Advances in Computational Mathematics (2023) 49:62
https://doi.org/10.1007/s10444-023-10065-9

Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations

Ben Moseley[1] · Andrew Markham[2] · Tarje Nissen-Meyer[3]

Received: 23 June 2022 / Accepted: 28 June 2023 / Published online: 31 July 2023

Abstract

Recently, physics-informed neural networks (PINNs) have offered a powerful new paradigm for solving problems relating to differential equations. Compared to classical numerical methods, PINNs have several advantages, for example their ability to provide mesh-free solutions of differential equations and their ability to carry out forward and inverse modelling within the same optimisation problem. Whilst promising, a key limitation to date is that PINNs have struggled to accurately and efficiently solve problems with large domains and/or multi-scale solutions, which is crucial for their real-world application. Multiple significant and related factors contribute to this issue, including the increasing complexity of the underlying PINN optimisation problem as the problem size grows and the spectral bias of neural networks. In this work, we propose a new, scalable approach for solving large problems relating to differential equations called *finite basis physics-informed neural networks (FBPINNs)*. FBPINNs are inspired by classical finite element methods, where the solution of the differential equation is expressed as the sum of a finite set of basis functions with compact support. In FBPINNs, neural networks are used to learn these basis functions, which are defined over small, overlapping subdomains. FBINNs are designed to address the spectral bias of neural networks by using separate input normalisation over each subdomain and reduce the complexity of the underlying optimisation problem by using many smaller neural networks in a parallel divide-and-conquer approach. Our numerical experiments show that FBPINNs are effective in solving both small and larger, multi-scale problems, outperforming standard PINNs in both accuracy and computational resources required, potentially paving the way to the application of PINNs on large, real-world problems.

Communicated by: Siddhartha Mishra

✉ Ben Moseley
benjamin.moseley@ai.ethz.ch

[1] AI Center, ETH Zürich, Zürich, Switzerland

[2] Department of Computer Science, University of Oxford, Oxford, UK

[3] Department of Earth Sciences, University of Oxford, Oxford, UK

Keywords Physics-informed neural networks · Domain decomposition · Multi-scale modelling · Forward modelling · Differential equations · Parallel computing

Mathematics Subject Classification (2010) 65M99 · 68T01

1 Introduction

Solving forward and inverse problems relating to differential equations has been the subject of intense research focus over many decades. Indeed, many areas of science rely on our ability to accurately solve these problems, from modelling the climate to understanding fundamental particle physics [1–3]. Classical approaches such as finite difference, finite element, and spectral methods have emerged as the dominant approach, with great advances in their performance being made throughout the decades. Today, these methods are capable of solving highly complex, large-scale problems and are often implemented using sophisticated, adaptive and parallel algorithms utilising thousands of CPU and GPU cores [4, 5].

Whilst powerful, classical methods also have some long-standing limitations. For example, many of today's applications require the study of multi-physics, multi-scale systems, which can be very challenging to incorporate and model accurately, often requiring the use of adaptive schemes and subgrid parameterisations [6, 7]. Another is the vast computational resources that are typically required, which render many real-world applications unfeasible. Thirdly, implementations of classical approaches are often elaborate and can require millions of lines of code, making them challenging to maintain from generation to generation [4].

In recent years, researchers have turned towards the rapid advances in machine learning as a potential way to address some of these challenges. One strategy is to entirely replace classical methods with purely data-driven algorithms (for example see [8, 9]), and whilst this can lead to advantages such as dramatic savings in computational efficiency, it reveals important flaws in current machine learning techniques such as issues with generalisation and the need for large amounts of training data. A more recent strategy is to blend together physical principles and traditional algorithms with machine learning in a more nuanced way to create more powerful models, in the burgeoning field of scientific machine learning (SciML) [10, 11].

One such approach which has received significant attention are *physics-informed neural networks* (PINNs) [12–15], which can be used to solve both forward and inverse problems relating to differential equations. PINNs use a deep neural network to represent the solution of a differential equation, which is trained using a loss function which directly penalises the residual of the underlying equation. PINNs have multiple benefits in comparison to classical methods, for example they provide approximate mesh-free solutions which have tractable analytical gradients, and they provide an elegant way to carry out joint forward and inverse modelling within the same optimisation problem [13]. Many of the fundamental concepts behind PINNs were proposed in the 1990s by [12] and others, whilst [13] implemented and extended them with modern deep learning techniques.

Since these works, PINNs have been utilised in a wide range of applications, for example in simulating fluid flow [16, 17], carrying out cardiac activation mapping [18], modelling wave physics [19], and even in predicting the temperature distribution over an espresso cup [20]. Many extensions of PINNs have also been investigated. For example, one extension is to include uncertainty estimation within PINNs, and [21] proposed Bayesian PINNs which use Bayesian neural networks to estimate uncertainty in the PINN solution. Another is to use the variational form of differential equations to construct an alternative optimisation problem (see, for example [22]). [23], [24] and [25] instead used finite difference filters to construct a differential equation residual in a PINN-like loss function when training CNNs and RNNs to approximate the solution of the differential equation on a mesh. The advantage of this approach is that these models can be easily conditioned on the initial conditions of the problem, i.e. they allow surrogate models to be learned. [26] proposed DeepONets, which also condition neural networks on their initial conditions such that they learn families of solutions, and DeepONets can be extended to include PINN loss functions [27]. In other applications, PINNs have been extended to fractional differential equations [28], stochastic differential equations [29, 30], and have even been used to discover underlying differential equations themselves [31]. Multiple software libraries have been developed which allow PINNs to be trained easily and quickly, such as DeepXDE [32], SimNet [33], PyDEns [34] and NeuroDiffEq [35].

Whilst popular and effective, PINNs also suffer from some significant limitations. One is that, in comparison to classical approaches, the theoretical convergence properties of PINNs are still poorly understood. For example, the PINN loss function can be highly non-convex and result in a stiff optimisation problem, yet it is unclear which classes of problems this affects [36, 37]. Work by [37–39] and others have made initial steps towards understanding the theoretical properties of PINNs but this sub-field is in its early stages. Another is the poor computational efficiency of PINNs. A standard PINN must be retrained for each solution, which is expensive and typically means classical methods strongly outperform PINNs, at least for forward modelling tasks. Works which condition PINNs on their initial conditions so that they learn a family of solutions, such as those aforementioned above [23–25, 27], could potentially address this issue.

Finally, a major challenge is scaling PINNs to large problem domains. There are multiple related issues in this respect. One is that, as the domain size increases, the complexity of the solution typically increases, which inevitably requires the size of the neural network (or number of free parameters) to increase such that the network is expressive enough to represent it. This in turn results in a harder PINN optimisation problem, both in terms of the number of free parameters and the increased number of training points required to sufficiently sample the solution over the larger domain, and typically leads to much slower convergence times. Another is the *spectral bias* of neural networks. This is the well-studied observation that neural networks tend to learn higher frequencies much more slowly than lower frequencies [40–43], with various convergence rates being proved. Because a problem's input variables are typically normalised over the domain before being input to a PINN, low-frequency features in the solution effectively become high-frequency features as the domain size increases, which can severely hinder the convergence of PINNs. Similarly, spectral bias is a major issue when solving problems with multi-scale solutions [44]. These limitations have meant that the vast majority of PINN applications to date have only solved problems in small domains with limited frequency content.

Recent works have started to investigate these issues. [44] showed that spectral bias also affects PINNs and proposed the use of Fourier input features [45] to help alleviate this issue, which transform the input variables over multiple scales using trigonometric functions as a pre-processing step. [46] proposed multi-scale deep neural networks, which used radial scaling of the input variables in the frequency domain to achieve uniform convergence across multiple scales when solving problems with a PINN loss. However, for both of these methods, the scale of their input features must be chosen to match the frequency range of interest.

For reducing the complexity of the PINN optimisation problem, an increasingly popular approach is to use *domain decomposition* [47], taking inspiration from existing classical methods such as finite element modelling. Instead of solving one large PINN optimisation problem, the idea is to use a "divide and conquer" approach and train many smaller problems in parallel. This can reduce training times and could potentially reduce the difficulty of the global optimisation problem too. For example, [48] proposed XPINNs, which divide the problem domain into many subdomains and use separate neural networks in each subdomain to learn the solution. A key consideration in any domain decomposition approach is to ensure that individual subdomain solutions are communicated and match across the subdomain interfaces, and [48] rely upon additional interfaces terms in their PINN loss function to do so. In follow-up work, [49] showed their approach can be parallelised to multiple GPUs, decreasing training times. However, a key downside of their approach is that it contains discontinuities in the PINN solution across subdomain interfaces, as the interface conditions are only weakly constrained in the loss function. Similar domain decomposition strategies were proposed by [50] and [51], but who instead used extreme learning machines (ELMs) [52] as the neural networks in each subdomain, which can be rapidly trained. However, whilst [51] claimed computational efficiency comparable to finite element modelling, ELMs restrict the capacity of neural networks as only the weights in the last layer can be updated. In other related approaches, [53] replaced subdomain solvers of the classical Schwarz domain decomposition approach with PINNs, resulting in a similar strategy to [48]. [54] proposed GatedPINNs, which use an auxiliary neural network to learn the domain decomposition itself, and [55] use domain decomposition to train variational PINNs, although only when defining the space of test functions.

In this work, we propose a new domain decomposition approach for solving large, multi-scale problems relating to differential equations called *finite basis physics-informed neural networks (FBPINNs)*. In contrast to the existing domain decomposition approaches, we ensure that interface continuity is strictly enforced across the subdomain boundaries by the mathematical construction of our PINN solution ansatz, which removes the need for additional interface terms in our PINN loss function. Furthermore, we explicitly consider the effects of spectral bias by using separate input variable normalisation within each subdomain, which restricts the effective solution frequency each subdomain network sees. We further propose flexible training schedules which can aid the convergence of FBPINNs and propose a parallel training algorithm which allows FBPINNs to be scaled computationally. To numerically validate our approach, we compare the accuracy of FBPINNs to standard PINNs across a range of problem sizes, both in terms of domain size and dimensionality, including problems with multi-scale solutions. These experiments suggest that FBPINNs are effective in solving both small and larger, multi-scale problems, outperforming standard PINNs in both accuracy and computation resources required. All of the code for reproducing the results of this paper can be found at github.com/benmoseley/FBPINNs.

The remainder of this paper is as follows. In Section 2, we describe the standard PINN implementation as defined by [13], as well as the alternative constructed ansatz formulation proposed by [12]. In Section 3, we motivate FBPINNs by showing a simple example where PINNs break down: that of learning the solution to $\frac{du}{dx} = \cos \omega x$ for large values of ω. In Section 4, we present the FBPINN methodology in detail. In Section 5, we present numerical results which compare the performance PINNs and FBPINNs across many smaller and larger scale problems, including the Burgers equation and the wave equation. Finally, in Sections 6 and 7, we discuss the implications and conclusions of our work.

2 Physics-informed neural networks (PINNs)

PINNs are designed to solve differential equations of the general form [13]

$$\begin{aligned} \mathcal{D}[u(x);\lambda] &= f(x)\ , \quad x \in \Omega\ , \\ \mathcal{B}_k[u(x)] &= g_k(x)\ , \quad x \in \Gamma_k \subset \partial\Omega\ , \end{aligned} \tag{1}$$

for $k = 1, 2, ..., n_b$ where $\mathcal{D}$ is a differential operator, $\mathcal{B}_k$ is a set of boundary operators, $u \in \mathbb{R}^{d_u}$ is the solution to the differential equation, $f(x)$ is a forcing function, $g_k(x)$ is a set of boundary functions, x is an input vector in the domain $\Omega \subset \mathbb{R}^d$ (i.e. x is a d-dimensional vector), $\partial\Omega$ denotes the boundary of Ω and λ is an optional vector of additional parameters of the differential operator. Many different physical systems can be described in this form, including problems with time-dependence or time-independence, linear or nonlinear differential operators, and different types of initial and boundary conditions such as Dirichlet and Neumann conditions. As an example, the inhomogeneous wave equation with Dirichlet and Neumann boundary conditions at $t = 0$ reads

$$\begin{aligned} \left[\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\right] u(x,t) &= f(x,t)\ , \\ u(x,0) &= g_1(x)\ , \\ \frac{\partial u}{\partial t}(x,0) &= g_2(x)\ , \end{aligned} \tag{2}$$

where c is the wave speed (note, for all time-dependent problems described in this paper, we have labelled the time dimension separately from the spatial dimensions of the input vector for readability).

Given the problem definition in Equation 1, PINNs train a neural network, denoted as $NN(x;\theta)$ where θ denotes the free parameters of the neural network, to directly approximate the solution to the differential equation, i.e. $NN(x;\theta) \approx u(x)$. To solve the problem, the network is trained by minimising the following loss function:

$$\mathcal{L}(\theta) = \mathcal{L}_p(\theta) + \mathcal{L}_b(\theta) \ , \tag{3}$$

where

$$\mathcal{L}_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \| \ \mathcal{D}[NN(x_i;\theta);\lambda] - f(x_i) \ \|^2 \ , \tag{4}$$

$$\mathcal{L}_b(\theta) = \sum_k \frac{1}{N_{bk}} \sum_j^{N_{bk}} \| \ \mathcal{B}_k[NN(x_{kj};\theta)] - g_k(x_{kj}) \ \|^2 \ , \tag{5}$$

$\{x_i\}$ is a set of training points sampled over the full domain Ω, and $\{x_{kj}\}$ is a set of training points sampled from the boundary Γ_k associated with each boundary condition. We denote the first term, $\mathcal{L}_p(\theta)$, as the "physics loss" and the second term, $\mathcal{L}_b(\theta)$, as the "boundary loss". Intuitively, the physics loss pushes the neural network to learn solutions which are consistent with the underlying differential equation, whilst the boundary loss attempts to ensure the solution is unique by matching the solution to the boundary conditions. For the wave equation example above, Equation 3 becomes

$$\begin{aligned}\mathcal{L}(\theta) = & \frac{1}{N_p} \sum_i^{N_p} \| \left[\nabla^2 - \frac{1}{c^2}\frac{\partial^2}{\partial t^2} \right] NN(x_i,t_i;\theta) - f(x_i,t_i) \ \|^2 \\ & + \frac{1}{N_{b1}} \sum_j^{N_{b1}} \| \ NN(x_{1j},0;\theta) - g_1(x_{1j}) \ \|^2 + \frac{1}{N_{b2}} \sum_j^{N_{b2}} \| \ \frac{\partial}{\partial t} NN(x_{2j},0;\theta) \\ & - g_2(x_{2j}) \ \|^2 . \end{aligned} \tag{6}$$

There are a number of important points to consider when training PINNs. The first is that a sufficient number of training points $\{x_i\}$ and $\{x_{kj}\}$ should be selected such that the network is able to learn a consistent solution across the entire domain. For large domains, training points are often sampled in mini-batches and the optimisation problem becomes a stochastic optimisation problem.

