdomain respectively. The window functions for each level are plotted in Fig. 3(a). A FCN Eq. (1) with 1 hidden layer, 16 hidden units, and tanh activation functions is placed in each subdomain, and the $x$ inputs to each subdomain network are normalized to the range $[-1, 1]$ over their individual subdomains.

Note that multilevel FBPINNs are not restricted to the particular choice of DD, level structure, window function, partition of unity, and subdomain network architecture used above. This choice may not be optimal, and the optimal choice is clearly problem-dependent. For example, it may be beneficial to use an irregular DD, level structure, and varying subdomain network sizes for problems where the solution has varying complexity in different parts of the domain.

The multilevel FBPINN is trained using the hard-constrained loss function Eq. (8) with a constraining operator given by $[\mathcal{C}u](x,\boldsymbol{\theta}) = \tanh(x/\sigma)\tanh((1-x)/\sigma)u(x,\boldsymbol{\theta})$ and $\sigma = 0.2$. The loss function is minimized using the Adam optimizer with a learning rate of $1 \times 10^{-3}$ and $N = 80$ uniformly-spaced collocation points across the domain.

The resulting multilevel FBPINN solution is shown in Fig. 3(c), and the individual subdomain network solutions (with the constraining operator and window function applied) are shown in Fig. 3(b). In this case, we find the FBPINN closely matches the exact solution.

Next we consider the 2D case ($d = 2$), and set $f(x_1, x_2) = 32(x_1(1 - x_1) + x_2(1 - x_2))$. Then the exact solution is given by $u(x_1, x_2) = 16(x_1(1 - x_1)x_2(1 - x_2))$.

In this case we create a $L = 3$ level FBPINN to solve this problem, using a uniform rectangular DD for each level with $J^{(1)} = 1 \times 1 = 1$, $J^{(2)} = 2 \times 2 = 4$, and $J^{(3)} = 4 \times 4 = 16$, as shown in Fig. 3(d) and (e). The size of each subdomain along each dimension is defined similar as in Eq. (9) using, again, an overlap ratio of $\delta = 1.9$. The subdomain window functions are given by

$$\omega_j^{(l)} = \frac{\hat{\omega}_j^{(l)}}{\sum_{j=1}^{J^{(l)}}\hat{\omega}_j^{(l)}}, \quad \text{where} \quad \hat{\omega}_j^{(l)}(\mathbf{x}) = \begin{cases} 1 & l = 1 \\ \prod_i^d [1 + \cos(\pi(x_i - \mu_{ij}^{(l)})/\sigma_{ij}^{(l)})]^2 & l > 1, \end{cases} \tag{10}$$

where $\mu_{ij}^{(l)}$ and $\sigma_{ij}^{(l)}$ represent the center and half-width of each subdomain along each dimension, respectively. A FCN Eq. (1) with 1 hidden layer, 16 hidden units, and tanh activation functions is placed in each subdomain, and the $\mathbf{x}$ inputs to each subdomain network are normalized to the range $[-1, 1]$ along each dimension over their individual subdomains.

Similar to above, the multilevel FBPINN is trained using the hard-constrained loss function Eq. (8), using a constraining operator given by

$$[\mathcal{C}u](\mathbf{x},\boldsymbol{\theta}) = \tanh(x_1/\sigma)\tanh((1-x_1)/\sigma)\tanh(x_2/\sigma)\tanh((1-x_2)/\sigma)u(\mathbf{x},\boldsymbol{\theta}),$$

with $\sigma = 0.2$. The loss function is minimized using the Adam optimizer with a learning rate of $1 \times 10^{-3}$ and $N = 80 \times 80 = 6{,}400$ uniformly-spaced collocation points across the domain.

The resulting multilevel FBPINN solution is shown in Fig. 3(f). Similar to the 1D case, we find the multilevel FBPINN solution closely matches the exact solution.

### 2.2.2. *Multilevel FBPINNs versus classical multilevel DDMs*

Whilst multilevel FBPINNs are inspired by classical multilevel DDMs, a number of differences and similarities exist between these approaches. We believe it is insightful to briefly discuss these below.

Most classical DDMs can be described in terms of the abstract Schwarz framework [54,57]. Similar to FBPINNs, this framework is based on a decomposition of a global function space $V$ into local spaces $\{V_j\}_{j=1}^J$ defined on overlapping subdomains $\Omega_j$, where

$$V = \sum_{j=1}^{J} R_j^\top V_j. \tag{11}$$

Here, $R_j^\top : V_j \to V$ is an interpolation respectively prolongation operator from the local into the global space. These notions can be defined in a similar fashion at the continuous and discrete level. For the sake of simplicity, we suppose here a variational discretization of the PDE to solve. The space decomposition Eq. (11) allows for decomposing any given discrete function $u \in V$ as

$$u = \sum_{j=1}^{J} R_j^\top v_j, \quad v_j \in V_j; \tag{12}$$

due to the overlap, this decomposition is generally not unique. Schwarz DDMs are then based on solving local overlapping problems corresponding to the local spaces $\{V_j\}_{j=1}^J$ and merging them via the prolongation operators $R_j^\top$.

Classical one-level Schwarz methods based on this framework are typically not scalable to large numbers of subdomains. In particular, since information is only transported via the overlap, their rate of convergence will deteriorate when increasing the number of subdomains [54]. In order to fix this, multilevel methods add coarser problems to the Schwarz framework to facilitate the global transfer of information; in particular, the coarsest level typically corresponds to a global problem.

We note that:

- In classical Schwarz methods, the global discretization space $V$ is often fixed first, and then, the local spaces $\{V_j\}_{j=1}^J$ are constructed. In FBPINNs, we do the opposite; we define a local space of neural network functions on an overlapping DD $\{\Omega_j\}_{j=1}^J$ and construct the global discretization space from them.
- In classical Schwarz methods, the local functions $v_j \in V_j$ are generally not defined on the global domain $\Omega$ outside the overlapping subdomain $\Omega_j$; the prolongation operators $R_j^\top$ extend the local functions to $\Omega$ such that $\mathrm{supp}(R_j^\top v_j) \subset \Omega_j, \forall v_j \in V_j$. On the other hand, in FBPINNs, the local neural network functions $v_j$ generally have global support, and the window functions $\omega_j$ are used to confine them to their subdomains. This difference stems from the fact that the local neural networks are not based on a spatial discretization but a function approximation; cf. Section 1.3. Nonetheless, both the prolongation operators and the window functions ensure locality; cf. Eqs. (5) and (12). Note that the prolongation operators in the restricted additive Schwarz (RAS) method [58] also include a partition of unity, such that they are very close to the window functions in FBPINNs.
- A key difference is how the boundary value problem is solved. Whereas in DDMs, local subdomain problems are explicitly defined and solved in a global iteration, in FBPINNs, the global loss function is minimized. Moreover, classical DDMs can exploit properties of the system to be solved. For instance, if the PDE is linear elliptic, convergence guarantees for classical DDMs can be derived; cf. [54,55]. In FBPINNs, we always have to solve a non-convex optimization problem Eq. (3) or Eq. (4), which makes the derivation of convergence bounds difficult. Note that there are also nonlinear overlapping DDMs, for instance, additive Schwarz preconditioned inexact Newton (ASPIN) [59] and additive Schwarz preconditioned exact Newton (RASPEN) methods [60].

## 3. Numerical results

In this section we assess the performance of multilevel FBPINNs. In particular, we investigate the accuracy and computational cost of using multilevel FBPINNs to solve various differential equations, and compare them to PINNs, PINNs with Fourier input features, SA-PINNs, and one-level FBPINNs.

First, in Section 3.1, we introduce the problems studied. Then, in Section 3.2 we introduce a notion of strong and weak scaling, inspired by classical DDMs, for assessing how the accuracy of FBPINNs and PINNs scales with computational effort and solution complexity. In Section 3.3, we list the common implementation details used across all experiments. Finally, in Section 3.4 we present our numerical results.

### 3.1. *Problems studied*

The following problems are used to assess the performance of multilevel FBPINNs;

#### 3.1.1. *Homogeneous Laplacian problem in two dimensions*

First, we consider the 2D homogeneous Laplacian problem already presented above, namely

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega = [0,1]^2, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{13}$$

where

$$f(x_1, x_2) = 32(x_1(1 - x_1) + x_2(1 - x_2)).$$

In this case, the exact solution is given by

$$u(x_1, x_2) = 16(x_1(1 - x_1)x_2(1 - x_2)).$$

This problem is used to carry out simple ablation tests of the multilevel FBPINN. In particular, we assess how varying the number of levels and subdomains as well as the overlap ratio and size of the subdomain networks (architecture) affects the multilevel FBPINN performance.

### *3.1.2. Multi-scale Laplacian problem in two dimensions*

Next, we consider a multi-scale variant of the Laplacian problem Eq. (13) above by using the source term

$$f(x_1, x_2) = \frac{2}{n}\sum_{i=1}^{n}\left(\omega_i \pi\right)^2 \sin(\omega_i \pi x_1)\ \sin(\omega_i \pi x_2). \tag{14}$$

Then, the exact solution is given by

$$u(x_1, x_2) = \frac{1}{n}\sum_{i=1}^{n} \sin(\omega_i \pi x_1)\ \sin(\omega_i \pi x_2).$$

In this case, multi-scale frequencies are contained in the solution, and the values of $n$ and $\omega_i$ allow us to control the number of components and the frequency of each component. We use this problem to assess how the performance of the multilevel FBPINN scales when more multi-scale components are added to the solution.

