Alexander Heinlein, Amanda A. Howard, Damien Beecroft, and Panos Stinis

Multifidelity domain decomposition-based physics-informed neural networks and operators for time-dependent problems

Abstract: Multiscale problems are challenging for neural network-based discretizations of differential equations, such as physics-informed neural networks (PINNs) and operator networks. This can be (partly) attributed to the so-called spectral bias of neural networks. To improve the performance of PINNs for time-dependent problems, a combination of multifidelity stacking PINNs and domain decomposition-based finite basis PINNs is employed. In particular, to learn the high-fidelity part of the multifidelity model, a domain decomposition in time is employed. The performance is investigated for a pendulum and a two-frequency problem as well as the Allen–Cahn equation. It can be observed that the domain decomposition approach clearly improves the PINN and stacking PINN approaches. Finally, it is demonstrated that the FBPINN approach can be extended to multifidelity physics-informed deep operator networks.

Keywords: Multifidelity, domain decomposition, physics-informed neural network, deep operator networks

MSC 2020: 65M22, 65M55, 68T07

1 Introduction

Many problems arising in science and engineering exhibit a multiscale nature, with different processes taking place on various temporal and spatial scales. The solution of

Acknowledgement: This project was completed with support from the U. S. Department of Energy, Advanced Scientific Computing Research program, under the Scalable, Efficient, and Accelerated Causal Reasoning Operators, Graphs and Spikes for Earth and Embedded Systems (SEA-CROGS) project (Project No. 80278). Pacific Northwest National Laboratory (PNNL) is a multiprogram national laboratory operated for the U. S. Department of Energy (DOE) by Battelle Memorial Institute under Contract No. DE-AC05-76RL01830. The computational work was performed using PNNL Institutional Computing.

Alexander Heinlein, Delft University of Technology, Delft Institute of Applied Mathematics, 4 Mekelweg, 2628 CD Delft, South Holland, Netherlands, e-mail: a.heinlein@tudelft.nl
Amanda A. Howard, **Panos Stinis,** Battelle Memorial Institute, P. O. Box 999, Richland, WA, 99352, USA, e-mails: amanda.howard@pnnl.gov, panos.stinis@pnnl.gov
Damien Beecroft, University of Washington, Applied Mathematics, 7462 Woodlawn Ave NE, Seattle, WA, 98195, USA, e-mail: dob1998@uw.edu

https://doi.org/10.1515/9783111376776-006

these problems is generally difficult for numerical methods. Multiscale methods have been developed to make numerical simulations of such problems feasible; examples are the multiscale finite element [12], homogeneous multiscale [9], generalized finite element [4], or variational multiscale [16] methods.

In recent years, inspired by the early work by Lagaris et al. [17], machine learning-based techniques for the solution of partial differential equations (PDEs) have been developed. In this paper, we focus on physics-informed neural networks (PINNs) [25]; other methods, such as the Deep Ritz method [10], have been developed around the same time. Those methods have many potential advantages: they are easy to implement, allow for direct integration of data, and can be employed to solve inverse and high-dimensional problems. However, their convergence properties are not yet well understood, and hence the resulting accuracy is often limited. This can be partly accounted to the spectral bias of neural networks (NNs) [24], meaning that NNs tend to learn low frequency components of functions much faster than high frequency components. In multiscale problems, the high frequency components typically correspond to the fine scales, whereas the low frequency components correspond to the coarse scales. Therefore, multiscale problems are also particularly challenging to solve using PINNs.

In this paper, we aim to combine two techniques that have recently been developed to improve the training of PINNs in this context. On the one hand, we consider the multifidelity training approach introduced for PINNs [21] and extended to Deep Operator Networks (DeepONets, [18]) in [14, 19, 6]. In particular, we consider the approach of stacked PINNs [13] in which multiple networks are stacked on top of each other, such that models on top of the stack may learn those features that are not captured by the previous models. On the other hand, we employ multilevel Schwarz domain decomposition neural network architectures [8], which are based on the finite-basis PINNs (FBPINNs) [22] approach. In this approach, the learning of multiscale features is improved by localization. In particular, the network architecture is decomposed, such that the individual parts of the network learn features on the corresponding spatial or temporal scale. For an overview on the combination of domain decomposition approaches and machine learning see, for instance, [11].

In related recent works, methods for iteratively training PINNs to progressively reduce the errors have been developed; see [1, 2, 3, 31]. These approaches, which vary in their implementation details, train each new network to reduce the residual from the previous network. In contrast, the work presented here trains for the entire solution at each iteration.

This paper is structured as follows: First, in Section 2, we describe the methodological framework. In particular, we first discuss PINNs in Section 2.1, then we describe multifidelity stacking PINNs in Section 2.2, as well as the domain decomposition approach in Section 2.3. Next, we introduce the specific domain decomposition in time employed in the model problems in Section 3. In Section 4, we present numerical results for several model problems, a pendulum, and a two-frequency problem as well as the Allen–Cahn equation. We extend the results to DeepONets in Section 5. We conclude with a brief

discussion of the current and future work in Section 6. All training parameters used to generate the results are given in Table 6.1.