Secondly, when evaluating the loss function, the gradients of the neural network with respect to its inputs are required. These are typically analytically available, and this allows the loss function to be evaluated and further differentiated with respect to θ, allowing gradient descent methods to be used to optimise θ. In practice, these gradients are easily obtainable in modern deep learning packages through the use of autodifferentiation (AD). For example, when using the `PyTorch` library [56], one can use reverse or forward-mode AD to obtain a computational graph for computing the gradients of a network's output with respect to its inputs and then use AD again on this graph to obtain gradients of the PINN loss function with respect to the parameters of the network.

Thirdly, it is important to note that whilst the known values of the solution (and/or its derivatives) are required at the domain boundaries to evaluate the boundary loss, evaluating the physics loss only requires samples of the input vector, i.e. it can be viewed as an unsupervised regularisation term, and therefore, PINNs require very little training data in this sense.

Finally, PINNs can be naturally extended to inverse problems, for example when estimating the parameters of the differential equation λ. In this setting, an additional "data loss" is usually added, which penalises the difference between the network solution and the solution at a set of observed points within the domain, and the parameters λ are jointly optimised alongside θ.

2.1 Weakly vs strongly constrained PINNs

A downside of the standard PINN approach described above is that the boundary conditions are only weakly enforced in the loss function described by Equation 3, meaning that the learned solution may be inconsistent. Furthermore, recent work has shown theoretically and empirically that the PINN optimisation problem can be stiff to solve, to the point where it may not converge at all, due to the two terms in the loss function competing with each other [17, 36, 37]. An alternative approach, as originally proposed by [12], is to strictly enforce the boundary conditions by using the neural network as part of a solution ansatz. For example, for the wave equation problem above, instead of defining a neural network to directly approximate the solution $NN(x, t; \theta) \approx u(x, t)$, one could instead use the ansatz

$$\hat{u}(x, t; \theta) = g_1(x) + t\, g_2(x) + t^2\, NN(x, t; \theta)\ , \tag{7}$$

to define the approximate solution in the PINN optimisation problem. It is straightforward to verify that this ansatz automatically satisfies the boundary conditions in Equation 2. Because the boundary conditions are automatically satisfied, only the physics loss, $\mathcal{L}_p(\theta)$, needs to be included in Equation 3, which turns the optimisation problem from a constrained one into a simpler unconstrained one. This formulation has since been extended to irregular domains [57, 58], and general schemes for constructing suitable ansatze have been proposed [59]. We will use this strategy for the rest of the examples in the paper, although we note that the FBPINN framework is not limited to this approach and can use both formulations.

3 A motivating example

As discussed in Section 1, a major challenge is to scale PINNs to large domains. In this section, we will show a simple empirical 1D example highlighting this difficulty, which motivates the FBPINN framework. Specifically, we consider the following problem:

$$\begin{aligned} \frac{du}{dx} &= \cos(\omega x)\ , \\ u(0) &= 0\ , \end{aligned} \tag{8}$$

where $x, u, \omega \in \mathbb{R}^1$, which has the exact solution

$$u(x) = \frac{1}{\omega} \sin(\omega x) \ . \tag{9}$$

We will use the PINN solution ansatz

$$\hat{u}(x; \theta) = \tanh(\omega x) NN(x; \theta) \ , \tag{10}$$

to solve this problem. We choose to use the tanh function in the ansatz because away from the boundary condition, its value tends to ± 1, such that the neural network does not need to learn to dramatically compensate for this function away from the boundary. We also approximately match the width of the tanh function to the wavelength of the exact solution, again such that the compensation the network needs to learn in the vicinity of the boundary is of similar frequency to the solution. In our subsequent experiments, we find both of these strategies help PINNs converge over large domains and different solution frequencies.

The PINN is trained using the unconstrained loss function

$$\mathcal{L}(\theta) = \mathcal{L}_p(\theta) = \frac{1}{N_p} \sum_{i}^{N_p} \| \frac{d}{dx} \hat{u}(x_i; \theta) - \cos(\omega x_i) \|^2 \ , \tag{11}$$

as derived using Equation 4.

3.1 Low-frequency case ($\omega = 1$)

First, the above PINN is trained for the $\omega = 1$ case. We use a fully connected network with 2 hidden layers, 16 hidden units per layer and tanh activation functions, using 200 training points ($\{x_i\}$) regularly spaced over the domain $x \in [-2\pi, 2\pi]$. The input variable x is normalised to $[-1, 1]$ over the domain before being input to the network, and the output of the network is unnormalised by multiplying it by $\frac{1}{\omega}$ before being used in the ansatz defined by Equation 10. The PINN is trained using gradient descent with the Adam optimiser [60] and a learning rate of 0.001. All code is implemented using the `PyTorch` library, using its autodifferentiation features [56].

Figure 1 (a) and (e) shows the PINN solution after training and its L1 error compared to the exact solution (evaluated using 1000 regularly spaced points in the domain) as a function of training step. We find that the PINN is able to quickly and accurately converge to the exact solution.

3.2 High-frequency case ($\omega = 15$)

Next, the same experiment is run, but with two changes; we increase the frequency of the solution to $\omega = 15$ and the number of regularly spaced training points to $200 \times 15 = 3000$. Figure 1 (b) and (e) shows the resulting PINN solution and L1 convergence curve (using 5000 test points). We find for this case the PINN is unable

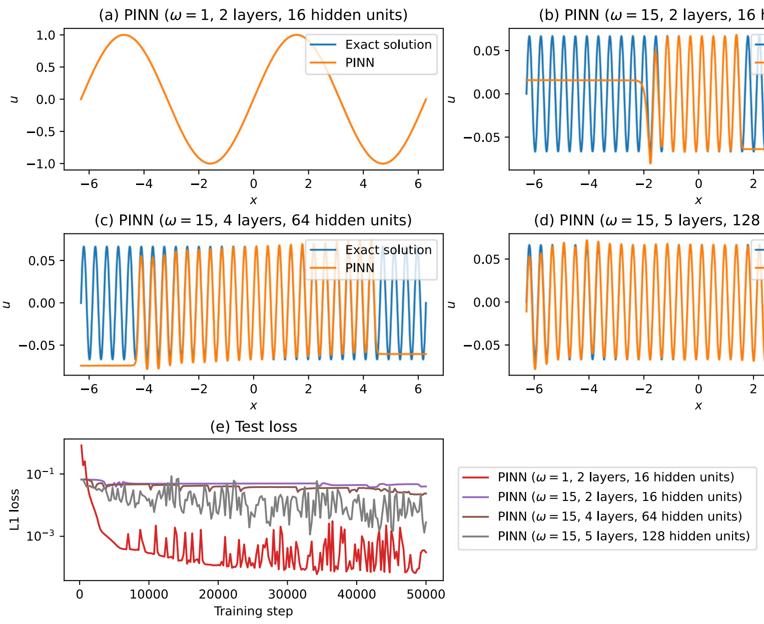

Fig. 1 Using PINNs to solve $\frac{du}{dx} = \cos(\omega x)$. When $\omega = 1$, a PINN with 2 hidden layers and 16 hidden units quickly converges to the solution: (a) shows the PINN solution compared to the exact solution, and (e) shows the L1 error between the PINN solution and the exact solution against training step. When $\omega = 15$, the same PINN struggles to converge, as shown in (b). Increasing the number of free parameters of the PINN improves its accuracy, as shown in (c) and (d)

to accurately learn the solution, only being able to capture the first few cycles of the solution away from the boundary condition. We further retrain the PINN using larger network sizes of 4 layers and 64 hidden units and 5 layers and 128 hidden units, shown in Fig. 1 (c) and (d), and find that only the PINN with 5 layers and 128 hidden units is able to fully model all of the cycles, although its final relative L1 error is much worse than the $\omega = 1$ case and its convergence curve is much slower and more unstable. We note that the $\omega = 1$ PINN with 2 layers and 16 hidden units uses 321 free parameters whilst the $\omega = 15$ PINN with 5 layers and 128 hidden units uses 66,433 free parameters.

3.3 Remarks

Whilst the PINN performs well for low frequencies, it struggles to scale to higher frequencies; the higher frequency PINN requires many more free parameters, converges much more slowly and has worse accuracy. Multiple significant and related factors contribute to this issue. One is that as the complexity of the solution increases, the network requires more free parameters to accurately represent it, making the optimisation problem harder. Another is that as the frequency increases, more training sample points are required to sufficiently sample the domain, again making the optimisation problem harder. Third is the *spectral bias* of neural networks, which is the observation that neural networks tend to learn higher frequencies much more slowly than low frequencies, a property which has been rigorously studied. Compounding these factors is the fact that, as the size of the network, number of training points, and convergence time grow, the computational resources required increase significantly.

It is important to note that for this problem, scaling to higher frequencies is equivalent to scaling to larger domain sizes. Because we have normalised the input variable to $[-1, 1]$ within the domain before inputting it to the network, keeping $\omega = 1$ but expanding the domain size by 15 times and re-normalising presents the same optimisation problem to the neural network as changing ω to 15. Indeed, increasing the domain size and scaling to higher frequencies are related problems, and the above case study highlights a general observation of PINNs: as the domain size increases, the PINN optimisation typically becomes much harder and takes much longer to converge.

Another important note is that classical methods also typically scale poorly to large domain sizes/higher frequencies. For example, when solving the problem above, the number of mesh points required by standard finite difference modelling would scale $\propto \omega^d$. However, FD modelling would not suffer from the additional PINN-related problems described above of many more free parameters in the optimisation problem and slower convergence due to spectral bias.

In the next section, we will present FBPINNs, which, as we shall see in Section 5, are able to solve the case study above much more accurately and efficiently than the PINNs studied.

4 Finite basis physics-informed neural networks (FBPINNs)

4.1 Workflow overview

FBPINNs are a general domain decomposition approach for solving large, multi-scale problems relating to differential equations. The main goal of FBPINNs is to address the scaling issues of PINNs described above, which is achieved by using a combination of domain decomposition, subdomain normalisation and flexible training schedules. In this subsection, we give an overview of FBPINNs before giving a detailed mathematical description in Section 4.2.

The FBPINN workflow is shown in Fig. 2. In FBPINNs, the problem domain is divided into overlapping subdomains. A neural network is placed within each subdomain such that within the center of the subdomain, the network learns the full solution, whilst in the overlapping regions, the solution is defined as the sum over all overlapping networks. Before being summed, each network is multiplied by a smooth, differentiable window function which locally confines it to its subdomain. In addition to this, the input variables of each network are separately normalised over each subdomain, and we can define flexible training schedules which allow us to restrict which subdomain networks are updated at each gradient descent training step (described in more detail in Section 4.3).

By dividing the domain into many subdomains, the single, large PINN optimisation problem is turned into many smaller subdomain optimisation problems. By using separate subdomain normalisation (as well as a domain decomposition appropriate for the complexity of the solution), we ensure that the effective solution frequency each subdomain optimisation problem sees is low. The use of flexible training schedules allows us to focus on solving smaller parts of the domain sequentially instead of all of them at once. Our hypothesis is that all three of these strategies alleviate the scaling issues described above, leading to an easier global optimisation problem.

With any domain decomposition technique, it is important to ensure that the individual neural network solutions are communicated and match across the subdomain interfaces. In FBPINNs, during training, the neural networks share their solutions in the overlap regions between subdomains, and by mathematically constructing our global solution as the sum of these solutions, we automatically ensure the solution is continuous across subdomains. This approach allows FBPINNs to use a similar loss function to PINNs, without requiring the use of additional interface loss terms, in contrast to other domain decomposition approaches (e.g. [48]).

Finally, FBPINNs can be trained in a highly parallel fashion, decreasing their training times, and we present an algorithm for doing so in Section 4.4. FBPINNs are inspired by classical finite element methods (FEMs), where the solution of the differential equation is expressed as the sum of a finite set of basis functions with compact support, although we note that FBPINNs use the strong form of the governing equation as opposed to the weak form in FEMs.

4.2 Mathematical description

We will now give a detailed mathematical description of FBPINNs. In FBPINNs, the problem domain $\Omega \subset \mathbb{R}^d$ is subdivided into n overlapping subdomains, $\Omega_i \subset \Omega$. FBPINNs can use any type of subdivision, regular or irregular, with any overlap width, as long as the subdomains overlap. An example subdivision, a regular hyperrectangular division, is shown in Fig. 2. For simplicity, we use this type of division for the rest of this work.

Given a subdomain definition, the following FBPINN solution ansatz is used to define the approximate solution to the problem defined by Equation 1

$$\hat{u}(x;\theta) = \mathcal{C}\left[\,\overline{NN}(x;\theta)\,\right] \,, \tag{12}$$

where

$$\overline{NN}(x;\theta) = \sum_{i}^{n} w_i(x) \cdot \text{unnorm} \circ NN_i \circ \text{norm}_i(x) \,, \tag{13}$$

$NN_i(x;\theta_i)$ is a separate neural network placed in each subdomain Ω_i, $w_i(x)$ is a smooth, differentiable window function that locally confines each network to its subdomain, $\mathcal{C}$ is a constraining operator which adds appropriate "hard" constraints to the ansatz such that it satisfies the boundary conditions (following the same procedure described in Section 2.1), norm_i denotes separate normalisation of the input vector x in each subdomain, unnorm denotes a common unnormalisation applied to each neural network output, and $\theta = \{\theta_i\}$.