### *3.1.3. Helmholtz problem in two dimensions*

Lastly, we study the 2D Helmholtz problem

$$\begin{aligned} \Delta u - k^2 u &= f \quad \text{in } \Omega = [0,1]^2, \\ u &= 0 \quad \text{on } \partial\Omega, \\ f(\mathbf{x}) &= e^{-\frac{1}{2}(\|\mathbf{x}-0.5\|/\sigma)^2}, \end{aligned} \tag{15}$$

with a constant (scalar) wave number, $k$. Here, homogeneous Dirichlet boundary conditions and a Gaussian point source with a scalar width, $\sigma$, placed in the center of the domain are used. Note that, for this problem, the exact solution is not known, and instead, we compare our models to the solution obtained from FD modeling, as described in Appendix B.

In this case, the solution contains complex patterns of standing waves where the dominant frequency of the solution depends on the wave number, $k$. We use this problem to test the multilevel FBPINN on a more realistic problem. We first carry out some simple ablation tests by assessing how varying the number of levels, subdomains, overlap ratio and size of the subdomain networks affects the multilevel FBPINN performance. Then, we assess how the performance of the multilevel FBPINN scales when the value of $k$ is increased.

## *3.2. Definition of strong and weak scaling*

For both the multi-scale Laplacian and Helmholtz problems, we carry out strong and weak scaling tests. These assess how the accuracy of the multilevel FBPINN scales with computational effort and solution complexity and are inspired by the strong and weak scaling tests commonly used in classical DD. They are defined in the following way;

- *Strong scaling:* We fix the complexity of the problem and increase the model capacity. For optimal scaling, we expect the convergence rate and/or accuracy to improve at the same rate as the increase of model capacity.
- *Weak scaling:* We increase the complexity of the problem and the model capacity at the same rate. For optimal scaling, we expect the convergence rate and/or accuracy to stay approximately constant.

For all our tests, increasing the model capacity means increasing the number of levels, number of subdomains, and/or the size of the subdomain networks. The exact factors varied and their rates of increase are detailed in the relevant results sections below. Note all of the multilevel FBPINNs tested have been trained on a single GPU, and hence we only show strong and weak scaling tests with respect to model capacity and not hardware parallelization.

## *3.3. Common implementation details*

Many of the implementation details of the multilevel FBPINNs, one-level FBPINNs, and PINNs tested are the same across all tests. These details are presented here; some are only changed for ablation studies, in which case they are described in the relevant results section below.

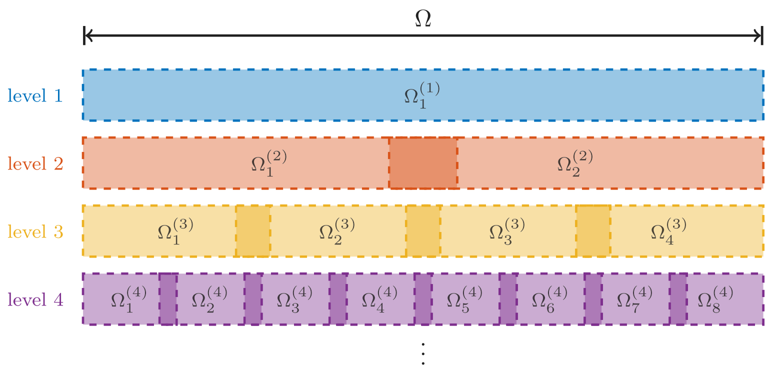

**Fig. 4.** Hierarchy of levels used in the multilevel FBPINN. For all the multilevel FBPINNs tested we use an exponential level structure. This means that the number of subdomains in each level is given by $2^{d(l-1)}$, where $l$ is the level number and $d$ is the dimensionality of the domain. Our hypothesis is that this helps the multilevel FBPINN model solutions with frequency components that span multiple orders of magnitude.

*Level structure.* Firstly, all multilevel FBPINNs use an exponentially increasing number of subdomains per level. In particular, we choose $J^{(l)} = 2^{d(l-1)}$ for $l = 1, \dots, L$. This level structure is shown in Fig. 4. This constraint is chosen so that the multilevel FBPINN is able to contain an exponentially large number of subdomains with a relatively small number of levels; our hypothesis is that this helps the multilevel FBPINN model solutions with frequency components that span multiple orders of magnitude.

*Domain decomposition.* All FBPINNs tested use a uniform rectangular DD for each level, with all multilevel FBPINNs having $2^{l-1}$ subdomains along each dimension. The size of each subdomain along each dimension is defined similar to Eq. (9), i.e., all 2D DDs look similar to those shown in Fig. 3(d) and (e). Furthermore, all FBPINNs use the same subdomain window functions, given by Eq. (10).

*Network architecture.* All FBPINNs tested use FCNs with identical architectures as their subdomain networks. The PINNs tested either use FCNs or FCNs with Fourier input features (see Appendix C for the definition of Fourier features). For all the FBPINNs tested, the $\mathbf{x}$ inputs to each subdomain network are normalized to the range $[-1, 1]$ along each dimension over their individual subdomains. For the PINNs tested, the $\mathbf{x}$ inputs are normalized to the range $[-1, 1]$ along each dimension over the global domain. tanh is used for all activation functions.

*Loss function and optimization.* All FBPINNs and PINNs tested use the hard-constrained variants of their loss functions. For fairness, the same constraining operator is used across all models tested for a given problem. Furthermore, the same collocation points are used for training whenever multiple models are compared on a given problem. This is similarly the case for all testing points used after training. The SA-PINNs tested use learnable weights for each collocation point in their loss function; this is described in more detail in Appendix D. All tests use the Adam optimizer with a learning rate of $1 \times 10^{-3}$ except for the PINNs with Fourier input features, which are trained using a learning rate of $1 \times 10^{-4}$, because it was found their convergence is unstable when using a larger learning rate. For robustness, all models are trained 10 times using different random starting seeds, and all results are reported as averages over these different seeds. All models are evaluated using the normalized L1 test loss, given by $\mathcal{L}(\theta) = \frac{1}{M} \sum_i^M \|u(\mathbf{x}_i, \theta) - u(\mathbf{x}_i)\| / \sigma$, where $M$ is the number of test points and $\sigma$ is the standard deviation of the set of true solutions $\{u(\mathbf{x}_i)\}_i^M$.

*Software and hardware implementation.* All FBPINNs and PINNs tested are implemented using a common training framework written in JAX [46]. Further details on our software implementation are given in Appendix A. All models are trained on a single NVIDIA RTX 3090 GPU.

## 3.4. Results

Here, we will discuss the results for the model problems described in Section 3.1.

### 3.4.1. Homogeneous Laplacian problem in two dimensions

First, we carry out simple ablation tests of the multilevel FBPINN using the 2D homogeneous Laplacian problem described in Section 3.4.1.

To carry out our ablation tests, we first train a baseline multilevel FBPINN to solve this problem, using $L = 3$ levels, an overlap ratio along each dimension of $\delta = 1.9$, and FCNs with 1 hidden layer and 16 hidden units for each subdomain network. The multilevel FBPINN is trained using the constraining operator

$$[\mathcal{C}u](\mathbf{x}, \theta) = \tanh(x_1/\sigma)\tanh((1 - x_1)/\sigma)\tanh(x_2/\sigma)\tanh((1 - x_2)/\sigma)u(\mathbf{x}, \theta)$$

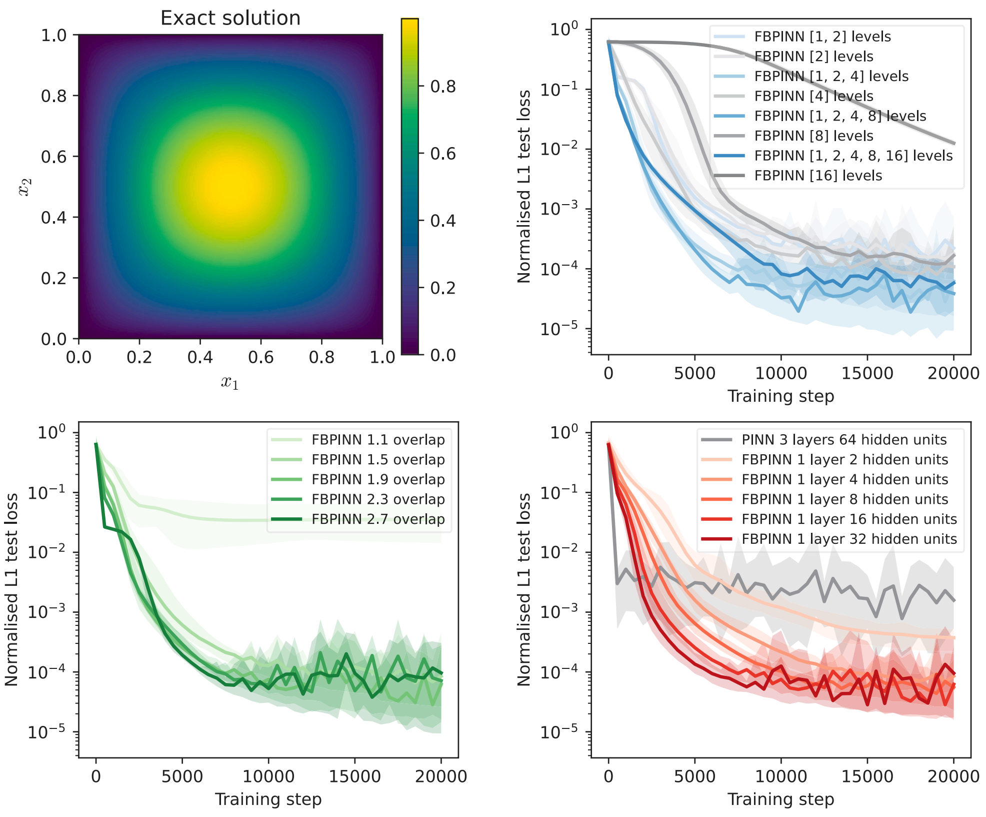

**Fig. 5.** Ablation tests using the homogeneous Laplacian problem. The convergence curve of a baseline multilevel FBPINN is plotted when changing the number of levels (top right), overlap ratio (bottom left), and number of hidden units for each subdomain network (bottom right). The baseline model has $L = 3$ levels, an overlap ratio of $\delta = 1.9$, and 16 hidden units for each subdomain network. The exact solution is shown (top left). Convergence curves of two other benchmarks are shown; a PINN (bottom right), and one-level FBPINNs with varying numbers of subdomains (top right). The lists which label each model in the top right plot contain the number of subdomains along each dimension for each level in the model. Filled region edges show the minimum and maximum loss values across 10 random starting seeds and lines show the average.

with $\sigma = 0.2$. Here, $N = 80 \times 80 = 6400$ uniformly-spaced collocation points and $M = 350 \times 350$ uniformly-spaced test points across the global domain are used to train and test the multilevel FBPINN, respectively.

Given this baseline model, we then vary different hyperparameters over a range of values and measure the change in performance. This is carried out for the number of levels ranging from $L = 2$ to 5, the overlap ratio ranging from 1.1 to 2.7, and the number of hidden units in the subdomain network ranging from 2 to 32. Our results are shown in Fig. 5. We observe that the accuracy of the multilevel FBPINN does not depend significantly on the number of levels, likely because in this case the solution is very simple. However, its accuracy increases as the overlap ratio increases, likely because there is more communication between the subdomain networks, which is similar to what is expected in classical DDMs. Furthermore, its accuracy increases as the number of free parameters of the subdomain networks increases. This is expected as the capacity of the model increases. Thus, the multilevel FBPINN has similar characteristics to classical DDMs for this problem.

We carry out two other benchmark tests. First, we train a PINN with 3 hidden layers and 64 hidden units, and second, we train four one-level FBPINNs with $J^{(1)} = 2, 4, 8$, and 16 subdomains along each dimension, respectively. All other relevant hyperparameters are kept the same as the baseline model. These results are also shown in Fig. 5. In these tests, the PINN is able to solve the problem, although its final accuracy is lower than the baseline multilevel FBPINN and its convergence curve is more unstable. Furthermore, the accuracy of the one-level FBPINN reduces as more subdomains are added. This is analogous to the expected behavior of one-level classical DDMs, which is not scalable to large numbers of subdomains, and shows that coarse levels are required for scalability. It is therefore likely that the additional levels in FBPINNs serve the same purpose as in classical DDMs, i.e., they allow direct transfer of global information.

### *3.4.2. Multi-scale Laplacian problem in two dimensions*

Next, we evaluate the strong and weak scalability of the multilevel FBPINN using the multi-scale Laplacian problem described in Section 3.4.2.

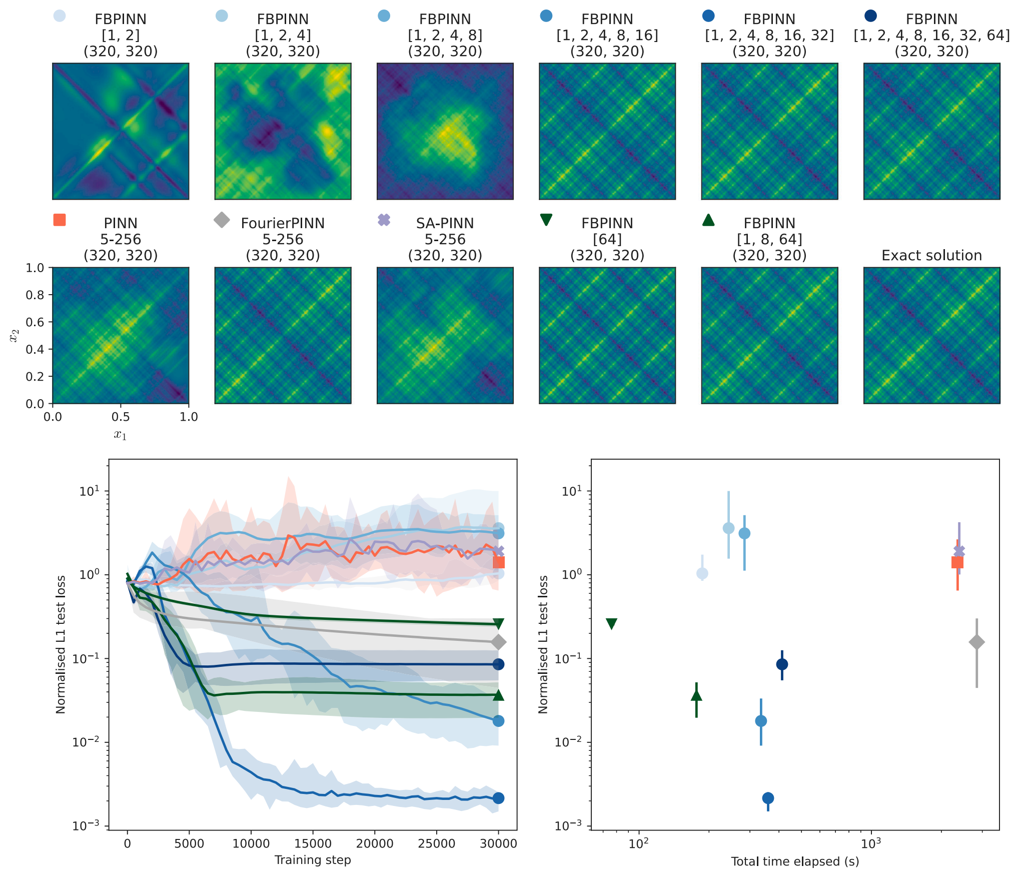

**Fig. 6.** Strong scaling test using the multi-scale Laplacian problem. In this test the problem complexity is fixed and the solution estimated using multilevel FBPINNs with increasing numbers of levels are plotted (top row). The title of each plot describes the level structure (first line) and the number of collocation points along each dimension (second line). The color-coded convergence curves and training times for each model are shown (bottom row). Filled region edges show the minimum and maximum loss values across 10 random starting seeds and lines show the average. Error bars show the minimum and maximum loss values and training times. The exact solution is shown (middle row). Plots of the solutions and convergence curves of a PINN, PINN with Fourier input features, SA-PINN, one-level FBPINN and three-level FBPINN benchmark are also shown (middle and bottom row).

*Strong scaling test.* First, we carry out a strong scaling test. Here, the problem complexity is fixed and we assess how the performance of the multilevel FBPINN changes as the capacity of the model is increased. In particular, we fix the problem complexity by choosing $n = 6$ with $\omega_i = 2^i$ for $i = 1, \dots, n$ in Eq. (14). Thus, the solution contains 6 multi-scale components with exponentially increasing frequencies. This represents a much more challenging problem than the homogeneous problem studied above. The exact solution in this case is shown in Fig. 6.

We then increase the capacity of the multilevel FBPINN by increasing the number of levels, testing from $L = 2$ to 7. The rest of the hyperparameters of the multilevel FBPINN are kept fixed across all tests. Namely, we use $N = 320\times 320 = 102{,}400$ uniformly-spaced collocation points throughout the domain, an overlap ratio of $\delta = 1.9$ and FCNs with 1 hidden layer and 16 hidden units for each subdomain network. All models are trained using the constraining operator $[\mathcal{C}u](\mathbf{x}, \boldsymbol{\theta}) = \tanh(x_1/\sigma)\tanh((1-x_1)/\sigma)\tanh(x_2/\sigma)\tanh((1-x_2)/\sigma)u(\mathbf{x}, \boldsymbol{\theta})$ with $\sigma = 1/\omega_n$. $M = 350 \times 350$ uniformly-spaced test points are used to test all models.

The results of this study are shown in Fig. 6. We find that the accuracy of the multilevel FBPINN increases as the number of levels increases, where the $L = 2, 3,$ and 4 models are unable to accurately model the solution, whilst the $L = 5, 6$ and 7 models are able to accurately model all of the frequency components. The test shows that the multilevel FBPINN is able to solve a high frequency, multi-scale problem, and exhibits strong scaling behavior somewhat analogous to what is expected by classical DDMs. However, we note that the accuracy of the 7-level FBPINN is worse than the 6-level FBPINN. We believe this may because at the finest level of the 7-level PINN, each subdomain network only contains approximately $10 \times 10$ collocation points. More collocation points may allow this level to converge more accurately.