The purpose of the window function is to locally confine each neural network solution, NN_i, to its subdomain. Any differentiable function can be used, as long as it

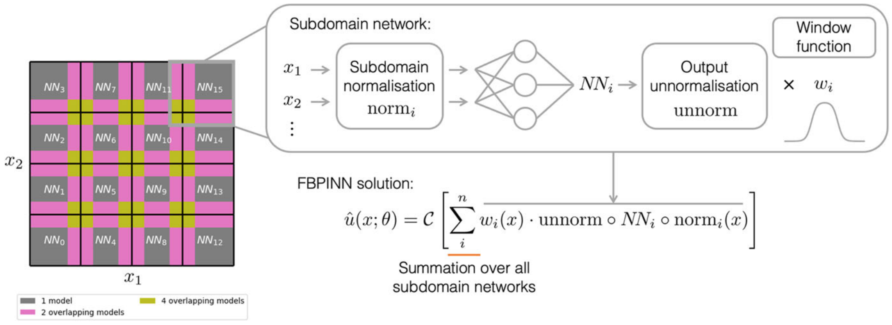
Subdomain network:
Window function
x_1
x_2
Subdomain normalisation
norm_i
NN_i
Output unnormalisation
unnorm
$\times$ w_i
NN_3 NN_7 NN_{11} NN_{15}
NN_2 NN_6 NN_{10} NN_{14}
NN_1 NN_5 NN_9 NN_{13}
NN_0 NN_4 NN_8 NN_{12}
FBPINN solution:
$\hat{u}(x;\theta) = \mathcal{C}\left[\sum_i^n w_i(x) \cdot \mathrm{unnorm} \circ NN_i \circ \mathrm{norm}_i(x)\right]$
Summation over all subdomain networks
1 model
2 overlapping models
4 overlapping models

Fig. 2 FBPINN workflow. FBPINNs use domain decomposition and separate subdomain normalisation to address the issues related to scaling PINNs to large domains. First, the problem domain is divided into overlapping subdomains; an example 2D hyperrectangular subdivision is shown. Next, separate neural networks are placed within each subdomain. Each network is locally confined to its subdomain by multiplying it with a differentiable window function, such that within the center of the subdomain, the network learns the full solution to the differential equation, whilst in the overlapping regions, the solution is defined as the sum over all overlapping networks. For each network, the input variables $x = (x_1, x_2, ...)$ are normalised between [-1,1] over the subdomain, and the output of the network is unnormalised using a common unnormalisation. Finally, an optional constraining operator $\mathcal{C}$ can be applied which appropriately constrains the ansatz such that it automatically satisfies the boundary conditions

is (negligibly close to) zero outside of the subdomain and greater than zero within it. For all the problems studied in this paper, which all use hyperrectangular subdomains, we use the following window function:

$$w_i(x) = \prod_j^d \phi((x^j - a_i^j)/\sigma_i^j)\phi((b_i^j - x^j)/\sigma_i^j) \; , \tag{14}$$

where j denotes each dimension of the input vector, a_i^j and b_i^j denote the midpoint of the left and right overlapping regions in each dimension (where $a_i^j < b_i^j$, i.e. the black lines in Fig. 2), $\phi(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, and σ_i^j is a set of parameters defined such that the window function is (negligibly close to) zero outside of the overlap region.

The window function is an important choice when defining FBPINNs. In particular, whilst any window function can be used, the window function for each subdomain network should sufficiently overlap with the window functions of its neighbouring subdomains, so that the network can communicate its solution and boundary values. Empirically, we find that larger overlaps lead to improved accuracy, although the computational cost is higher as more subdomain network evaluations are required in the overlapping regions.

The individual normalisation, norm_i in Equation 13, for each subdomain is applied by normalising the input vector between $[-1, 1]$ in each dimension over the subdomain before it is input to the network, whilst the common output unnormalisation unnorm is chosen such that each neural network output stays within the range $[-1, 1]$ and depends on the solution itself.

Finally, any neural network can be used to define $NN_i(x;\theta_i)$; in this paper, for simplicity, we only consider fully connected neural networks.

Given the ansatz defined by Equation 12, FBPINNs are trained using the unconstrained loss function

$$\mathcal{L}(\theta) = \mathcal{L}_p(\theta) = \frac{1}{N_p}\sum_{i}^{N_p} \| \mathcal{D}[\hat{u}(x_i;\theta);\lambda] - f(x_i) \|^2 , \tag{15}$$

using a set of training points $\{x_i\}$ sampled over the full domain Ω. This loss function is the same form as used when training the strongly constrained PINNs described in Section 2.1 and notably does not require the use of additional interface terms by construction of our ansatz. Alternatively, the constraining operator $\mathcal{C}$ in the FBPINN ansatz can be removed and the corresponding "weak" loss function (cf Equation 3) used to train FBPINNs, in a similar fashion to PINNs.

4.3 Flexible training schedules

Alongside the issues with scaling PINNs to large domains described in Section 3, another issue is the difficulty of ensuring that the PINN solution learnt far away from the boundary is consistent with the boundary conditions. More precisely, it is conceivable that, early-on in training, the PINN could fixate on a particular solution which is inconsistent with the boundary condition at a location in the domain far away from the boundary, because the network has not yet learnt the consistent solution closer to the boundary to constrain it. With two different particular solutions being learned, the optimisation problem could end up in a local minima, resulting in a harder optimisation problem. Indeed, we find evidence for this effect in our numerical experiments (in particular, Sections 5.2.4 and 5.5). Thus, for some problems, it may make sense to

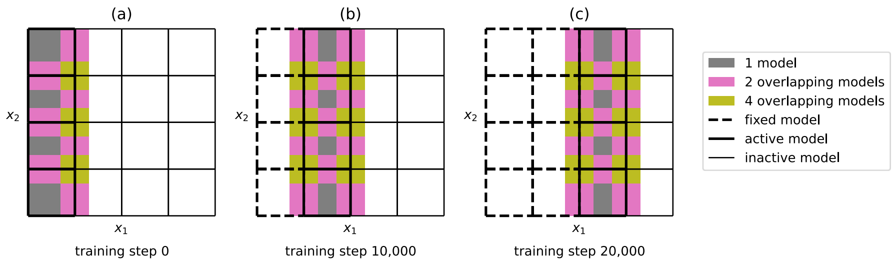

Fig. 3 Flexible training schedules for FBPINNs. We can design flexible training schedules which can help to improve the convergence of FBPINNs. These schedules define which subdomain networks are updated during each training step. Within these schedules, we define "active" models, which are the networks which are currently being updated; "fixed" models, which are networks which have already been trained and have their free parameters fixed; and "inactive" models which are as-of-yet untrained networks. The plots above show one particular training schedule designed to learn the solution "outwards" from the boundary condition, which in this case is assumed to be along the left edge of the domain. Note that during each training step, only training points from the active subdomains are required, shown by the coloured regions in the plot

sequentially learn the solution “outwards” from the boundary, in a similar fashion to, for example time-marching schemes employed in classical methods.

FBPINNs allow for such functionality, which is easy to implement because of their domain decomposition. At any point during training, we can restrict which subdomain networks are updated and can therefore design flexible training schedules to suit a particular boundary problem. An example training schedule is shown in Fig. 3. Within a training schedule, we define “active” models, which are the networks which are currently being updated; “fixed” models, which are networks which have already been trained and have their free parameters fixed; and “inactive” models which are as-of-yet untrained networks. During each training step, only active models and their fixed neighbours contribute to the summation in the FBPINN ansatz, and only training points within the active subdomains are sampled.

Algorithm 1 `PyTorch` psuedocode for a single global FBPINN training step

```
# 1) Get NN output and gradients in each subdomain
# Can use separate thread for each subdomain (indexed by ``im'')
for im in active_or_fixed_neighbour_models:
    x[im] = sample_subdomain_points(im)
    x_norm = (x[im] - x_mu[im]) / x_sd[im]
    u[im] = NN[im](x_norm)
    u[im] = u[im]*u_mu + u_sd
    u[im] = windows[im] * u[im]
    g[im] = problem.get_gradients(u[im], x[im])

# 2) Sum NN outputs in overlapping regions
# Requires communication between threads
for im in active_models:
    for iseg in overlapping_regions[im]:
        for im2 in overlapping_models[iseg]:
            if im2 != im:
                u[im][iseg] += u[im2][iseg].detach()
                g[im][iseg] += g[im2][iseg].detach()

# 3) Apply hard boundary conditions, compute loss,
# backpropagate and update free parameters
# Can use separate thread for each subdomain
for im in active_models:
    u[im], g[im] = problem.boundary(u[im], g[im])
    loss = problem.physics_loss(x[im], u[im], g[im])
    loss.backward()
    optimizers[im].step()
```

4.4 Parallel implementation

The FBPINN optimisation problem defined by Equation 15 is solved by using gradient descent, similar to PINNs. This can be naively implemented within a single, global optimisation loop, but in practice, we can train FBPINNs in a highly parallel and more data-efficient way by taking advantage of the domain decomposition. In this section, we describe a parallel implementation of FBPINNs.

There are two key considerations when parallelising FBPINNs. First is that, outside of each neural network's subdomain, its output is always zero after the window function has been applied, which means that training points outside of the subdomain will provide zero gradients when updating its free parameters. Thus, only training points within each subdomain are required to train each network, which allows training to be much more data-efficient. Second is that multiple parts of each training step can be implemented in parallel; a separate thread for each network can be used when calculating their outputs and gradients with respect to the input vector, and once the network outputs in overlapping regions have been summed, a separate thread for each network can be used to backpropagate the loss function and update their free parameters. This allows the training time to be dramatically reduced.

We now describe the parallel training algorithm in detail. We use a standard gradient descent training algorithm, consisting of a number of identical gradient descent steps implemented inside a `for` loop. The pseudocode for each global training step is shown in Algorithm 1, and a schematic of how each training step affects each subdomain is shown in Fig. 4. Each training step consists of three distinct steps. First, for each subdomain, training points are sampled throughout the subdomain, normalised and input into the subdomain network. The output of the network is unnormalised, multiplied by the window function, and its appropriate gradients (depending on the specific problem) with respect to the input variables are computed using autodifferentiation. Second, for training points which intersect the overlapping regions between subdomains, the network outputs and gradients are shared across subdomains and summed. Third, for each subdomain, the constraining operator is applied to the summed solution

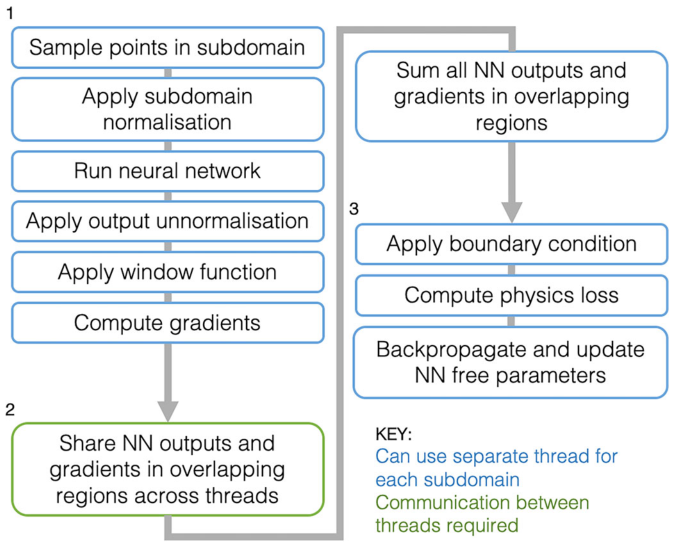
1
Sample points in subdomain
Apply subdomain normalisation
Run neural network
Apply output unnormalisation
Apply window function
Compute gradients
2
Share NN outputs and gradients in overlapping regions across threads
Sum all NN outputs and gradients in overlapping regions
3
Apply boundary condition
Compute physics loss
Backpropagate and update NN free parameters
KEY:
Can use separate thread for each subdomain
Communication between threads required

Fig. 4 Visual schematic of a single FBPINN training step for each subdomain. Training can be carried out in parallel across subdomains, and communication between subdomains is only required during step 2 where each subdomain network must share its outputs with its neighbouring subdomains

and its gradients, the loss function is computed using these quantities, and the free parameters of the network are updated using backpropagation.

We note that because each network has an independent set of free parameters, the computational graph can be discarded (or "detached") for all outputs shared in its overlapping regions except its own, allowing the backpropagation operation to be implemented in parallel without needing communication. Another note is that we must ensure that the same training data points are used in the overlapping regions for each network such that their solutions can be summed.

This parallel design makes training FBPINNs highly scalable. As the problem size grows, more subdomains can be added, and each subdomain can be trained on a separate thread. If the number of training points per subdomain and size of each subdomain network stays constant, the computational cost of the global FBPINN training step grows linearly with the number of subdomains. Each subdomain thread can be placed on a separate GPU, allowing multi-GPU training if necessary. As discussed above, the only communication required between threads is the sharing of subdomain outputs within overlapping regions.

5 Numerical experiments

5.1 Overview of experiments

In this section, we carry out a number of experiments to test the accuracy and efficiency of FBPINNs. We are interested in how FBPINNs scale to larger problem sizes, and our experiments range from smaller to larger problems, both in terms of their domain size and dimensionality. Problems with multi-scale solutions are included.

First, in Section 5.2, FBPINNs are tested on the motivating 1D example problem presented in Section 3 (that of learning the solution to $\frac{du}{dx} = \cos \omega x$). Harder versions of this problem are introduced, including one with a multi-scale solution and another using second-order derivatives in the underlying differential equation. In summary, we find that FBPINNs are able to accurately and efficiently learn the solutions to these problems when ω is high, significantly outperforming PINNs. In Section 5.3, we extend the motivating 1D problem to 2D, learning the solution to the equation $\frac{\partial u}{\partial x_1} + \frac{\partial u}{\partial x_2} = \cos(\omega x_1) + \cos(\omega x_2)$ when ω is high. Again, the FBPINN significantly outperforms the PINN tested. In Section 5.4, we test FBPINNs on a standard PINN benchmark problem, which is the (1+1)D viscous time-dependent Burgers equation. In this case, the FBPINN matches the accuracy of the PINN tested, whilst being significantly more data-efficient. Finally, in Section 5.5, we learn the solution to the (2+1)D time-dependent wave equation for a high-frequency point source propagating through a medium with a non-uniform wave speed, which is the most challenging problem studied here. Whilst the PINN tested exhibits unstable convergence, the FBPINN using a time-marching training schedule is able to robustly converge to the solution.

Some of the FBPINN settings and hyperparameters are fixed across all experiments. Specifically, the same optimiser (Adam [60]), learning rate (0.001), network type (fully connected), network activation function (tanh), type of subdomain division (hyperrectangular) and window function (as defined in Equation 14) are used across all experiments. The relevant corresponding aspects are also fixed and the same across all of the PINN benchmarks used. Other settings and hyperparameters, such as the number of training steps, number of training points, number of hidden layers, number of hidden units, number of subdomains, overlapping width of each subdomain, output unnormalisation and training schedule, vary depending on the problem, and for some case studies, we show ablations of them. For clearer comparison, we always use the same training point sampling scheme and density for both PINNs and FBPINNs, the same unnormalisation of their network outputs and the same constraining operator when forming their ansatze. The PINN input variable is always normalised between $[-1, 1]$ in each dimension across the problem domain before being input to the network. The PINNs are implemented within the same coding framework as the FBPINNs, which uses the `PyTorch` library [56]. All 1D problems are trained using a single CPU core, whilst all other problems are trained using a single NVIDIA Titan V GPU. For this work, we only use a single thread when training FBPINNs, although this thread does exactly implement the parallel training algorithm described in Section 4.4. Evaluating the multi-threaded performance of our parallel algorithm will be the subject of future work.