Five other benchmark tests are carried out for this problem. First, we train a PINN with 5 hidden layers and 256 hidden units, a PINN with 256 Fourier input features with $\sigma = 5$, 5 hidden layers and 256 hidden units, and a SA-PINN with 5 hidden layers and 256 hidden units. Then, we train a one-level FBPINN with $J^{(1)} = 64$ subdomains along each dimension and a three-level FBPINN with $J^{(1)} = 1$, $J^{(2)} = 8$, and $J^{(3)} = 64$ subdomains along each dimension, respectively. All other relevant hyperparameters are kept the same as the baseline model above. These results are also shown in Fig. 6. We find that the accuracy of the standard PINN is

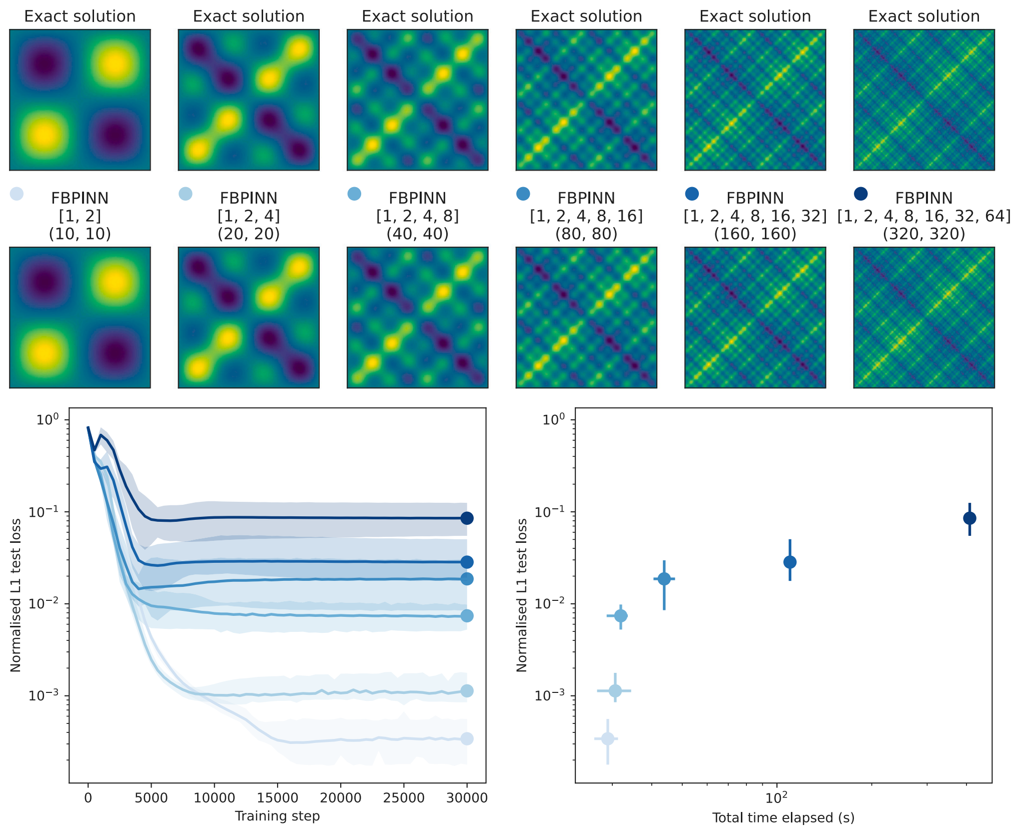

**Fig. 7.** Weak scaling test using the multi-scale Laplacian problem. In this test the problem complexity is increased (in this case, the number of frequency components in the solution) (top row) and the solution estimated using multilevel FBPINNs with increasing numbers of levels and collocation points are plotted (middle row). The title of each plot describes the level structure (first line) and the number of collocation points along each dimension (second line). The color-coded convergence curves and training times for each model are shown (bottom row). Filled region edges show the minimum and maximum loss values across 10 random starting seeds and lines show the average. Error bars show the minimum and maximum loss values and training times.

poor, and it is only able to model some of the cycles in the solution. Furthermore its convergence curve is very unstable, and its training time is an order of magnitude larger than the $L = 7$ level FBPINN tested. Its poor convergence is likely due to spectral bias and the increasing complexity of the PINN's optimization problem, as discussed in Section 2.1 and [22]. For this problem, adding Fourier features to the PINN significantly improves its accuracy, although it its training time remains high and it converges slower than the multilevel FBPINNs tested. The SA-PINN does not offer any improvement over the standard PINN. The one-level FBPINN is able to model the solution, although its accuracy is less than the $L = 7$ level FBPINN. Finally, the three-level FBPINN benchmark performs similarly to the $L = 7$ level FBPINN. This suggests that multilevel FBPINNs with stronger coarsening ratios can be used, and that multilevel FBPINNs are not strongly dependent on their coarsening ratio.

*Weak scaling test.* Next, we carry out a weak scaling test. Here, the problem complexity, number of collocation points and model capacity are scaled at the same rate, and we assess how the performance of the multilevel FBPINN changes. We increase the model capacity in the same way as the strong scaling test above, i.e., the number of levels is increased from $L = 2$ to 7. However, now the problem complexity is also scaled, such that for each test $n = L - 1$ and $\omega_i = 2^i$ for $i = 1, \dots, n$. Furthermore, each test has $(5 \times 2^{L-1}) \times (5 \times 2^{L-1})$ uniformly-spaced collocation points. Note that the number of subdomains, number of collocation points, and the frequency range of the solution all grow exponentially, and the multilevel FBPINN is in alignment with the problem structure. All other hyperparameters are fixed to the same values as the strong scaling test above.

The results of this test are shown in Fig. 7. We find that the multilevel FBPINNs are able to model all of the problems tested accurately, that is, modeling all of their frequency components. However, the normalized L1 accuracy of the multilevel FBPINNs does reduce somewhat as the problem complexity increases. Thus in this test the multilevel FBPINN exhibits near – but not perfect – weak scaling. As an additional test we plot the contribution to the FBPINN solution from each level for the 5-level FBPINN in Appendix E.

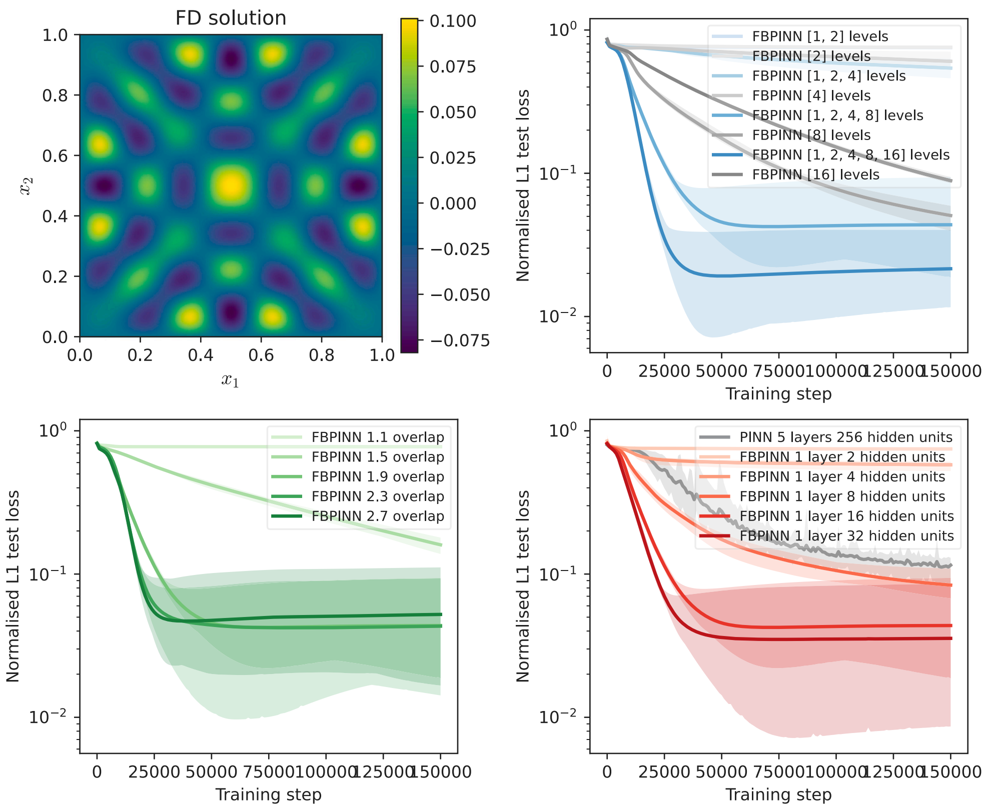

**Fig. 8.** Ablation tests using the Helmholtz problem. The convergence curve of a baseline multilevel FBPINN is plotted when changing the number of levels (top right), overlap ratio (bottom left), and number of hidden units for each subdomain network (bottom right). The baseline model has $L = 4$ levels, an overlap ratio of $\delta = 1.9$, and 16 hidden units for each subdomain network. The solution obtained from FD modeling is shown (top left). Convergence curves of two other benchmarks are shown; a PINN (bottom right), and one-level FBPINNs with varying numbers of subdomains (top right). The lists which label each model in the top right plot contain the number of subdomains along each dimension for each level in the model. Filled region edges show the minimum and maximum loss values across 10 random starting seeds and lines show the average.

### 3.4.3. *Helmholtz problem in two dimensions*

Finally, we test the multilevel FBPINN using the more complex Helmholtz problem described in Section 3.4.3. Again, we carry out ablation tests first and then carry out a weak scaling study assessing how the performance of the multilevel FBPINN changes as the wave number, $k$, increases.

*Ablation tests.* For our ablation tests, we fix the problem parameters to be $k = 2^4\pi/1.6$ and $\sigma = 0.8/2^4$ in Eq. (15). Then, similar to Section 3.4.1, we train a baseline multilevel FBPINN to solve this problem, using $L = 4$ levels, an overlap ratio along each dimension of $\delta = 1.9$, and FCNs with 1 hidden layer and 16 hidden units for each subdomain network. The multilevel FBPINN is trained using the constraining operator $[\mathcal{C}u](\mathbf{x}, \boldsymbol{\theta}) = \tanh(x_1/\sigma)\tanh((1-x_1)/\sigma)\tanh(x_2/\sigma)\tanh((1-x_2)/\sigma)u(\mathbf{x}, \boldsymbol{\theta})$ with $\sigma = 1/k$. We use $N = 160 \times 160 = 25{,}600$ uniformly-spaced collocation points and $M = 320 \times 320$ uniformly-spaced test points to train and test the multilevel FBPINN, respectively.

Given this baseline model, we then vary different hyperparameters over a range of values and measure the change in performance. This is carried out for the number of levels ranging from $L = 2$ to 5, the overlap ratio ranging from 1.1 to 2.7, and the number of hidden units in the subdomain network ranging from 2 to 32. Our results are shown in Fig. 8. We obtain similar results to the ablation tests carried out in Section 3.4.1 for the homogeneous Laplace problem. Namely, that the accuracy of the multilevel FBPINN improves as the overlap ratio and the number of free parameters of the subdomain networks increases. Furthermore, its accuracy improves as the number of levels increases, likely because the solution contains relatively high frequencies and multiple subdomains are needed. We observe relatively large variations of the test loss between different random initializations of the models below a value of $10^{-1}$. In particular, the final loss can be somewhere between $10^{-2}$ and $10^{-1}$.

We carry out two other benchmark tests. First, we train a PINN with 5 hidden layers and 256 hidden units, and second, we train four one-level FBPINNs with $J^{(1)} = 2, 4, 8,$ and 16 subdomains along each dimension, respectively. All other relevant hyperparameters are kept the same as the baseline model. These results are also shown in Fig. 8. Here, the PINN converges poorly, which again highlights the shortcomings of PINNs when solving more complex problems. Furthermore, the convergence of all the one-level FBPINNs is much slower than the multilevel FBPINN, and their final accuracy is worse. This again suggests that multiple levels are required for scalability.

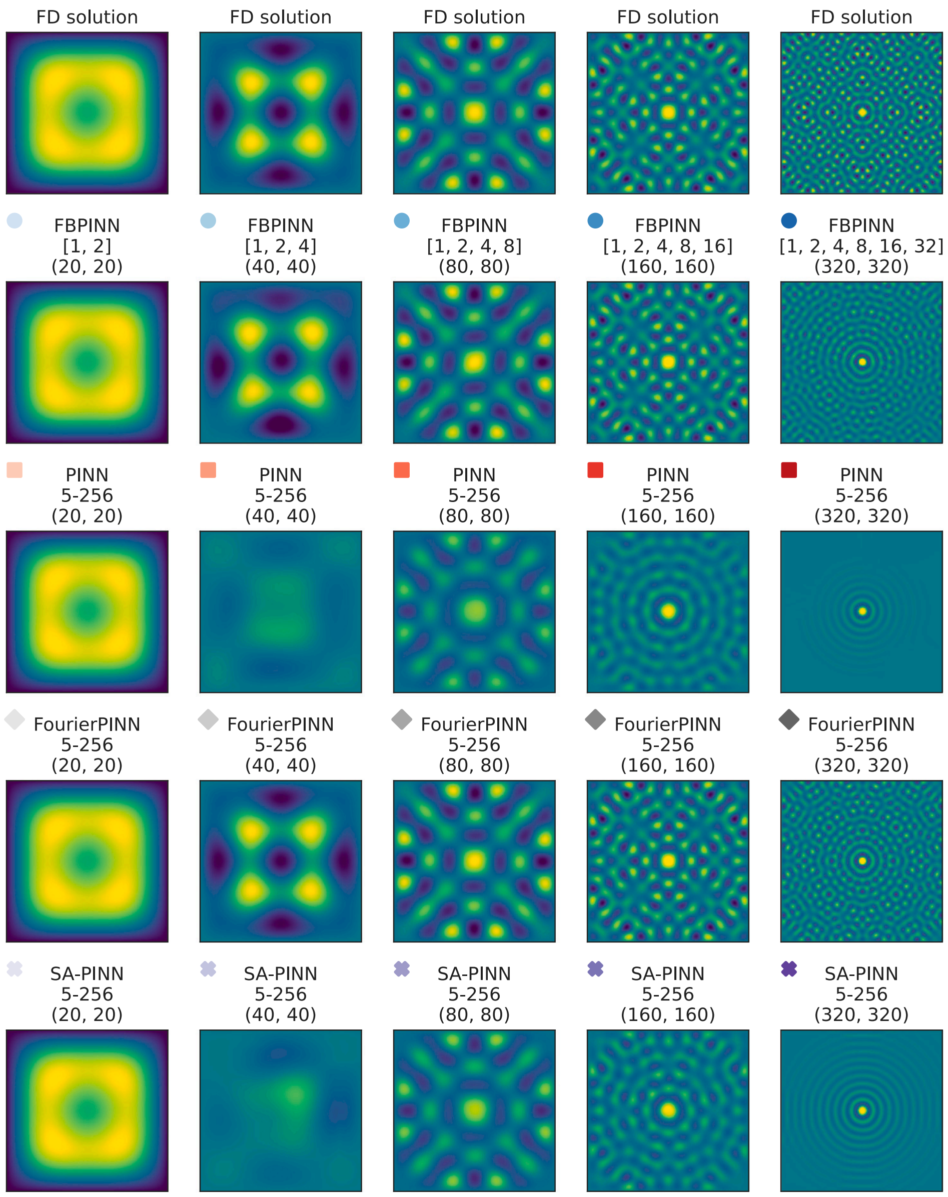

**Fig. 9.** Weak scaling test using the Helmholtz problem. In this test the problem complexity is increased (in this case, the wave number) (top row) and the solution estimated using multilevel FBPINNs with increasing numbers of levels and collocation points are plotted (second row). The title of each plot describes the level structure (first line) and the number of collocation points along each dimension (second line). Three benchmarks using a PINN, a PINN with Fourier input features, and a SA-PINN, all with a fixed network size and increasing numbers of collocation points are also shown (third and fourth row).

*Weak scaling test.* We carry out a weak scaling study, where both the problem complexity and model capacity are scaled at the same rate. In a similar fashion to the weak scaling test in Section 3.4.2, the capacity of the multilevel FBPINN is increased by increasing the number of levels, testing from $L = 2$ to 6. For each test, $(10 \times 2^{L-1}) \times (10 \times 2^{L-1})$ uniformly-spaced collocation points are used. The problem complexity for each test is increased by setting $k = 2^L\pi/1.6$ and $\sigma = 0.8/2^L$ in Eq. (15). All other hyperparameters are fixed to the same values as the baseline model used in the ablation tests above.

The results of this test are shown in Figs. 9 and 10. We find that the multilevel FBPINN is able to accurately model all the problems tested, except for the highest wave number test. In this case, the multilevel FBPINN successfully models the dominant frequency and overall concentricity of the solution but fails to model its more complex motifs. In this case, we believe that the FBPINN is struggling to satisfy both the point source and Dirichlet boundary conditions. Without the Dirichlet boundary condition, the solution to Eq. (15) is that of a simple point source. For all tests, we notice that in the first few training steps this is the solution learned by the multilevel FBPINN, which is then updated to the correct solution after further training. Thus, it appears the presence of the Dirichlet boundary condition leads to an optimization problem which remains challenging. This is consistent with challenges

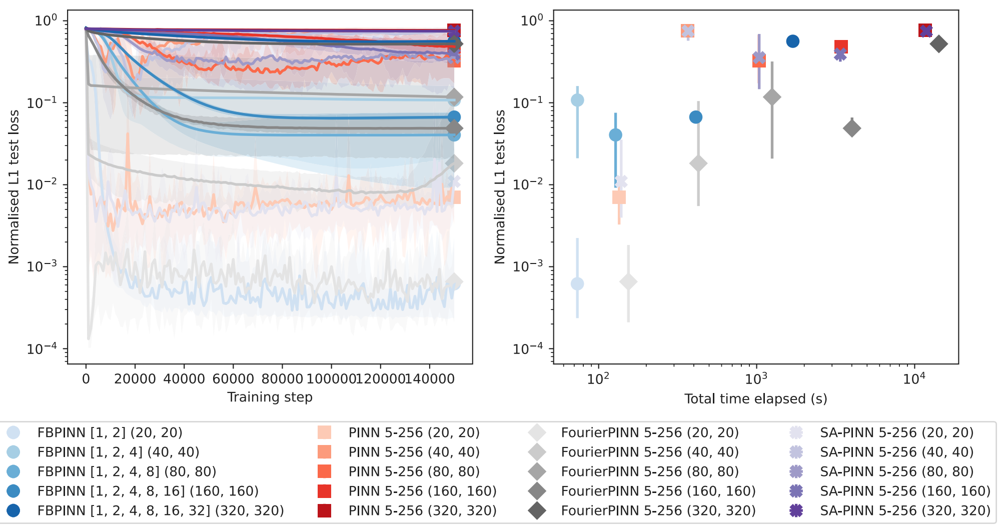

**Fig. 10.** Color-coded convergence curves and training times for each model displayed in Fig. 9. Filled region edges show the minimum and maximum loss values across 10 random starting seeds and lines show the average. Error bars show the minimum and maximum loss values and training times.

arising in solving Helmholtz problem using classical iterative numerical solvers. Further work is required to understand this behavior; one may be able to address this problem by using subdomain scheduling strategies to incrementally train the multilevel FBPINN, as proposed in [22].