5.2 1D sinusoidal experiments

First, we test FBPINNs using the motivating 1D problem described in Section 3 (Equation 8). The following FBPINN ansatz is used:

$$\hat{u}(x;\theta) = \tanh(\omega x)\overline{NN}(x;\theta)\ , \tag{16}$$

which uses the same constraining operator as the PINN ansatz defined in Equation 10.

5.2.1 Low-frequency case ($\omega = 1$)

For the low-frequency case, the domain $x \in [-2\pi, 2\pi]$ is divided into $n = 5$ overlapping subdomains shown in Fig. 5 (c). Each subdomain is defined such that all of their overlapping regions have a width of 1.3 and their associated window functions (as defined in Equation 14) are also shown in Fig. 5 (c). Each subdomain network has 2 hidden layers and 16 hidden units per layer. Similar to the PINN in Section 3.1, the output of each subdomain network is unnormalised by multiplying it by $\frac{1}{\omega}$ before summation, and the FBPINN is trained using 50,000 training steps and 200 training points regularly spaced over the domain. An "all-active" training schedule is used, where all of the subdomain networks are active every training step. The FBPINN has 1605 free parameters in total.

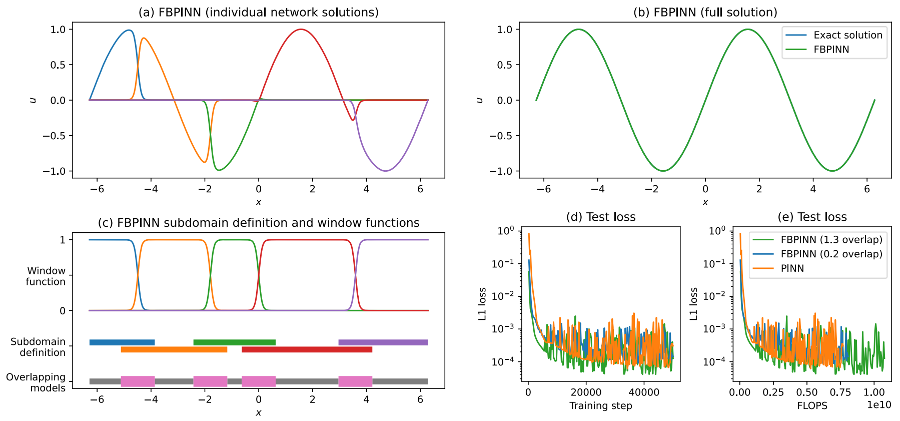

Fig. 5 Performance of FBPINNs on the problem $\frac{du}{dx} = \cos(\omega x)$ when $\omega = 1$. For this case, we find the FPINN described in Section 5.2.1 has similar performance to the $\omega = 1$ PINN in Section 3.1 (shown in Fig. 1 (a)). The individual FBPINN subdomain solutions after training are shown in (a). The full FBPINN solution compared to the exact solution is shown in (b). The FBPINN subdomain definition, overlapping regions in the domain (thick pink lines), and window function for each subdomain are shown in (c). The L1 error between the FBPINN solution and the exact solution against training step is shown in (d) and (e). Also shown in (d) and (e) are the convergence curves for a FBPINN trained with a smaller subdomain overlap width and the $\omega = 1$ PINN from Section 3.1

Figure 5 (a) shows the individual network solutions after training and Fig. 5 (b) shows the full FBPINN solution. For ease of interpretation, each individual network plot in Fig. 5 (a) shows the output of each subdomain network just before summation but with the constraining operator ($\tanh(\omega x)\cdot$) applied. Figure 5 (d) compares the L1 convergence curve of the FBPINN to the L1 convergence curve of the low-frequency PINN (with 2 layers and 16 hidden units) studied in Section 3.1. Figure 5 (e) shows the same curve against the cumulative number of training FLOPS required during forward inference of the subdomain networks, which is a measure of data-efficiency[1]. For this case study, we find that the FBPINN is able to solve the problem as accurately and with a similar data-efficiency to the PINN.

For this case study, we also test the sensitivity of the FBPINN to different subdomain overlap widths. Figure 5 (d) and (e) show the convergence curve for the same FBPINN but with its subdomains defined such that all overlapping regions have a width of 0.2. In this case, the FBPINN has similar performance.

5.2.2 High-frequency case ($\omega = 15$)

Next, we test the performance of the FBPINN when $\omega = 15$ which is a much harder problem, as discussed in Section 3. For this case, we divide the domain into n =

[1] Note, this measure only counts FLOPS spent during the forward inference of the networks and does not count FLOPS spent during gradient computation, backpropagation or any other part of the training algorithm. See Appendix A for the exact formula used.

30 equally spaced subdomains with overlapping widths of 0.3, as shown in Fig. 6 (c). The subdomain network size is kept the same as the case above at 2 layers and 16 hidden units per layer, and the same "all-active" training schedule is used. The FBPINN has 9630 free parameters in total. Similar to the high-frequency PINNs tested in Section 3.2, the number of regularly spaced training points is increased to $200 \times 15 = 3000$. We compare the FBPINN to the best performing high-frequency PINN from Section 3.2, namely the PINN with 5 layers and 128 hidden units.

Figure 6 shows the same plots as Fig. 5 for this case. We find that, in stark contrast to the PINN, the FBPINN is able to converge to the solution with very high accuracy in very few training steps. Furthermore, training the FBPINN requires multiple orders of magnitude less forward inference FLOPS than the PINN. This is because it uses much smaller network sizes in each of its subdomains, dramatically reducing the amount of computation required.

As an additional benchmark, we compare the FBPINN to a PINN where the input vector is transformed using Fourier features [45] before being input to the PINN (see Appendix B for the mathematical definition of these Fourier features). Fourier features can help neural networks learn high-frequency functions [45], and they have been shown to help the convergence of PINNs [44]. A PINN with 3 layers and 32 hidden units (similar to the PINNs in Section 3.2) is trained using 32 Fourier features with frequencies randomly sampled from a Gaussian distribution with a mean equal to the frequency of the solution ($\omega = 15$) and a standard deviation of 10. Figure 6(d) shows the resulting L1 convergence curve of the PINN. We find that the Fourier features significantly improve the performance of the PINN; however, its rate of convergence and final L1 error remains worse than the FBPINN.

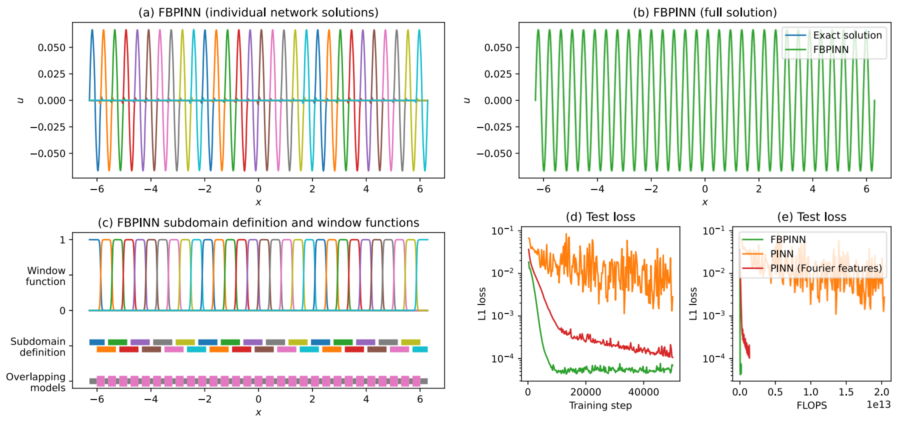

Fig. 6 Performance of FBPINNs on the problem $\frac{du}{dx} = \cos(\omega x)$ when $\omega = 15$. The FBPINN described in Section 5.2.2 is compared to the best-performing $\omega = 15$ PINN in Section 3.2 (namely the PINN with 5 layers and 128 hidden units, shown in Fig. 1 (d)). For this case, we find the FPINN significantly outperforms the PINN, converging to the solution with much higher accuracy and much less training steps. This plot has the same layout as Fig. 5

5.2.3 Multi-scale case

We extend the difficulty of this problem by including multi-scale frequency components in its solution. Specifically, we consider the modified problem

$$\begin{aligned} \frac{du}{dx} &= \omega_1 \cos(\omega_1 x) + \omega_2 \cos(\omega_2 x) \ , \\ u(0) &= 0 \ , \end{aligned} \tag{17}$$

which has the exact solution

$$u(x) = \sin(\omega_1 x) + \sin(\omega_2 x) \ . \tag{18}$$

Here, we chose $\omega_1 = 1$ and $\omega_2 = 15$, i.e. the solution contains both a high- and low-frequency component.

The same FBPINN and PINN from Section 5.2.2 are retrained for this problem, except that their loss functions are modified to use the differential equation above, unnormalisation is applied to both by multiplying their network outputs by 2, and ω_2 is used in their ansatz constraining operator ($\tanh(\omega_2 x)\cdot$). For this case, we also train PINNs with smaller network sizes, namely 2 layers and 16 hidden units and 4 layers and 64 hidden units.

The FBPINN individual network solutions, the FBPINN full solution and the three PINN solutions are shown in Fig. 7 (a), (b), (c), (d) and (e) respectively, and their L1 convergence curves are compared in Fig. 7 (f) and (g). Similar results are observed to Section 5.2.2: in stark contrast to the PINNs, the FBPINN is able to converge to the solution with a much higher accuracy in a much smaller number of training steps. Whilst the PINNs with 4 and 5 layers are able to model all of the cycles in the solution, their accuracy is nearly two orders of magnitude worse than the FBPINN, and their convergence curve is much more unstable.

Similar to Section 5.2.2, we compare the FBPINN to a PINN with Fourier input features. The same PINN with Fourier features from Section 5.2.2 is retrained for this problem, and its L1 convergence curve is shown in Fig. 7 (f). Similar results are found, namely that the Fourier features significantly improve the performance of the PINN; however, its rate of convergence and final L1 error remains worse than the FBPINN.

5.2.4 Second-order derivative case

We also extend the difficulty of this problem by changing the underlying equation from a first-order differential equation to a second-order equation. Namely, we consider the related problem

$$\begin{aligned} \frac{d^2u}{dx^2} &= \sin(\omega x) \ , \\ u(0) &= 0 \ , \\ \frac{du}{dx}(0) &= -\frac{1}{\omega} \ , \end{aligned} \tag{19}$$

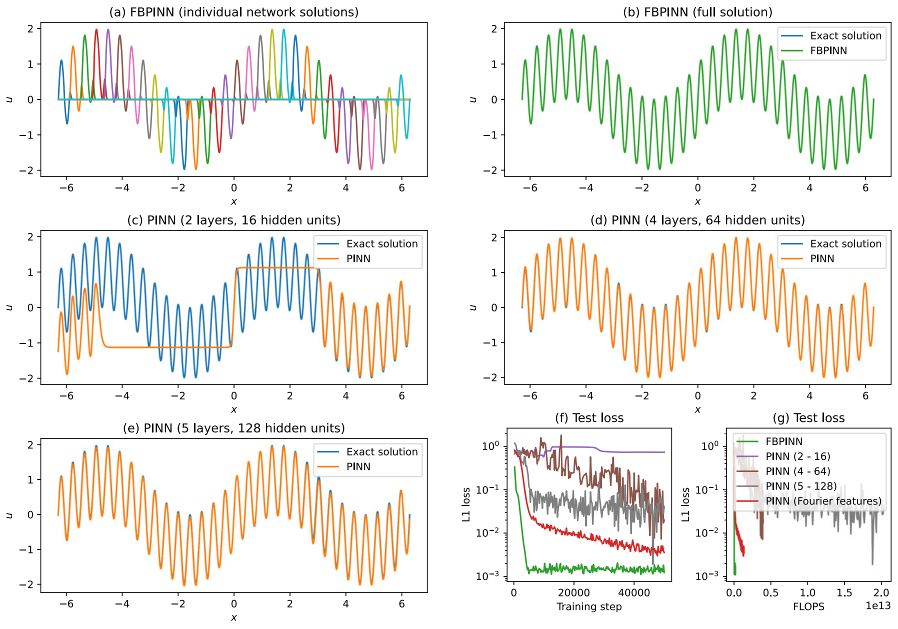

Fig. 7 Performance of FBPINNs on the multi-scale problem $\frac{du}{dx} = \omega_1 \cos(\omega_1 x) + \omega_2 \cos(\omega_2 x)$ where $\omega_1 = 1$ and $\omega_2 = 15$. The FBPINN described in Section 5.2.3 is compared to three different PINNs which have 2 layers and 16 hidden units, 4 layers and 64 hidden units and 5 layers and 128 hidden units. Similar to Fig. 6, the FBPINN significantly outperforms the PINNs tested. The individual FBPINN subdomain solutions after training are shown in (a). The full FBPINN solution is shown in (b). The three PINN solutions are shown in (c)–(e). The L1 errors of the FBPINN and PINN solutions compared to the exact solution are shown in (f) and (g)

which has the exact solution

$$u(x) = -\frac{1}{\omega^2} \sin(\omega x) \; . \tag{20}$$

We use the FBPINN ansatz

$$\hat{u}(x;\theta) = -\frac{1}{\omega^2} \tanh(\omega x) + \tanh^2(\omega x) \overline{NN}(x;\theta) \; , \tag{21}$$

such that the boundary conditions are satisfied and the same construction for the PINN ansatz. Alike Section 5.2.2, we consider the high-frequency case $\omega = 15$.

The same FBPINN and PINN from Section 5.2.2 are retrained for this problem, except for the changes to the problem definition above, and that the outputs of the FBPINN and PINN networks are unnormalised by multiplying them by $\frac{1}{\omega^2}$. We also train both networks for twice as long (100,000 steps).