Finally, we carry out the same weak scaling test but using a PINN instead of a multilevel FBPINN. For each test, the PINN's architecture is kept fixed at 5 hidden layers and 256 hidden units whilst the number of collocation points and problem complexity is increased in the same way as the previous test. We also test a PINN with 256 Fourier input features, 5 hidden layers and 256 hidden units, and a SA-PINN with 5 hidden layers and 256 hidden units. The $\sigma$ values of the Fourier input features are hand-selected for each test, and are $0.4, 1, 2, 1.5$, and $5$ in order of increasing problem complexity. All other relevant hyperparameters are kept the same as the FBPINNs tested. The result of this study is shown in Figs. 9 and 10. In this case, we find that the PINN and SA-PINN are unable to accurately model any of the solutions, and their training time is an order of magnitude larger than the multilevel FBPINN. The PINN with Fourier input features is able to model the solutions with a similar level of accuracy as the multilevel FBPINN, but its training time is an order of magnitude larger. Thus, multilevel FBPINNs still outperform PINNs for this problem.

## 4. Discussion

Across all the problems studied, we find that the multilevel FBPINNs consistently outperform the one-level FBPINNs and PINNs tested. The multilevel FBPINNs are more accurate than the one-level FBPINNs when a large number of subdomains are used, suggesting that coarse levels are required for scalability by improving the global communication. Furthermore, the multilevel FBPINNs are significantly more accurate and computationally efficient than the PINNs tested. However, we have only started to investigate multilevel FBPINNs in this work and there are many important research questions outstanding.

One important question would be to investigate the performance of multilevel FBPINNs on problems with complex geometries and solutions which have varying complexity in different parts of the domain. In this work, we restrict ourselves to problems with rectangular geometries and homogeneous solution complexity, and use multilevel FBPINNs with uniform rectangular decompositions and an exponential level structure. However, for problems with complex geometry and varying solution complexity, it is likely to be beneficial to use irregular DDs with irregular level structures and varying subdomain network sizes, to capture inhomogeneities in the solution. Whilst the FBPINN framework can represent such a model, implementing it is practically challenging for two reasons. First, to remain computationally efficient it is likely that a fully asynchronous training code across subdomains is required, because each subdomain network would require a different amount of computation to be evaluated and trained. Secondly, it may be challenging to choose an optimal DD, level structure and the subdomain network sizes, especially if characteristics of the solution are not known beforehand. One interesting direction here would be to try to learn the DD, for example by jointly learning the parameters of the FBPINN window functions with the subdomain network parameters or using a gating network similar to [52,53].

Another valuable direction would be to study the theoretical convergence properties of multilevel FBPINNs. A major limitation of PINNs compared to classical DDMs is that their convergence properties are still poorly understood. In particular, whilst the multilevel FBPINN exhibits good scaling properties for the Laplacian problems studied, it remains unclear why the optimization of the high wave number Helmholtz problem is challenging; note that the convergence of classical DDMs for high wave number Helmholtz problems is also not fully understood.

Furthermore, it is important to investigate ways to accelerate the computational efficiency of multilevel FBPINNs further. Whilst multilevel FBPINNs are over an order of magnitude more efficient than the PINNs tested, their training times are still likely to be slower than many traditional methods, such as numerical solvers for finite difference or finite element systems. Fundamentally, this is because (FB)PINNs yield a non-convex optimization problem, which is relatively expensive compared to the linear solves which traditional methods typically rely on. One way to accelerate (multilevel) FBPINNs, as suggested in [22], is to provide more inputs to the subdomain networks, such as BCs and PDE coefficients, and train across a range of these inputs so that the multilevel FBPINN learns a fast surrogate model which does not need to be retrained for each new solution.

Alongside this, multi-GPU training of (multilevel) FBPINNs should be investigated. Here we solve all problems using a single GPU, but multi-GPU training will become essential for problem sizes where $10,000+$ subdomains are required (for example, 3D problems, or problems with highly multi-scale solutions). In Section 2.1.2, we show that (multilevel) FBPINNs are theoretically scalable to large problem sizes: the computational cost of evaluating the FBPINN solution is $\mathcal{O}(NC\tilde{S})$, where $C$ is the average number of subdomains a collocation point belongs to, $\tilde{S}$ is the cost of computing the output of a single subdomain network for a single collocation point and $N$ is the number of collocation points. Importantly, this cost is independent of the total number of subdomains ($J$), and scales linearly with the number of collocation points. Practically, it may be challenging to achieve perfect linear scaling when using multiple GPUs because of the communication required between GPUs. For FBPINNs, the only communication required between subdomains is within their overlapping regions, where subdomain solutions are summed together; note that, in the multilevel case, subdomains on different levels may also overlap, therefore, increasing the required communication but improving the numerical scalability. One possible parallel implementation of FBPINNs was proposed by [22] (Algorithm 1 and Fig. 4), where a separate GPU is used to train each subdomain network. Importantly, communication between GPUs is only required in the forward pass when summing the outputs of the subdomain networks in the overlapping regions; the backpropagation and updating of the subdomain networks can then be done independently on each GPU for each subdomain network. Future work will investigate the parallel scalability of FBPINNs in detail.

## CRediT authorship contribution statement

**Victorita Dolean:** Writing – review & editing, Writing – original draft, Project administration, Methodology, Investigation, Conceptualization. **Alexander Heinlein:** Writing – review & editing, Writing – original draft, Project administration, Methodology, Investigation, Conceptualization. **Siddhartha Mishra:** Writing – review & editing, Writing – original draft, Resources, Project administration, Methodology, Investigation, Conceptualization. **Ben Moseley:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Data curation, Conceptualization.

## Declaration of competing interest

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

## Data availability

All the code for reproducing this work is available here: https://github.com/benmoseley/FBPINNs/tree/multilevel-paper/multilevel-paper. All data is generated synthetically using this code.

## Appendix A. Software implementation

All FBPINNs and PINNs are implemented using a common training framework written using the JAX automatic differentiation library [46]. When training (multilevel) FBPINNs, computing the FBPINN solution (either Eq. (5) or Eq. (7)) naively can be very expensive. This is because evaluating the solution at each collocation point involves summing over all subdomain networks and all levels. However, the cost of this summation can be significantly reduced by exploiting that, because the output of all subdomain networks is zero outside of the corresponding subdomains, only subdomains which contain each collocation point contribute to the summation. Practically, this can be carried out by pre-computing a mapping describing which subdomains contain each collocation point before training and only evaluating the corresponding subdomain networks during training. Another important efficiency gain in our software implementation is that the outputs of each subdomain network are computed in parallel on the GPU by using JAX's `vmap` functionality. This is important as the FBPINNs tested use small subdomain networks that if evaluated sequentially would not fully utilize the GPU's parallelism.

## Appendix B. Finite difference solver for Helmholtz equation

We use a finite difference (FD) solver to compute a reference solution for the Helmholtz problem studied in Section 3.4.3. For all the problem variants studied, we discretize the Laplacian operator in Eq. (15) using a 5-point stencil, and we discretize the solution using a $320 \times 320$ uniformly-spaced mesh over the problem domain. This turns Eq. (15) into a set of linear equations, which are solved using the `scipy.sparse.linalg` [61] sparse direct solver, that is, using UMFPACK [62].

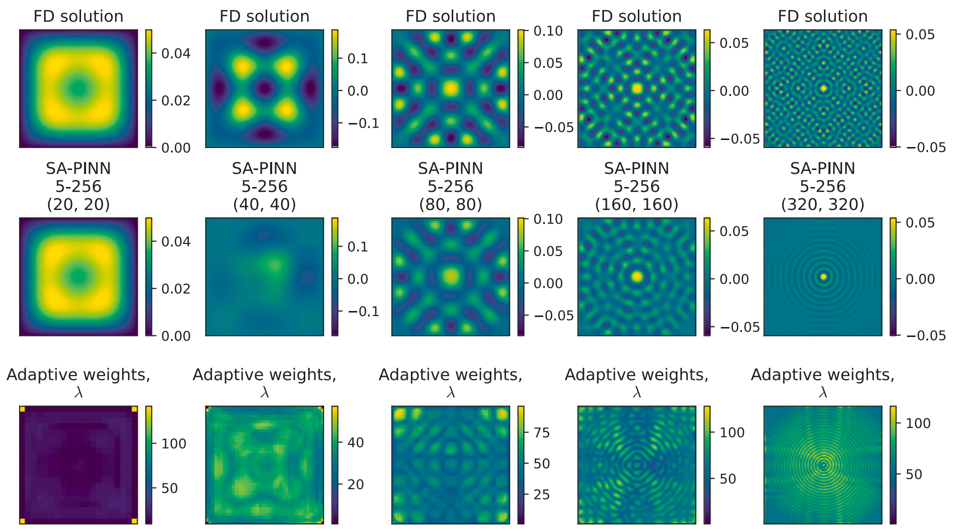

**Fig. D.11.** FD reference solution (top row), SA-PINN solution (middle row), and SA-PINN weights $\lambda$ (bottom row) obtained after training the SA-PINNs shown in Fig. 9.

## Appendix C. Definition of Fourier features

We compare the performance of FBPINNs to PINNs with Fourier features for a number of experiments. Fourier features can help neural networks learn high-frequency functions [28] and they have been shown to help the convergence of PINNs when solving multi-scale problems [23]. When using Fourier features, the inputs of neural networks are transformed using trigonometric functions before inputting them into the network. These Fourier features are given by

$$\gamma(x) = [\cos(2\pi\Gamma x), \sin(2\pi\Gamma x)] \ , \tag{C.1}$$

where $\Gamma$ is a matrix of shape $k \times d$, $k$ is the number of Fourier features and $x$ is the input vector $x \in \mathbb{R}^d$. The values of $\Gamma$ represent the frequency of the features and they are typically sampled from a univariate Gaussian distribution with a mean and standard deviation denoted by $\mu$ and $\sigma$. Similar to [23,28], we fix $\mu = 0$ and hand-select $\sigma$ based on test accuracy for all tests in this work. We find that the convergence of the PINN is highly sensitive to the value of $\sigma$ and the learning rate.

## Appendix D. Definition of self-adaptive PINNs (SA-PINNs)

We compare the performance of FBPINNs to self-adaptive SA-PINNs [29] for a number of experiments. SA-PINNs weight each collocation point in the PINN loss function separately, and learn these weights as the PINN trains by combining gradient descent steps on the PINN's parameters with gradient ascent steps on the weights. SA-PINNs have been shown to improve the performance of PINNs when solving multi-scale problems, including the Helmholtz equation [29]. More specifically, implementing the boundary conditions as hard constraints, SA-PINNs use the following loss function to train the PINN,

$$\mathcal{L}(\boldsymbol{\theta}, \lambda) = \frac{1}{N} \sum_{i=1}^{N} m(\lambda_i)(\mathcal{N}[\mathcal{C}u](\mathbf{x}_i, \boldsymbol{\theta}) - f(\mathbf{x}_i))^2 \ , \tag{D.1}$$

where $\{\lambda_i\}_{i=1}^N$ is a set of non-negative weights, with one weight for each collocation point, $m(\lambda_i)$ is a strictly increasing scalar function of $\lambda_i$, and the rest of the terms are as defined in Eq. (4). At each training step, gradient descent is used on Eq. (D.1) to update $\boldsymbol{\theta}$ and gradient ascent is used on Eq. (D.1) to update $\lambda$. This results in the values $\lambda_i$ increasing during training and placing greater weight on collocation points which have consistently high PDE residuals.

In this work, for all tests using SA-PINNs we fix $m(\lambda_i) = \lambda_i$, initialize all $\lambda_i = 1$ at the start of training, and use the same optimizer (Adam) and learning rate ($1 \times 10^{-3}$) to update $\boldsymbol{\theta}$ and $\lambda$. For reference, the weights $\lambda$ obtained after training the SA-PINNs used to solve the Helmholtz problems in Fig. 9 are shown in Fig. D.11.

## Appendix E. FBPINN individual level contributions

Fig. E.12 plots the contribution to the FBPINN solution from each level (the individual terms before summing across levels in Eq. (7)), for the 5-level FBPINN shown in Fig. 7. We find that the finest level roughly learns the highest frequencies in the solution, whilst the coarser levels learn the lower frequency contributions. We expect this behavior given the spectral bias of neural networks and the individual subdomain normalization employed in the FBPINN.

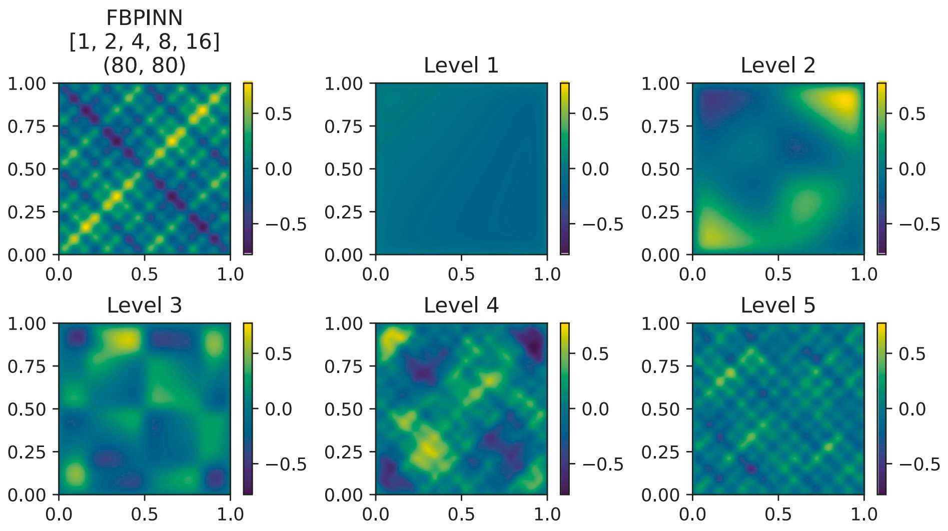

**Fig. E.12.** Contribution to the FBPINN solution from each level, for the 5-level FBPINN shown in Fig. 7. Top left shows the full FBPINN solution after summing the levels.

## References

[1] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, S. Lee, Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence, Tech. rep, USDOE Office of Science (SC) (United States), 2019, http://dx.doi.org/10.2172/1478744, URL http://www.osti.gov/servlets/purl/1478744/.

[2] J. Willard, X. Jia, S. Xu, M. Steinbach, V. Kumar, Integrating scientific knowledge with machine learning for engineering and environmental systems, ACM Comput. Surv. 55 (2022) http://dx.doi.org/10.1145/3514228, arXiv:2003.04919, URL https://dl.acm.org/doi/10.1145/3514228.

[3] S. Cuomo, V.S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, F. Piccialli, Scientific machine learning through physics–Informed neural networks: Where we are and what's next, J. Sci. Comput. 92 (3) (2022) 1–62, http://dx.doi.org/10.1007/s10915-022-01939-z, arXiv:2201.05624, URL https://link.springer.com/article/10.1007/s10915-022-01939-z.

[4] S. Arridge, P. Maass, O. Öktem, C.-B. Schönlieb, Solving inverse problems using data-driven models, Acta Numer. 28 (2019) 1–174, http://dx.doi.org/10.1017/s0962492919000059.

[5] B. Moseley, Physics-informed machine learning: from concepts to real-world applications (Ph.D. thesis), University of Oxford, 2022, http://dx.doi.org/10.13039/501100000266.

[6] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000, http://dx.doi.org/10.1109/72.712178, arXiv:9705023.

[7] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045.

[8] S. Mishra, R. Molinaro, Physics informed neural networks for simulating radiative transfer, J. Quant. Spectrosc. Radiat. Transfer 270 (2021).

[9] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. (2021) 1–19, http://dx.doi.org/10.1038/s42254-021-00314-5, URL https://www.nature.com/natrevphys.

[10] B. Moseley, A. Markham, T. Nissen-Meyer, Solving the wave equation with physics-informed deep learning, arxiv, 2020, arXiv:2006.11894, URL http://arxiv.org/abs/2006.11894.

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

[12] S. Cai, Z. Wang, F. Fuest, Y.J. Jeon, C. Gray, G.E. Karniadakis, Flow over an espresso cup: Inferring 3-D velocity and pressure fields from tomographic background oriented schlieren via physics-informed neural networks, J. Fluid Mech. 915 (2021) 102, http://dx.doi.org/10.1017/jfm.2021.135, arXiv:2103.02807.

[13] M. Raissi, A. Yazdani, G.E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030, http://dx.doi.org/10.1126/science.aaw4741.

[14] Y. Chen, L. Lu, G.E. Karniadakis, L. Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (8) (2020) 11618, http://dx.doi.org/10.1364/oe.384875, arXiv:1912.01085.

[15] L. Yang, X. Meng, G.E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, J. Comput. Phys. 425 (2021) 109913, http://dx.doi.org/10.1016/j.jcp.2020.109913.

[16] S. Wang, H. Wang, P. Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepONets, Sci. Adv. 7 (40) (2021) 8605–8634, http://dx.doi.org/10.1126/sciadv.abi8605, URL https://www.science.org/doi/full/10.1126/sciadv.abi8605, arXiv:2103.10974.

[17] Y. Zhu, N. Zabaras, P.S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81, http://dx.doi.org/10.1016/j.jcp.2019.05.024, arXiv:1901.06314.

[18] Z. Chen, Y. Liu, H. Sun, Physics-informed learning of governing equations from scarce data, Nature Commun. 12 (1) (2021) 1–13, http://dx.doi.org/10.1038/s41467-021-26434-1, URL https://www.nature.com/articles/s41467-021-26434-1, arXiv:2005.03448.

[19] S. Mishra, R. Molinaro, Estimates on the generalization error of physics-informed neural networks for approximating PDEs, IMA J. Numer. Anal. 00 (2022) 1–43, http://dx.doi.org/10.1093/imanum/drab093, URL https://academic.oup.com/imajna/advance-article/doi/10.1093/imanum/drab093/6503953, arXiv:2006.16144.