The resulting FBPINN and PINN solutions are shown in Fig. 8 (c) and (a), along with their second-order derivatives in Fig. 8 (d) and (b). We find that both methods struggle to accurately model the solution, although the FBPINN is able to capture all of its cycles. Whilst both models learn the solution accurately in the vicinity of

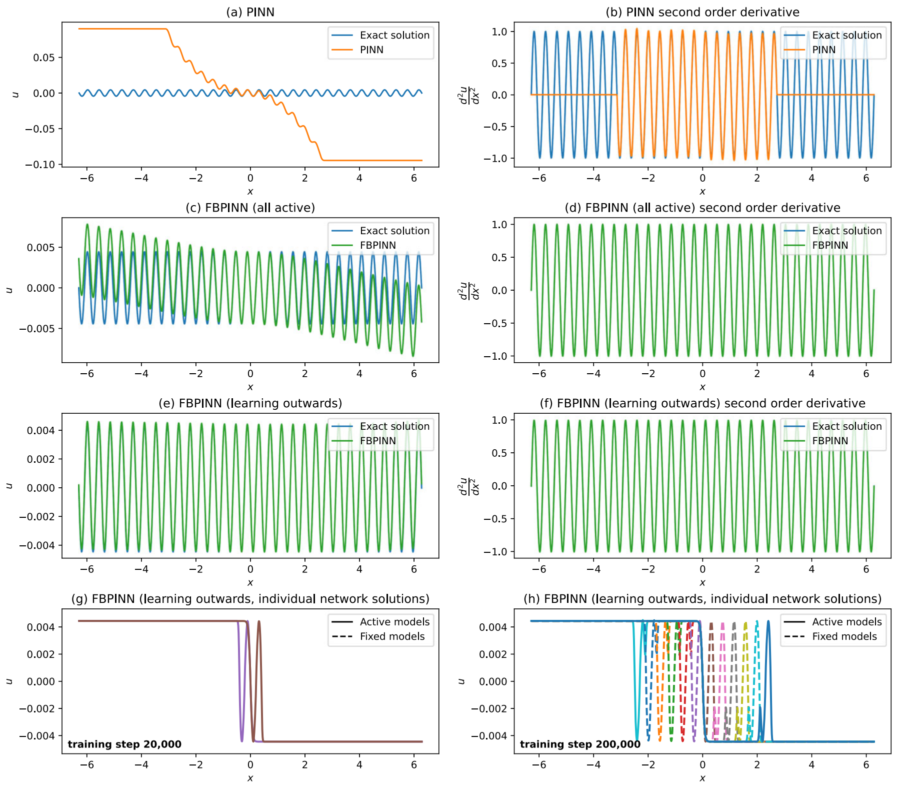

Fig. 8 Performance of FBPINNs on the problem $\frac{d^2u}{dx^2} = \sin(\omega x)$ with $\omega = 15$. Two FBPINNs with different training schedules are tested for this problem. The first has an "all-active" training schedule, where all models are active all of the time, and its resulting solution and second-order derivative are shown in (c) and (d). The second uses a "learning outwards" training schedule which slowly expands the active model outwards from the boundary condition at $x = 0$ as training progresses (as depicted in (g) and (h)), and its solution and second-order derivative are shown in (e) and (f). Both FBPINNs use the subdomain definition shown in Fig. 6 (c), and are compared to a PINN with 5 layers and 128 hidden units, shown in (a) and (b)

the boundary condition, they learn it poorly outside of it. One explanation is that both models are suffering from integration errors; a small error in the second-order derivative (which is being penalised in the loss function) can lead to large errors in the solution. Indeed, both models learn much more accurate second-order derivatives, as seen in Fig. 8. Another explanation is that away from the boundary, the FBPINN and PINN are fixating on a different (and incorrect) particular solution of the underlying equation. In particular, away from the boundary, the FBPINN solution appears to be superimposed with a linear function of the input variable, which is a feasible solution under this differential equation (but is not consistent with the boundary conditions).

To improve the FBPINN solution further, we retrain the FBPINN using a training schedule that allows the solution to be "learned outwards" from the boundary condition. The schedule starts with only the two models in the center of the domain being active and then slowly expands the active model outwards in the positive and negative directions, fixing the previously active models behind them, as shown in Fig. 8 (g) and (h). In this case, 500,000 training steps are used (equating to 33,333 training steps per active model). The resulting solution and its second-order derivative are shown in Fig. 8 (e) and (f). We find that this FBPINN performs best, accurately modelling the solution many cycles away from the boundary condition, although small errors remain at the edges of the domain.

5.3 2D sinusoidal experiments

In this section, we study the extension of the motivating problem above from 1D to 2D. Specifically, we consider the problem

$$\begin{aligned} \frac{\partial u}{\partial x_1} + \frac{\partial u}{\partial x_2} &= \cos(\omega x_1) + \cos(\omega x_2) \ , \\ u(0, x_2) &= \frac{1}{\omega} \sin(\omega x_2) \ , \end{aligned} \tag{22}$$

where $x = (x_1, x_2) \in \mathbb{R}^2, u \in \mathbb{R}^1$, with a problem domain $x_1 \in [-2\pi, 2\pi], x_2 \in [-2\pi, 2\pi]$. This problem has the exact solution

$$u(x_1, x_2) = \frac{1}{\omega} \sin(\omega x_1) + \frac{1}{\omega} \sin(\omega x_2) \ . \tag{23}$$

We use the FBPINN ansatz

$$\hat{u}(x_1, x_2; \theta) = \frac{1}{\omega} \sin(\omega x_2) + \tanh(\omega x_1) \overline{NN}(x_1, x_2; \theta) \ , \tag{24}$$

such that the boundary conditions are satisfied, using the same constraining operator for the PINN ansatz. Similar to Section 5.2.2, we consider the high-frequency case $\omega = 15$. Note that the solution along the second dimension x_2 is already provided in the ansatz, and so the FBPINN and PINN only need to learn to correct the ansatz along the first dimension (although x_2 is still input to the networks and so they could still learn an incorrect solution along the second dimension).

For the FBPINN, we divide the 2D domain into $n = 15 \times 15 = 225$ equally spaced subdomains with overlapping widths of 0.6, as shown in Fig. 9 (g). Each subdomain network has 2 layers and 16 hidden layers, and we use the "all-active" training schedule defined above. For the PINN, a network with 5 layers and 128 hidden units is chosen. The FBPINN has 75,825 free parameters whilst the PINN has 66,561 free parameters. The FBPINN and PINN are both unnormalised by multiplying their network outputs by $\frac{1}{\omega}$, and both are trained using $900 \times 900 = 810{,}000$ training points regularly spaced throughout the domain and 50,000 training steps. 1000×1000 regularly sampled test points throughout the domain are used when computing their L1 error compared to the exact solution.

Figure 9 (c) shows the exact solution, Fig. 9 (b) and a show the FBPINN and PINN solutions and Fig. 9 (f) and (e) show their difference to the exact solution. The FBPINN

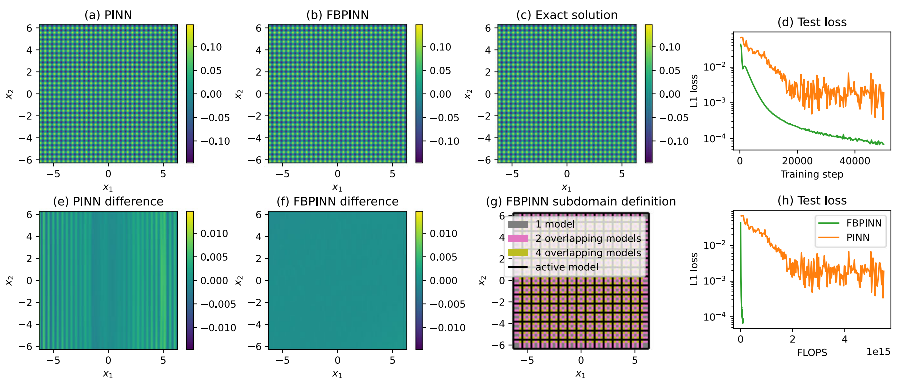

Fig. 9 Performance of FBPINNs on the problem $\frac{\partial u}{\partial x_1} + \frac{\partial u}{\partial x_2} = \cos(\omega x_1) + \cos(\omega x_2)$ with $\omega = 15$. The FBPINN described in Section 5.3 is compared to a PINN with 5 layers and 128 hidden units. Similar to the 1D sinusodial problems above, we find that the FBPINN significantly outperforms the PINN tested. The FBPINN subdomain definition is shown in (g). The exact solution is shown in (c). The FBPINN solution and its difference to the exact solution are shown in (b) and (f), and a similar set of plots for the PINN are shown in (a) and (e). The L1 errors of the FBPINN and PINN solutions compared to the exact solution are shown in (d) and (h)

and PINN convergence curves are compared in Fig. 9 (d) and (h). Similar observations to Section 5.2.2 can be made: the FBPINN is able to converge to the solution with much higher accuracy and much less training steps than the PINN; whilst the PINN is able to model all of the cycles of the solution, its accuracy is over one order of magnitude worse than the FBPINN; because a much smaller subdomain network size is used in the FBPINN, training the FBPINN requires multiple orders of magnitude less forward inference FLOPS than the PINN.

5.4 (1+1)D Burgers equation

In this section, the FBPINN is tested using a standard PINN benchmark problem, which is the (1+1)D viscous time-dependent Burgers equation, given by

$$\begin{aligned}\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} &= \nu\frac{\partial^2 u}{\partial x^2} \,, \\ u(x,0) &= -\sin(\pi x) \,, \\ u(-1,t) &= 0 \,, \\ u(+1,t) &= 0 \,,\end{aligned} \tag{25}$$

where $x, t, u, \nu \in \mathbb{R}^1$, with a problem domain $x \in [-1, 1], t \in [0, 1]$. Interestingly, for small values of the viscosity parameter, ν, the solution develops a discontinuity at $x = 0$ as time increases. We use $\nu = 0.01/\pi$ such that this is the case. The exact solution is analytically available by the use of the Hopf-Cole transform (see [61] for details, which are omitted here for brevity).

To solve the problem, we use the FBPINN ansatz

$$\hat{u}(x,t;\theta) = -\sin(\pi x) + \tanh(x+1)\tanh(x-1)\tanh(t)\overline{NN}(x,t;\theta) \,, \tag{26}$$

such that the boundary conditions are satisfied, using the same constraining operator for the PINN ansatz.

For the FBPINN, we divide the 2D domain into $n = 4 \times 2 = 8$ equally spaced subdomains with overlapping widths of 0.4, as shown in Fig. 10 (a). We purposefully coincide the subdomain interfaces with the discontinuity in the solution at $x = 0$, to test how the domain decomposition affects the solution accuracy across the discontinuity. Each subdomain network has 2 layers and 16 hidden layers, and we use the “all-active” training schedule. For the PINN, a network with 4 layers and 64 hidden units is used, as in testing, we found smaller networks performed worse. The FBPINN has 2696 free parameters whilst the PINN has 12,737 free parameters. The FBPINN and PINN are both unnormalised by multiplying their network outputs by 1, and both are trained using $200 \times 200 = 40{,}000$ training points regularly spaced throughout the domain and

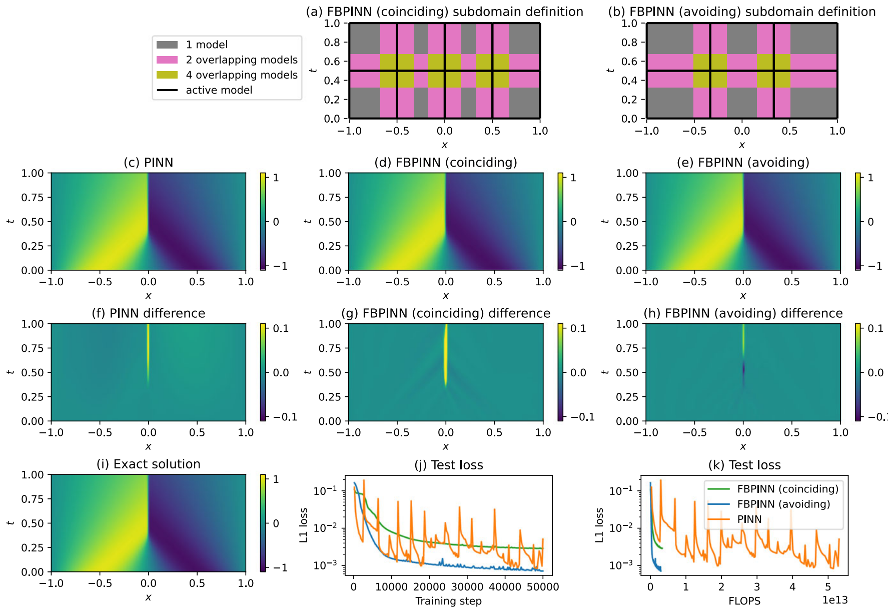

Fig. 10 Performance of FBPINNs on the (1+1)D viscous time-dependent Burgers equation. The exact solution is shown in (i). Two FBPINNs with different subdomain definitions are tested. The first uses a subdomain definition where the subdomain interfaces coincide with the discontinuity in the exact solution, shown in (a). The second uses a definition where the interfaces avoid the discontinuity, shown in (b). Both FBPINNs are compared to a PINN with 4 layers and 64 hidden units. The coinciding FBPINN solution and its difference to the exact solution are shown in (d) and (g). Similar sets of plots for the avoiding FBPINN and PINN are shown in (e) and (h) and (c) and (f) respectively. The L1 errors of the FBPINN and PINN solutions compared to the exact solution are shown in (j) and (k). We find that the coinciding FBPINN has slightly worse accuracy than the PINN, whilst the avoiding FBPINN has slightly better accuracy

50,000 training steps. 400×400 regularly sampled test points throughout the domain are used when computing their L1 error compared to the exact solution.

Figure 10 (i), (d) and (c) show the exact, FBPINN and PINN solutions respectively, and Fig. 10 (g) and (f) show the difference of the FBPINN and PINN solutions to the exact solution. Figure 10 (j) and (k) show the L1 convergence curves of the FBPINN and PINN. We observe that the FBPINN solution is slightly less accurate across the discontinuity than the PINN, although its overall convergence is more stable. Whilst the FBPINN is able to model the discontinuity, it appears that the window function and summation of networks does make it harder for the FBPINN to model the solution in this region. To avoid this issue, we retrain the FBPINN using 6 overlapping subdomains with interfaces which do not coincide with the discontinuity, as shown in Fig. 10 (b). The resulting solution, difference and convergence curves are shown in Fig. 10 (e), (h), (j) and (k), and we find that this FBPINN is able to more accurately model the discontinuity with a slightly higher overall accuracy than the PINN. Furthermore, as observed above, because a much smaller subdomain network size is used in the FBPINNs, training both FBPINNs requires multiple orders of magnitude less forward inference FLOPS than the PINN.

5.5 (2+1)D wave equation

Finally, we test the FBPINN using the (2+1)D time-dependent wave equation, modelling the dynamics of an initially stationary point source propagating through a medium with a non-uniform wave speed. Specifically, we solve the following problem:

$$\begin{aligned} \left[\nabla^2 - \frac{1}{c(x)^2}\frac{\partial^2}{\partial t^2}\right] u(x,t) &= 0 \,, \\ u(x,0) &= e^{-\frac{1}{2}(\|x-\mu\|/\sigma)^2} \,, \\ \frac{\partial u}{\partial t}(x,0) &= 0 \,, \end{aligned} \tag{27}$$

where $x, \mu \in \mathbb{R}^2$, $t, u, \sigma \in \mathbb{R}^1$, $c(x) \in \mathbb{R}^1$ is the spatially varying wave speed, and μ and σ control the starting location and central frequency of a Gaussian point source. The wave speed $c(x)$ is defined as a simple mixture of 2D Gaussian distributions and is shown in Fig. 11 (d). For this case, we use the problem domain $x_1 \in [-10, 10], x_2 \in [-10, 10], t \in [0, 10]$ and set $\mu = (0, 0)$ and $\sigma = 0.3$. Modelling the wave equation can be challenging in general because its solutions are typically broadband, oscillatory and dispersive in nature and can include reflections, refractions, wavefront compression and expansion through different velocity regions and a large range of amplitudes [62]. For this case, the solution (or "wavefield") expands outwards from the point source, compressing and expanding as it moves through regions with different wave speeds. We compare our results to the solution

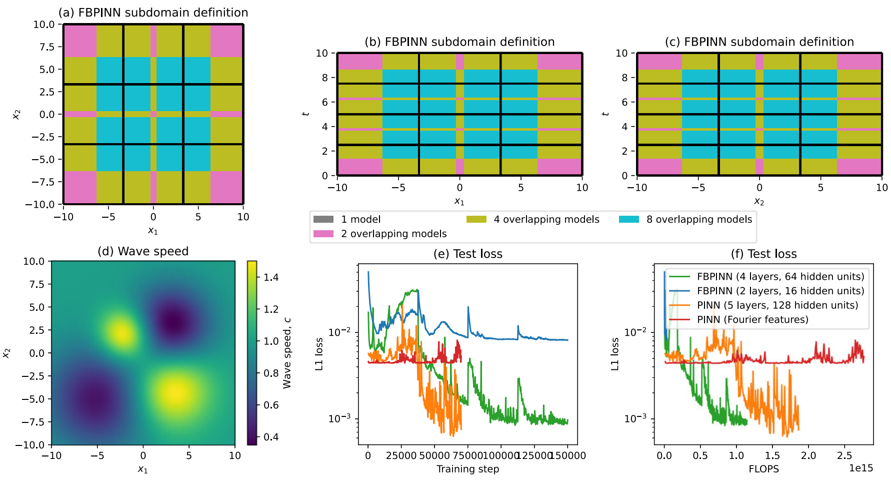

Fig. 11 Setup and convergence curves for the (2+1)D time-dependent wave equation problem. The FBPINN subdomain definition is shown in (a), (b) and (c), where the plots show orthogonal cross-sections through the middle of the domain. The spatially varying wave speed is shown in (d). The L1 errors of the FBPINN and PINN solutions compared to the solution from finite difference modelling are shown in (e) and (f))

from finite difference (FD) modelling (see Appendix C for our detailed implementation), which is shown in Fig. 12 (a)–(d) for 4 time steps spanning the domain.

To solve the problem, we use the FBPINN ansatz

$$\hat{u}(x, t; \theta) = \phi(5(2 - t/t_1))\, e^{-\frac{1}{2}(\|x-\mu\|/\sigma)^2} + \tanh^2(t/t_1)\overline{NN}(x, t; \theta)\ , \tag{28}$$

where $\phi(\cdot)$ is the sigmoid function and $t_1 = \sigma/c(\mu)$. This ansatz is designed such that at $t = 0$, the sigmoid function is (negligibly close to) 1, allowing the ansatz to match the boundary conditions. At times $t \gg 2\,t_1$, the sigmoid function is (negligibly close to) 0, removing the point source term from the ansatz. This is done so that the FBPINN does not need to learn to correct for this part of the ansatz once the point source has expanded away from its starting location. The same constraining operator is used for the PINN ansatz.

For the FBPINN, we divide the 3D domain into $n = 3\times3\times4 = 36$ subdomains with overlapping widths of 6 in the spatial dimensions and 2 in the time dimension, as shown in Fig. 11 (a)–(c). Each subdomain network has 4 layers and 64 hidden layers. For this problem, we use a "time-marching" training schedule, where initially only subdomains in the first time-step are active, before slowly expanding the active time-step outwards and fixing the earlier time-step models. For the PINN a network with 5 layers and 128 hidden units is chosen. The FBPINN has 460,836 free parameters whilst the PINN has 66,689 free parameters. The FBPINN and PINN are both unnormalised by multiplying their network outputs by 1, and both are trained using $58 \times 58 \times 58 = 195{,}112$ training points randomly sampled throughout the domain. The PINN is trained for 75,000 training steps, whilst the FBPINN is trained for 150,000 steps (equating to 37,500 training steps per active model). For this case, we chose random sampling over

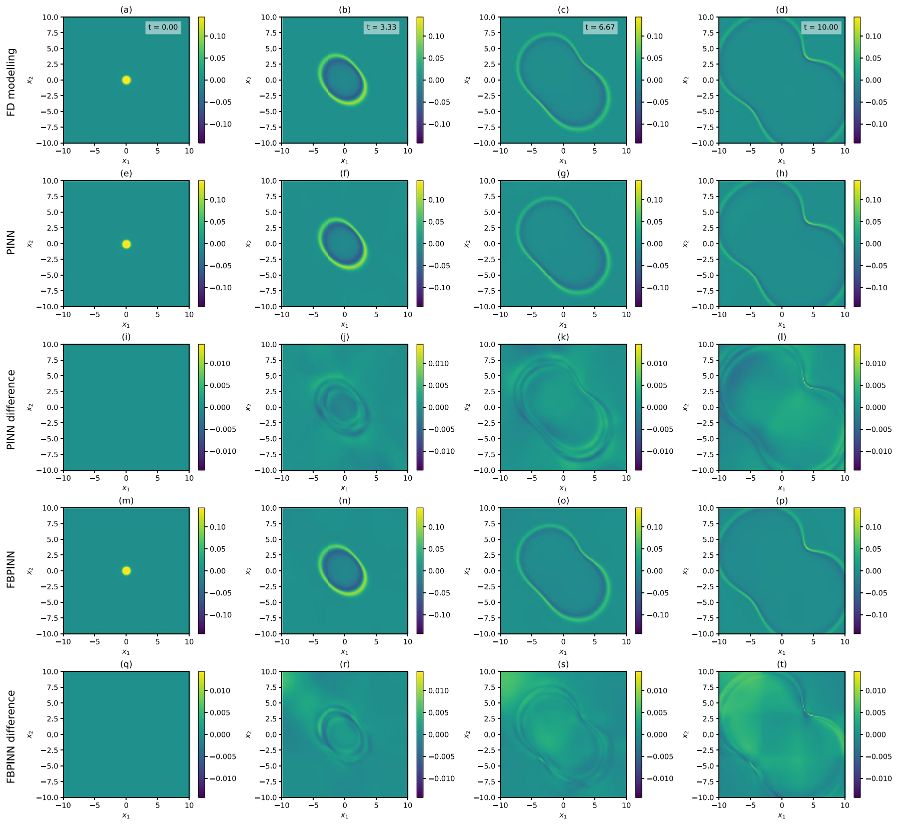

Fig. 12 Performance of FBPINNs on the (2+1)D time-dependent wave equation problem. The FBPINN described in Section 5.5 is compared to a PINN with 5 layers and 128 hidden units. We find that the FBPINN and PINN have similar accuracy, although the FBPINN converges more robustly to the solution (see Fig. 13). The solution at 4 time-steps spanning the domain from finite difference modelling is shown in (a)–(d). The PINN solution at these time-steps is shown in (e)–(h), and the FBPINN solution at these time-steps is shown in (m)–(p). The difference of the PINN solution to the solution from finite difference modelling is shown in (i)–(l), and similar difference plots for the FBPINN are shown in (q)–t

regular sampling because the training point density is relatively low compared to the frequency of the solution and in testing this allowed for better convergence of both the FBPINN and PINN. $100 \times 100 \times 10$ regularly sampled test points throughout the domain are used when computing their L1 error compared to the finite difference solution.

Figure 12 (e)–(h) and (m)–(p) show the resulting PINN and FBPINN solutions respectively over 4 time steps spanning the problem domain. Figure 12 (i)–(l) and (q)–(t) show their difference compared to the finite difference solution, and Fig. 11 (e) and (f) show their L1 convergence curves. We find that the FBPINN and PINN solutions have a similar accuracy, although the FBPINN takes roughly half as many forward inference FLOPS to train as the PINN, because its subdomain network size is smaller. We also test a FBPINN using a smaller subdomain network size (2 layers and 16 hidden units, the same as all previous examples), but this does not converge, as shown in the convergence plots in Fig. 11 (e) and (f). Whilst the PINN appears suitable for this problem, we plot the PINN and FBPINN solutions midway through training in Fig. 13. We find that whilst the FBPINN robustly learns the solution outwards from $t = 0$ as its time-marching training schedule progresses, even after 20,000 training steps, the PINN solution is still close to zero everywhere apart from the boundary

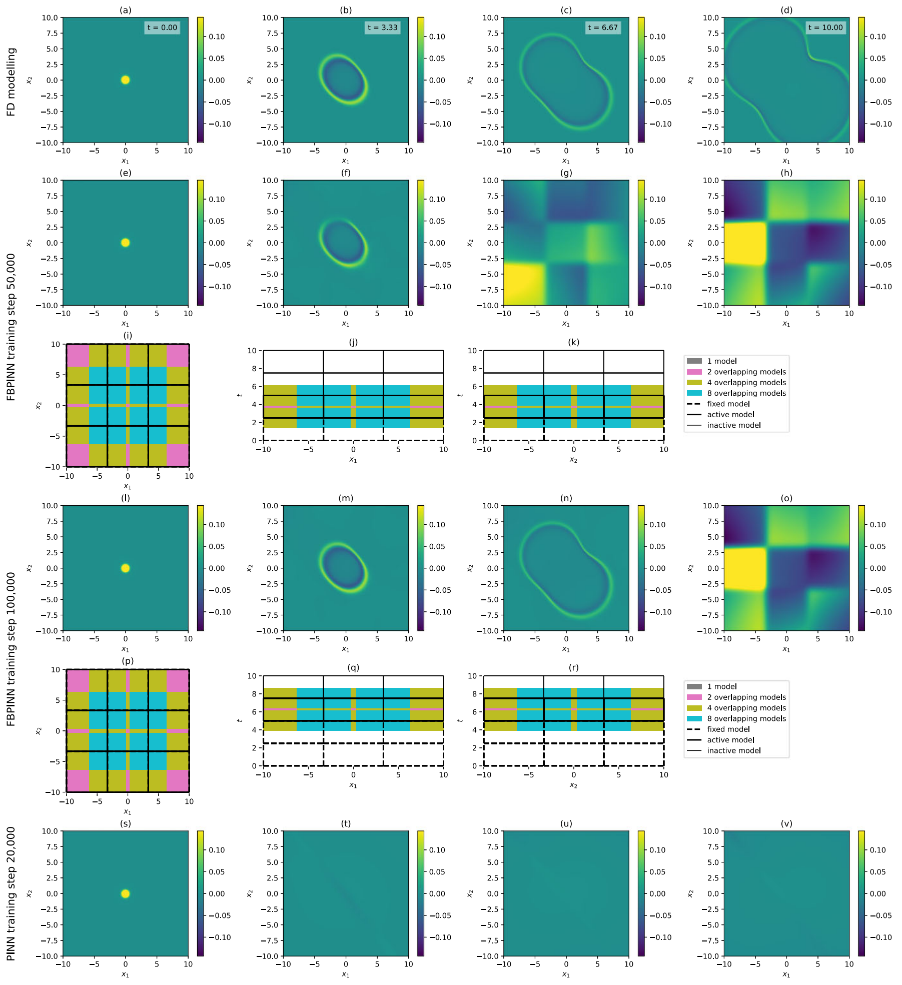

Fig. 13 FBPINN and PINN solutions during training for the (2+1)D time-dependent wave equation problem. The solution from finite difference modelling is shown in (a)–(d) (repeated from Fig. 12). The FBPINN solution at training step 50,000 and 100,000 as well as the PINN solution at training step 20,000 are plotted in (e)–(h), (l)–(o) and (s)–(v) respectively using the same 4 time steps. Whilst the FBPINN robustly learns the solution outwards from $t = 0$ as its time-marching training schedule progresses, even after 20,000 training steps, the PINN solution is still close to zero everywhere apart from the boundary condition. Orthogonal cross sections of the active, fixed and inactive FBPINN subdomains through the center of the domain during the time-marching training schedule are shown for each FBPINN training step in (i)–(k) and (p)–(r)

condition. This can also been seen in the convergence curve for the PINN in Fig. 11 (e), where its L1 error actually increases until approximately training step 40,000. One explanation for this is that the PINN is fixating on a different (incorrect) particular solution away from the boundary early-on in training, namely the easier particular solution $u(x, t) = 0$, causing the optimisation to become stuck in a local minima.

As a final test, we retrain the PINN using Fourier input features. Specifically, 128 Fourier features are used with frequencies randomly sampled from a Gaussian distribution with a mean equal to the dominant frequency of the solution ($\omega = 2\pi/\sigma$) and a standard deviation of 10. The L1 convergence curve of the PINN is shown in Fig. 11 (e). In this case, we find that the PINN is not able to converge.