[20] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs, Commun. Comput. Phys. 28 (5) (2020) 2042–2074, http://dx.doi.org/10.4208/cicp.oa-2020-0193, arXiv:2004.01806.

[21] S. Wang, X. Yu, P. Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768, http://dx.doi.org/10.1016/j.jcp.2021.110768, arXiv:2007.14527.
[22] B. Moseley, A. Markham, T. Nissen-Meyer, Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, Adv. Comput. Math. 49 (4) (2023) 1–39, http://dx.doi.org/10.1007/S10444-023-10065-9, URL https://link.springer.com/article/10.1007/s10444-023-10065-9.
[23] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 384 (2021) 113938, http://dx.doi.org/10.1016/j.cma.2021.113938, arXiv:2012.10047.
[24] Z.Q.J. Xu, Y. Zhang, T. Luo, Y. Xiao, Z. Ma, Frequency principle: Fourier analysis sheds light on deep neural networks, Commun. Comput. Phys. 28 (5) (2020) 1746–1767, http://dx.doi.org/10.4208/CICP.OA-2020-0085, arXiv:1901.06523.
[25] N. Rahaman, A. Baratin, D. Arpit, F. Draxlcr, M. Lin, F.A. Hamprecht, Y. Bengio, A. Courville, On the spectral bias of neural networks, in: 36th International Conference on Machine Learning, ICML 2019, 2019-June, International Machine Learning Society (IMLS), 2019, pp. 9230–9239, arXiv:1806.08734.
[26] R. Basri, D. Jacobs, Y. Kasten, S. Kritchman, The convergence rate of neural networks for learned functions of different frequencies, in: Advances in Neural Information Processing Systems, 32, Neural information processing systems foundation, 2019, arXiv:1906.00425.
[27] Y. Cao, Z. Fang, Y. Wu, D.-X. Zhou, Q. Gu, Towards understanding the spectral bias of deep learning, IJCAI (2021) arXiv:1912.01198.
[28] M. Tancik, P.P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J.T. Barron, R. Ng, Fourier features let networks learn high frequency functions in low dimensional domains, in: Advances in Neural Information Processing Systems, vol. 2020-Decem, Neural information processing systems foundation, 2020, http://dx.doi.org/10.48550/arxiv.2006.10739, arXiv:2006.10739, URL https://arxiv.org/abs/2006.10739v1.
[29] L.D. McClenny, U.M. Braga-Neto, Self-adaptive physics-informed neural networks, J. Comput. Phys. 474 (2023) 111722, http://dx.doi.org/10.1016/j.jcp.2022.111722.
[30] A. Heinlein, A. Klawonn, M. Lanser, J. Weber, Machine learning in adaptive domain decomposition methods - predicting the geometric location of constraints, SIAM J. Sci. Comput. 41 (6) (2019) A3887–A3912.
[31] A. Heinlein, A. Klawonn, M. Lanser, J. Weber, Combining machine learning and domain decomposition methods for the solution of partial differential equations – a review, GAMM-Mitt. 44 (1) (2021) e202100001.
[32] A. Klawonn, M. Lanser, J. Weber, Machine learning and domain decomposition methods – a survey, 2023, arXiv:2312.14050.
[33] K. Li, K. Tang, T. Wu, Q. Liao, D3M: A deep domain decomposition method for partial differential equations, IEEE Access 8 (2019) 5283–5294.
[34] W. Li, X. Xiang, Y. Xu, Deep domain decomposition method: Elliptic problems, in: Mathematical and Scientific Machine Learning, PMLR, 2020, pp. 269–286.
[35] W. Li, Z. Wang, T. Cui, Y.X. null, X. Xiang, Deep domain decomposition methods: Helmholtz equation, Adv. Appl. Math. Mech. 15 (1) (2023) 118–138, http://dx.doi.org/10.4208/aamm.OA-2021-0305.
[36] V. Mercier, S. Gratton, P. Boudier, A coarse space acceleration of deep-DDM, 2021, http://dx.doi.org/10.48550/ARXIV.2112.03732.
[37] P.-L. Lions, On the Schwarz alternating method. I, in: First International Symposium on Domain Decomposition Methods for Partial Differential Equations, Paris, 1987, SIAM, Philadelphia, PA, 1988, pp. 1–42, 972510.
[38] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041, http://dx.doi.org/10.4208/CICP.OA-2020-0164.
[39] K. Lee, N.A. Trask, R.G. Patel, M.A. Gulian, E.C. Cyr, Partition of unity networks: AAAI 2021 spring symposium on combining artificial intelligence and machine learning with physical sciences, AAAI-MLPS 2021, in: CEUR Workshop Proceedings, vol. 2964, No. 180, 2021, URL http://www.scopus.com/inward/record.url?scp=85116654015&partnerID=8YFLogxK.
[40] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[41] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2017, http://dx.doi.org/10.48550/arXiv.1412.6980, arXiv:1412.6980[cs].
[42] D.C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Math. Program. 45 (1) (1989) 503–528, http://dx.doi.org/10.1007/BF01589116.
[43] H.J. Kelley, Gradient theory of optimal flight paths, ARS J. 30 (10) (1960) 947–954, http://dx.doi.org/10.2514/8.5282.
[44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous distributed systems, 2016, http://dx.doi.org/10.48550/arXiv.1603.04467.
[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019, URL https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.
[46] J. Bradbury, R. Frostig, P. Hawkins, M.J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, Q. Zhang, JAX: composable transformations of python+numpy programs, 2018, URL http://github.com/google/jax.
[47] L. Sun, H. Gao, S. Pan, J.X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Engrg. 361 (2020) http://dx.doi.org/10.1016/j.cma.2019.112732, arXiv:1906.02382.
[48] C. Leake, D. Mortari, Deep theory of functional connections: A new method for estimating the solutions of partial differential equations, Machine Learn. Knowl. Extract. 2 (1) (2020) 37–55, http://dx.doi.org/10.3390/make2010004, https://europepmc.org/article/pmc/pmc7259480.
[49] S. Dong, Z. Li, Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations, Comput. Methods Appl. Mech. Engrg. 387 (2021) 114129, http://dx.doi.org/10.1016/j.cma.2021.114129, arXiv:2012.02895.
[50] V. Dwivedi, N. Parashar, B. Srinivasan, Distributed learning machines for solving forward and inverse problems in partial differential equations, Neurocomputing 420 (2021) 299–316, http://dx.doi.org/10.1016/j.neucom.2020.09.006.
[51] G.B. Huang, Q.Y. Zhu, C.K. Siew, Extreme learning machine: Theory and applications, Neurocomputing 70 (1–3) (2006) 489–501, http://dx.doi.org/10.1016/j.neucom.2005.12.126.
[52] P. Stiller, F. Bethke, M. Böhme, R. Pausch, S. Torge, A. Debus, J. Vorberger, M. Bussmann, N. Hoffmann, Large-scale neural solvers for partial differential equations, in: J. Nichols, B. Verastegui, A.B. Maccabe, O. Hernandez, S. Parete-Koon, T. Ahearn (Eds.), Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI, Springer International Publishing, Cham, 2020, pp. 20–34.
[53] Z. Hu, A.D. Jagtap, G.E. Karniadakis, K. Kawaguchi, Augmented Physics-Informed Neural Networks (APINNs): A gating network-based soft domain decomposition methodology, Eng. Appl. Artif. Intell. 126 (2022) 107183, http://dx.doi.org/10.1016/j.engappai.2023.107183, arXiv:2211.08939.
[54] A. Toselli, O. Widlund, Domain decomposition methods—algorithms and theory, in: Springer Series in Computational Mathematics, vol. 34, Springer-Verlag, Berlin, 2005, http://dx.doi.org/10.1007/b137868, 2104179.
[55] V. Dolean, P. Jolivet, F. Nataf, An introduction to domain decomposition methods, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015, Algorithms, theory, and parallel implementation.
[56] V. Dolean, A. Heinlein, S. Mishra, B. Moseley, Finite basis physics-informed neural networks as a Schwarz domain decomposition method, 2022, http://dx.doi.org/10.48550/ARXIV.2211.05560.
[57] B.F. Smith, P.E. Bjørstad, W.D. Gropp, Domain decomposition, Cambridge University Press, Cambridge, 1996, 1410757.

[58] X.-C. Cai, M. Sarkis, A restricted additive Schwarz preconditioner for general sparse linear systems, SIAM J. Sci. Comput. 21 (2) (1999) 792–797, http://dx.doi.org/10.1137/S106482759732678X, 1718707.

[59] X.-C. Cai, D.E. Keyes, Nonlinearly preconditioned inexact Newton algorithms, SIAM J. Sci. Comput. 24 (1) (2002) 183–200, http://dx.doi.org/10.1137/S106482750037620X, 1924420.

[60] V. Dolean, M.J. Gander, W. Kheriji, F. Kwok, R. Masson, Nonlinear preconditioning: How to use a nonlinear Schwarz method to precondition Newton's method, SIAM J. Sci. Comput. 38 (6) (2016) A3357–A3380, http://dx.doi.org/10.1137/15M102887X.

[61] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental algorithms for scientific computing in python, Nat. Methods 17 (2020) 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.

[62] T.A. Davis, UMFPACK-an unsymmetric-pattern multifrontal method with a column pre-ordering strategy, ACM Trans. Math. Software 30 (204) (2004) 196–199.