6 Discussion

The numerical tests above confirm that FBPINNs provide a promising approach for scaling PINNs to large domains. They are able to accurately solve all of the smaller and larger scale problems studied, whilst in many cases, the standard PINN struggles. For the problems studied with smaller domains, such as the Burgers equation and the low-frequency sinusoidal problem, the FBPINN generally matches the PINN in performance. For the problems with larger domains, such as the wave equation and high-frequency sinusodial case studies, the FBPINN outperforms the PINN. The largest differences are seen in the high-frequency sinusoidal problems, where across all tests, the FBPINN converges with much higher accuracy and much less training steps than the PINN. For the wave equation problem, the FBPINN more robustly converges to the solution. These findings demonstrate that the combined use of domain decomposition, separate subdomain normalisation and flexible training schedules helps to alleviate some of the major issues related to scaling PINNs to large domains.

FBPINNs also appear to be more data-efficient than standard PINNs. Across all experiments, we find that FBPINNs are able to converge using smaller network sizes in their subdomains than the network size required by PINNs. The total number of forward inference FLOPS required during training of FBPINNs only depends on the subdomain network size and not the number of subdomains (see Appendix D for proof), and thus the FBPINNs studied here require much less computation than the PINNs to train them. This is likely because of their "divide and conquer" strategy; each subdomain appears to present an easier optimisation problem which only requires a small number of free parameters to solve. Indeed, FBPINNs with more subdomains and smaller network sizes than those tested could be even more data-efficient, and in the future, we plan to study in detail how reducing the subdomain size further affects their accuracy and whether there is an optimal subdomain and network size to use (e.g. similar to h-p refinement in FEM).

It is important to note that each problem studied requires different configurations of the FBPINN to converge well. For example, for the high-frequency 1D first-order sinusoidal problem, a FBPINN with an "all-active" training schedule performs well, whilst for the second-order variant, a FBPINN with a "learning outwards" training schedule is required to learn an accurate solution. Similarly, a subdomain network size of 2 layers and 16 hidden units is effective for every problem except the wave equation, which requires a larger network size given its subdomain definition. Thus, it appears important to fine-tune the FBPINN to each particular problem. This may be because different scaling issues affect each problem differently. We also find that the FBPINN performs slightly worse for the Burgers equation problem when its subdomain interfaces coincide with the discontinuity in the solution, and therefore care must be taken when choosing the subdomain division.

Whilst we have focused on the issues related to scaling PINNs to large domains, another important consideration is the scaling of PINNs to higher dimensions. Similar to classical methods, a major challenge is likely to be the exponentially increasing number of (training) points required to sample the domain as the number of dimensions increases. It is important to note that domain decomposition may still help to reduce the complexity of the ensuing optimisation problem, and indeed, FBPINNs are effective across all of the 1D, 2D and 3D problems studied. However, FBPINNs still require the same number of training points as standard PINNs, and so issues such as the increased computational workload required are likely to remain. Specific to FBPINNs, the number of overlapping models summed in each overlapping region using the hyperrectangular subdivision grows exponentially with the number of dimensions, which could negatively affect the underlying FBPINN optimisation problem. We plan to investigate the scaling of FBPINNs to higher dimensions in future work.

A future direction is to study the performance of the multi-threaded version of FBPINNs in detail. For the single-threaded implementation of our parallel training algorithm used here, the FBPINNs are typically 2 to 10 times slower to train than their corresponding PINNs, despite the FBPINNs being more data-efficient. This is because the single thread updates each subdomain network sequentially and also because each subdomain has a small network size and number of training points, meaning that the parallelism of the GPU is not fully utilised. The multi-threaded version (as described in Section 4.4) should reduce these training times by a factor proportional to the number of subdomains and yield a significant performance increase. A potential bottleneck is that each subdomain thread must wait for its neighbouring subdomain threads to run before summing the solution in its overlapping regions; this step could be made fully asynchronous by, e.g. caching the "latest available" outputs from the neighbouring subdomains instead of necessarily waiting for their outputs at the current training step.

Another important direction is to test FBPINNs using irregular domains and subdomains. This is an essential step in many state-of-the-art classical approaches, and FBPINNs are readily extendable in this regard. The same FBPINN framework can be used, and only the subdomain window functions and the functions which sample points from each subdomain and define the neighbours and overlapping regions of each subdomain in the parallel training algorithm (Algorithm 1) need to be changed. Going further, one could draw inspiration from classical methods where adaptive grids are used to solve multi-scale problems; it may be useful to adaptively change the subdomain definition and/or subdomain network in FBPINNs to dynamically fit to the solution. It is also important to note that in comparison to classical methods, where mesh refinement can be highly non-trivial, this could be relatively simple to implement using the mesh-free environment of FBPINNs.

Many other directions for applying and improving FBPINNs are possible. For example, FBPINNs could be used to solve inverse problems in the same way as PINNs, and it would be interesting to compare their performance. There are many other types of differential equations which could be tested. Whilst we use simple fully connected subdomain networks here, other architectures and activation functions could be investigated. Another promising direction would be to use transfer learning within our flexible training schedules, for example by using neighbouring fixed models to initialise the free parameters of newly active models, which may improve accuracy and reduce training times further.

A major goal in the field of SciML is to provide ML tools which are practically useful for real-world problems and can extend or complement existing classical methods. For PINNs, one of the key remaining challenges is computational efficiency; training a PINN typically takes much more computational resources than using finite difference methods or FEM. Specifically for our wave equation problem in Section 5.5, training the PINN/single-threaded FBPINN takes of the order of 10 h on a single GPU, whilst FD modelling takes of the order of 1 min on a single CPU. We noted above that the data efficiency of FBPINNs increases as the size of its subdomain network decreases, and therefore, with a small enough network size and a multi-threaded implementation, FBPINNs may be able to match the efficiency of finite difference methods or FEM. It could also be powerful to combine efforts to learn families of solutions, such as the DeepONets mentioned above, with FBPINNs, allowing multiple large-scale solutions to be learnt without needing to retrain FBPINNs. Ultimately, this may lead to approaches that are faster and more accurate than classical methods, opening up many new potential applications. We also believe that standard benchmarks should be established to allow PINNs and their wide variety of derivatives to be more robustly compared, which will help the field achieve this goal.

7 Conclusions

In this work, we presented FBPINNs, which are a scalable approach for solving large problems related to differential equations. By using a combination of domain decomposition, individual subdomain normalisation and flexible training schedules, FBPINNs are able to alleviate some of the issues observed when scaling PINNs to large domains and/or multi-scale solutions, such as the increased complexity of the optimisation problem and the spectral bias of neural networks. We found that FBPINNs were able to accurately solve both the smaller and larger scale problems studied, including those with multi-scale solutions. Furthermore, FBPINNs are more data-efficient than PINNs, and they can be trained in a parallel fashion, which could eventually allow them to become more competitive with classical approaches such as finite difference or finite element methods. In future work, we plan to study the performance of the multi-threaded version of FBPINNs, as well as adaptive subdomain refinement to further improve their accuracy and efficiency.

Appendix A: Forward inference FLOPS calculation

To compare the computational resources required to train PINNs and FBPINNs, we use a measure which we call the forward inference FLOPS. This is simply the number of FLOPS spent during forward inference of their neural network(s) (i.e. evaluating the function $NN(x;\theta)$ for PINNs or $\overline{NN}(x;\theta)$ for FBPINNs).

For a fully connected neural network with tanh activations and a linear output layer, as used here, we assume the number of flops F spent during forward inference of the network is

$$F = N((2d+6)h + (l-1)(2h+6)h + (2h+1)d_u)\ , \tag{A1}$$

where N is the number of training points input to the network (or the batch size), d is the dimensionality of the input vector, d_u is the dimensionality of the output vector, h is the number of hidden units and l is the number of hidden layers. To see this, we note that the computation of each network layer consists of a matrix-matrix multiply, addition of the bias vector, and application of the activation function. For the first layer of the network, the matrix-matrix multiply requires $2Ndh$ operations, the bias addition requires Nh operations and we assume the tanh operation requires $5Nh$ operations. Similar calculations follow for the remaining layers and hence Equation A1 follows.

For PINNs, we use Equation A1 directly to estimate the forward inference FLOPS required for each training step. For FBPINNs, the forward inference FLOPS required for each training step are calculated by summing over all n subdomain networks used in the current training step, i.e. $F = \sum_i^n F_i$.

This gives a measure of data efficiency during training. However, we note it only counts FLOPS spent during the forward inference of the networks and does not count any additional FLOPS spent during gradient computation, backpropagation or any other parts of the training algorithm.

Appendix B: Definition of Fourier features

We compare the performance of FBPINNs to PINNs with Fourier features for a number of experiments. Fourier features have been shown to help neural networks learn high-frequency functions [45] and help PINNs converge [44]. 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

$$\gamma(x) = [\cos(\Gamma x),\ \sin(\Gamma x)]\ , \tag{B2}$$

where Γ 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 Γ represent the frequency of the features, and they are typically sampled from a univariate Gaussian distribution with a mean and standard deviation denoted by μ and σ.

As shown in [45] and [44], a key choice when using Fourier features are the values of μ and σ. In particular, the value of σ significantly affects the frequency of the leading eigenvectors of the neural network's neural tangent kernel [64]. If σ is too low, the network may take a long time to learn high-frequency features, whilst if σ is too high, it may over-fit its training data. In this study, we choose μ and σ empirically by scanning over them and selecting the values which produce the lowest L1 test error.

Appendix C: Finite difference modelling for (2+1)D wave equation

When studying the (2+1)D wave equation (Section 5.5) we use finite difference modelling as the ground truth solution. The SEISMIC_CPML library [63] is used (specifically, the SEISMIC_CPML code) which performs staggered-grid second-order finite difference modelling of the time-dependent 2D acoustic wave equation using a convolutional perfectly matched layer (PML) boundary condition at the edges of the domain. For ease of use, we re-implemented the original Fortran code in Python. The simulation is initialised by sampling the spatially varying wave speed, initial wavefield and initial wavefield derivative as defined in Section 5.5 on a regular $694 \times 694 \times 1891$ grid ($x_1 \times x_2 \times t$). A high density of grid points ($\sim 5\times$ spatial Nyquist frequency) is used so that the simulation is high-fidelity. An additional 10 grid points are used to define the PML boundary, which are cropped from the final solution before comparing it to the solution from the PINN and FBPINN.

Appendix D: Scaling of computational cost with number of subdomains

A noteable aspect of FBPINNs is that, for a fixed domain, if the number of training points stays constant, the total number of forward inference FLOPS required to evaluate the FBPINN ansatz only depends on the number of free parameters (size) of each subdomain network and not on the number of subdomains.

To see this, we note that only training points within each subdomain are needed to train each subdomain network (as described by Section 4.4). Thus, as the number of subdomains is increased, assuming that the proportion of the domain covered by overlapping regions stays constant, the same number of neural network evaluations are required to evaluate the FBPINN ansatz over the entire domain. This means that only the number of free parameters (size) of each subdomain network, and not the number of subdomains, affects the total number of forward inference FLOPS required.

Acknowledgements We would like to thank the EPSRC and AIMS CDT for supporting this work. This work makes use of the BURGERS_SOLUTION code written by John Burkardt (link) and the SEISMIC_CPML library [63].

Author Contributions Contributions (CRediT taxonomy) BM: conceptualisation, formal analysis, investigation, methodology, software, validation, visualisation, writing - original draft. AM, TNM: supervision, writing - review and editing.

Funding Open access funding provided by Swiss Federal Institute of Technology Zurich. This work was funded by the UKRI EPSRC Center for Doctoral Training in Autonomous Intelligent Machines and Systems (AIMS CDT).

Data Availibility All our training/test data were generated synthetically, and all the code required to reproduce this data and our results is available here: https://github.com/benmoseley/FBPINNs.

Declarations

Competing of interests The authors declare no competing interests.

References

1. Giorgi, F.: Thirty years of regional climate modeling: where are we and where are we going next? Journal of Geophysical Research: Atmospheres **124**(11), 5696–5723 (2019). https://doi.org/10.1029/2018JD030094
2. Prein, A.F., Langhans, W., Fosser, G., Ferrone, A., Ban, N., Goergen, K., Keller, M., Tölle, M., Gutjahr, O., Feser, F., Brisson, E., Kollet, S., Schmidli, J., Van Lipzig, N.P.M., Leung, R.: A review on regional convection-permitting climate modeling: demonstrations, prospects, and challenges. Rev. Geophys. **53**(2), 323–361 (2015). https://doi.org/10.1002/2014RG000475
3. Agostinelli, S., Allison, J., Amako, K., Apostolakis, J., Araujo, H., Arce, P., Asai, M., Axen, D., Banerjee, S., Barrand, G., Behner, F., Bellagamba, L., Boudreau, J., Broglia, L., Brunengo, A., Burkhardt, H., Chauvie, S., Chuma, J., Chytracek, R., Cooperman, G., Cosmo, G., Degtyarenko, P., Dell'Acqua, A., Depaola, G., Dietrich, D., Enami, R., Feliciello, A., Ferguson, C., Fesefeldt, H., Folger, G., Foppiano, F., Forti, A., Garelli, S., Giani, S., Giannitrapani, R., Gibin, D., Gomez Cadenas, J.J., Gonzalez, I., Gracia Abril, G., Greeniaus, G., Greiner, W., Grichine, V., Grossheim, A., Guatelli, S., Gumplinger, P., Hamatsu, R., Hashimoto, K., Hasui, H., Heikkinen, A., Howard, A., Ivanchenko, V., Johnson, A., Jones, F.W., Kallenbach, J., Kanaya, N., Kawabata, M., Kawabata, Y., Kawaguti, M., Kelner, S., Kent, P., Kimura, A., Kodama, T., Kokoulin, R., Kossov, M., Kurashige, H., Lamanna, E., Lampen, T., Lara, V., Lefebure, V., Lei, F., Liendl, M., Lockman, W., Longo, F., Magni, S., Maire, M., Medernach, E., Minamimoto, K., Mora de Freitas, P., Morita, Y., Murakami, K., Nagamatu, M., Nartallo, R., Nieminen, P., Nishimura, T., Ohtsubo, K., Okamura, M., O'Neale, S., Oohata, Y., Paech, K., Perl, J., Pfeiffer, A., Pia, M.G., Ranjard, F., Rybin, A., Sadilov, S., di Salvo, E., Santin, G., Sasaki, T., Savvas, N., Sawada, Y., Scherer, S., Sei, S., Sirotenko, V., Smith, D., Starkov, N., Stoecker, H., Sulkimo, J., Takahata, M., Tanaka, S., Tcherniaev, E., Safai Tehrani, E., Tropeano, M., Truscott, P., Uno, H., Urban, L., Urban, P., Verderi, M., Walkden, A., Wander, W., Weber, H., Wellisch, J.P., Wenaus, T., Williams, D.C., Wright, D., Yamada, T., Yoshida, H., Zschiesche, D.: GEANT4 - a simulation toolkit. Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**(3), 250–303 (2003). https://doi.org/10.1016/S0168-9002(03)01368-8
4. Jasak, H.: OpenFOAM: open source CFD in research and industry. International Journal of Naval Architecture and Ocean Engineering **1**(2), 89–94 (2009). https://doi.org/10.2478/ijnaoe-2013-0011
5. Leng, K., Nissen-Meyer, T., van Driel, M., Hosseini, K., Al-Attar, D.: AxiSEM3D: broad-band seismic wavefields in 3-D global earth models with undulating discontinuities. Geophys. J. Int. **217**(3), 2125–2146 (2019). https://doi.org/10.1093/gji/ggz092
6. Nochetto, R.H., Siebert, K.G., Veeser, A.: Theory of adaptive finite element methods: an introduction. In: DeVore, R., Kunoth, A. (eds.) Multiscale, Nonlinear and Adaptive Approximation, pp. 409–542. Springer, (2009)
7. Rasp, S., Pritchard, M.S., Gentine, P.: Deep learning to represent subgrid processes in climate models. Proc. Natl. Acad. Sci. U. S. A. **115**(39), 9684–9689 (2018). arXiv:1806.04731. https://doi.org/10.1073/pnas.1810286115
8. Moseley, B., Nissen-Meyer, T., Markham, A.: Deep learning for fast simulation of seismic waves in complex media. Solid Earth **11**(4), 1527–1549 (2020). https://doi.org/10.5194/se-11-1527-2020

9. Kasim, M.F., Watson-Parris, D., Deaconu, L., Oliver, S., Hatfield, P., Froula, D.H., Gregori, G., Jarvis, M., Khatiwala, S., Korenaga, J., Topp-Mugglestone, J., Viezzer, E., Vinko, S.M.: Building high accuracy emulators for scientific simulations with deep neural architecture search. Machine Learning: Science and Technology **3**(1), 015013 (2022). arXiv:2001.08055. https://doi.org/10.1088/2632-2153/ac3ffa
10. Baker, N., Alexander, F., Bremer, T., Hagberg, A., Kevrekidis, Y., Najm, H., Parashar, M., Patra, A., Sethian, J., Wild, S., Willcox, K., Lee, S.: Workshop report on basic research needs for scientific machine learning: core technologies for artificial intelligence. Technical report, USDOE Office of Science (SC) (United States) (2019). https://doi.org/10.2172/1478744. http://www.osti.gov/servlets/purl/1478744/
11. Willard, J., Jia, X., Xu, S., Steinbach, M., Kumar, V.: Integrating scientific knowledge with machine learning for engineering and environmental systems. arXiv **1**, 1 (2020). arXiv:2003.04919
12. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks **9**(5), 987–1000 (1998).arXiv:9705023[physics]. https://doi.org/10.1109/72.712178
13. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045
14. Karniadakis, G.E., Kevrekidis, I.G., Lu, L., Perdikaris, P., Wang, S., Yang, L.: Physics-informed machine learning. Nature Reviews Physics, 1–19 (2021). https://doi.org/10.1038/s42254-021-00314-5
15. Cuomo, S., di Cola, V.S., Giampaolo, F., Rozza, G., Raissi, M., Piccialli, F.: Scientific machine learning through physics-informed neural networks: where we are and what's next. arXiv (2022). arXiv:2201.05624
16. Raissi, M., Yazdani, A., Karniadakis, G.E.: Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. Science **367**(6481), 1026–1030 (2020). https://doi.org/10.1126/science.aaw4741
17. Sun, L., Gao, H., Pan, S., Wang, J.X.: Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. Comput. Methods Appl. Mech. Eng. **361** (2020). arXiv:1906.02382. https://doi.org/10.1016/j.cma.2019.112732
18. Sahli Costabal, F., Yang, Y., Perdikaris, P., Hurtado, D.E., Kuhl, E.: Physics-informed neural networks for cardiac activation mapping. Frontiers in Physics **8**, 42 (2020). https://doi.org/10.3389/fphy.2020.00042
19. Moseley, B., Markham, A., Nissen-Meyer, T.: Solving the wave equation with physics-informed deep learning. arXiv (2020). arXiv:2006.11894
20. Cai, S., Wang, Z., Fuest, F., Jeon, Y.J., Gray, C., Karniadakis, G.E.: 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**, 102 (2021). arXiv:2103.02807. https://doi.org/10.1017/jfm.2021.135
21. Yang, L., Meng, X., Karniadakis, G.E.: B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. J. Comput. Phys. **425**, 109913 (2021). https://doi.org/10.1016/j.jcp.2020.109913
22. Kharazmi, E., Zhang, Z., Karniadakis, G.E.M.: VPINNs: variational physics-informed neural networks for solving partial differential equations. arXiv (2019). arXiv:1912.00873
23. Zhu, Y., Zabaras, N., Koutsourelakis, P.S., Perdikaris, P.: Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J. Comput. Phys. **394**, 56–81 (2019). arXiv:1901.06314. https://doi.org/10.1016/j.jcp.2019.05.024
24. Geneva, N., Zabaras, N.: Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. J. Comput. Phys. **403**, 109056 (2020). arXiv:1906.05747. https://doi.org/10.1016/j.jcp.2019.109056
25. Gao, H., Sun, L., Wang, J.X.: PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. J. Comput. Phys. **428**, 110079 (2021). arXiv:2004.13145. https://doi.org/10.1016/j.jcp.2020.110079
26. Lu, L., Jin, P., Pang, G., Zhang, Z., Karniadakis, G.E.: Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence **3**(3), 218–229 (2021). https://doi.org/10.1038/s42256-021-00302-5

27. Wang, S., Wang, H., Perdikaris, P.: Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. Science Advances **7**(40), 8605–8634 (2021). arXiv:2103.10974. https://doi.org/10.1126/sciadv.abi8605
28. Pang, G., Lu, L.U., Karniadakis, G.E.M.: FPinns: fractional physics-informed neural networks. SIAM J. Sci. Comput. **41**(4), 2603–2626 (2019). arXiv:1811.08967. https://doi.org/10.1137/18M1229845
29. Yang, L., Zhang, D., Karniadakis, G.E.: Physics-informed generative adversarial networks for stochastic differential equations. SIAM J . Sci. Comput. **42**(1), 292–317 (2020). arXiv:1811.02033. https://doi.org/10.1137/18M1225409
30. Zhang, D., Lu, L., Guo, L., Karniadakis, G.E.: Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. J. Comput. Phys. **397**, 108850 (2019). arXiv:1809.08327. https://doi.org/10.1016/j.jcp.2019.07.048
31. Chen, Z., Liu, Y., Sun, H.: Physics-informed learning of governing equations from scarce data. Nat. Commun. **12**(1), 1–13 (2021). arXiv:2005.03448. https://doi.org/10.1038/s41467-021-26434-1
32. Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: DeepXDE: a deep learning library for solving differential equations. SIAM Review **63**(1), 208–228 (2021). arXiv:1907.04502. https://doi.org/10.1137/19M1274067
33. Hennigh, O., Narasimhan, S., Nabian, M.A., Subramaniam, A., Tangsali, K., Rietmann, M., Ferrandis, J.d.A., Byeon, W., Fang, Z., Choudhry, S.: NVIDIA SimNet^{TM}: an AI-accelerated multi-physics simulation framework. arXiv (2020) arXiv:2012.07938
34. Koryagin, A., Khudorozkov, R., Tsimfer, S.: PyDEns: a Python-framework for solving differential equations with neural networks. arXiv (2019). arXiv:1909.11544
35. Chen, F., Sondak, D., Protopapas, P., Mattheakis, M., Liu, S., Agarwal, D., Di Giovanni, M.: NeuroDiffEq: a Python package for solving differential equations with neural networks. Journal of Open Source Software **5**(46), 1931 (2020). https://doi.org/10.21105/joss.01931
36. Wang, S., Teng, Y., Perdikaris, P.: Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J. Sci. Comput. **43**(5), 3055–3081 (2021). arXiv:2001.04536. https://doi.org/10.1137/20M1318043
37. Wang, S., Yu, X., Perdikaris, P.: When and why PINNs fail to train: a neural tangent kernel perspective. J. Comput. Phys. **449**, 110768 (2022). arXiv:2007.14527. https://doi.org/10.1016/j.jcp.2021.110768
38. Shin, Y., Darbon, J., Karniadakis, G.E.: On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs. Communications in Computational Physics **28**(5), 2042–2074 (2020). arXiv:2004.01806. https://doi.org/10.4208/cicp.oa-2020-0193
39. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA J. Numer. Anal. **00**, 1–43 (2022). arXiv:2006.16144. https://doi.org/10.1093/imanum/drab093
40. Xu, Z.Q.J., Zhang, Y., Luo, T., Xiao, Y., Ma, Z.: Frequency principle: Fourier analysis sheds light on deep neural networks. Communications in Computational Physics **28**(5), 1746–1767 (2020). arXiv:1901.06523. https://doi.org/10.4208/CICP.OA-2020-0085
41. Rahaman, N., Baratin, A., Arpit, D., Draxlcr, F., Lin, M., Hamprecht, F.A., Bengio, Y., Courville, A.: On the spectral bias of neural networks. In: 36th International Conference on Machine Learning, ICML 2019, vol. 2019-June, pp. 9230–9239. International Machine Learning Society (IMLS), (2019). arXiv:1806.08734
42. Basri, R., Jacobs, D., Kasten, Y., Kritchman, S.: The convergence rate of neural networks for learned functions of different frequencies. In: Advances in Neural Information Processing Systems, vol. 32. Neural information processing systems foundation, (2019). arXiv:1906.00425
43. Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X., Gu, Q.: Towards understanding the spectral bias of deep learning. IJCAI (2021). arXiv:1912.01198
44. Wang, S., Wang, H., Perdikaris, P.: On the eigenvector bias of Fourier feature networks: from regression to solving multi-scale PDEs with physics-informed neural networks. Comput. Methods Appl. Mech. Eng. **384**, 113938 (2021). arXiv:2012.10047. https://doi.org/10.1016/j.cma.2021.113938
45. Tancik, M., Srinivasan, P.P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J.T., Ng, R.: 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). https://doi.org/10.48550/arxiv.2006.10739. arXiv:2006.10739v1

46. Liu, Z., Cai, W., Xu, Z.Q.J.: Multi-scale deep neural network (MscaleDNN) for solving Poisson-Boltzmann equation in complex domains. Communications in Computational Physics **28**(5), 1970–2001 (2020). arXiv:2007.11207. https://doi.org/10.4208/CICP.OA-2020-0179
47. Heinlein, A., Klawonn, A., Lanser, M., Weber, J.: Combining machine learning and domain decomposition methods for the solution of partial differential equations–a review. GAMM-Mitteilungen **44**(1), 202100001 (2021). https://doi.org/10.1002/gamm.202100001
48. Jagtap, A.D., Karniadakis, G.E.: Extended physics-informed neural networks (XPINNs): a generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. Communications in Computational Physics **28**(5), 2002–2041 (2020). https://doi.org/10.4208/CICP.OA-2020-0164
49. Shukla, K., Jagtap, A.D., Karniadakis, G.E.: Parallel physics-informed neural networks via domain decomposition. J. Comput. Phys. **447**, 110683 (2021). arXiv:2104.10013. https://doi.org/10.1016/j.jcp.2021.110683
50. Dwivedi, V., Parashar, N., Srinivasan, B.: Distributed learning machines for solving forward and inverse problems in partial differential equations. Neurocomputing **420**, 299–316 (2021). https://doi.org/10.1016/j.neucom.2020.09.006
51. Dong, S., Li, Z.: Local extreme learning machines and domain decomposition for solving linear and nonlinear partial differential equations. Comput. Methods Appl. Mech. Eng. **387**, 114129 (2021) arXiv:2012.02895. https://doi.org/10.1016/j.cma.2021.114129
52. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1–3), 489–501 (2006). https://doi.org/10.1016/j.neucom.2005.12.126
53. Li, K., Tang, K., Wu, T., Liao, Q.: D3M: a deep domain decomposition method for partial differential equations. IEEE Access **8**, 5283–5294 (2020). arXiv:1909.12236. https://doi.org/10.1109/ACCESS.2019.2957200
54. Stiller, P., Bethke, F., Böhme, M., Pausch, R., Torge, S., Debus, A., Vorberger, J., Bussmann, M., Hoffmann, N.: Large-scale neural solvers for partial differential equations. In: Nichols, J., Verastegui, B., Maccabe, A.B., Hernandez, O., Parete-Koon, S., Ahearn, T. (eds.) Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI, pp. 20–34. Springer, (2020)
55. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: hp-VPINNs: variational physics-informed neural networks with domain decomposition. Comput. Methods Appl. Mech. Eng. **374** (2021). arXiv:2003.05385. https://doi.org/10.1016/j.cma.2020.113547
56. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: an imperative style, high-performance deep learning library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d\textquotesingle Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 8024–8035. Curran Associates, Inc., (2019). http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
57. Lagaris, I.E., Likas, A.C., Papageorgiou, D.G.: Neural-network methods for boundary value problems with irregular boundaries. IEEE Transactions on Neural Networks **11**(5), 1041–1049 (2000). https://doi.org/10.1109/72.870037
58. Berg, J., Nyström, K.: A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing **317**, 28–41 (2018). arXiv:1711.06464. https://doi.org/10.1016/j.neucom.2018.06.056
59. Leake, C., Mortari, D.: Deep theory of functional connections: a new method for estimating the solutions of partial differential equations. Machine Learning and Knowledge Extraction **2**(1), 37–55 (2020). https://doi.org/10.3390/make2010004
60. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: ICLR 2015 (2015). arXiv:1412.6980
61. Basdevant, C., Deville, M., Haldenwang, P., Lacroix, J.M., Ouazzani, J., Peyret, R., Orlandi, P., Patera, A.T.: Spectral and finite difference solutions of the Burgers equation. Computers and Fluids **14**(1), 23–41 (1986). https://doi.org/10.1016/0045-7930(86)90036-8
62. Igel, H.: Computational seismology: a practical introduction. Oxford University Press, (2017)
63. Komatitsch, D., Martin, R.: An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation. Geophysics **72**(5), 155–167 (2007)

64. Jacotécole, A., Jacotécole, J., Fédérale De Lausanne, P., Gabriel, F.: Neural tangent kernel: convergence and generalization in neural networks. In: Advances in Neural Information Processing Systems, vol. 31 (2018)