2 Methodology

2.1 Physics-informed neural networks

We consider a generic differential equation-based problem in residual form: Find u such that

$$\begin{aligned} \mathcal{A}u &= 0 \quad \text{in } \Omega, \\ \mathcal{B}u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{6.1}$$

where $\mathcal{A}$ is a differential operator and $\mathcal{B}$ an operator for specifying the initial or boundary conditions. The solution u is defined on the domain Ω and should have sufficient regularity to apply $\mathcal{A}$ and $\mathcal{B}$. In order to solve Eq. (6.1), we follow [17] and employ a collocation approach. In particular, we exploit that solving Eq. (6.1) is equivalent to solving $\arg\min_{\mathcal{B}u=0 \text{ on } \partial\Omega} \int_\Omega (\mathcal{A}u(\mathbf{x}))^2 \, d\mathbf{x}$. We discretize the solution using a neural network $\hat{u}(\mathbf{x}, \theta)$, with parameters θ, and the integral is approximated by the sum

$$\underset{\mathcal{B}\hat{u}(\mathbf{x},\theta)=0 \text{ on } \partial\Omega}{\arg\min} \sum_{\mathbf{x}_i \in \Omega} \left(\mathcal{A}\hat{u}(\mathbf{x}_i, \theta)\right)^2,$$

where the collocation points $\mathbf{x}_i$ are sampled from Ω. Different types of neural network architectures may be employed, and we will employ a combination of the approaches explained in Sections 2.2 and 2.3.

The initial or boundary conditions in the second equation of Eq. (6.1) can be enforced via hard or soft constraints. In the approach of hard constraints, they are explicitly implemented in the neural network function; cf. [17]. In this paper, we employ the approach of soft constraints instead, in which we incorporate the constraints into the loss function:

$$\underset{\theta}{\arg\min}\ \lambda_r \sum_{\mathbf{x}_i \in \Omega} \left(\mathcal{A}\hat{u}(\mathbf{x}_i, \theta)\right)^2 + \lambda_{bc} \sum_{\mathbf{x}_i \in \partial\Omega} \left(\mathcal{B}\hat{u}(\mathbf{x}_i, \theta)\right)^2 \tag{6.2}$$

Here, λ_r and λ_{bc} weight the residual of the differential equation and the initial and boundary conditions in the loss function, respectively. As discussed, for instance in [28], an appropriate weighting is crucial for the convergence in optimizing Eq. (6.2) using a gradient-based optimization method. θ denotes all the trainable parameters in the network.

This approach has also been denoted as physics-informed neural networks (PINNs) in [25].

2.2 Multifidelity stacking PINNs

Multifidelity PINNs use two NNs to learn the correlation between low- and high-fidelity physics [21]. The goal is to train a linear network (with no activation function) to learn the linear correlation between the low- and high-fidelity models, and a nonlinear network to learn the nonlinear correlation. By training a linear network, the resulting model is more expressive than just assuming that the correlation between the models is the identity. Moreover, under the assumption that the main part of the correlation is linear, separating the network into the linear and nonlinear parts allows for a smaller nonlinear network.

To train a multifidelity PINN, we first train a standard single fidelity PINN $\hat{u}^{\mathrm{SF}}(\mathbf{x}, \theta^{\mathrm{SF}})$. In a second step, we then train a multifidelity network $\hat{u}^{\mathrm{MF}}$, which consists of linear and nonlinear subnetworks that learn the correlation between the single fidelity PINN $\hat{u}^{\mathrm{SF}}(\mathbf{x}, \theta^{\mathrm{SF}})$ and the solution:

$$\hat{u}^{\mathrm{MF}}(\mathbf{x}, \theta^{\mathrm{MF}}) = (1 - |\alpha|)\hat{u}^{\mathrm{MF}}_{\mathrm{linear}}(\mathbf{x}, \hat{u}^{\mathrm{SF}}, \theta^{\mathrm{MF}}) + |\alpha|\hat{u}^{\mathrm{MF}}_{\mathrm{nonlinear}}(\mathbf{x}, \hat{u}^{\mathrm{SF}}, \theta^{\mathrm{MF}}). \tag{6.3}$$

The linear network does not have activation functions to force learning a linear correlation, and can be very small. α is a trainable parameter to enforce maximizing the linear correlation.

The loss function in Eq. (6.2) is modified to include the penalty α^4:

$$\arg\min_{\theta} \lambda_r \sum_{\mathbf{x}_i \in \Omega} \left(\mathcal{A}\hat{u}(\mathbf{x}_i, \theta)\right)^2 + \lambda_{bc} \sum_{\mathbf{x}_i \in \partial\Omega} \left(\mathcal{B}\hat{u}(\mathbf{x}_i, \theta)\right)^2 + \lambda_a \alpha^4 \tag{6.4}$$

In multifidelity stacking PINNs as presented in [13], multifidelity PINNs are trained recursively, each taking the output of the previously trained stacking layer as input. In this way, the previous layer serves as the low-fidelity model for the new stacking layer. The difference between [13] and the current work is that [13] does not consider domain decomposition, so each stacking layer has a single multifidelity PINN covering the entire domain. The approach considered here is more flexible, and as we will show in Section 4, results in smaller relative errors when trained on the same equations.

2.3 Domain decomposition-based neural network architectures

It has been observed in [22] that the high frequency components in the solution can be learned better if a domain decomposition is introduced into the PINN approach. To scale to larger numbers of subdomains, this approach has first been extended to two-levels in [7] and then to an arbitrary number of levels in [8]. The general idea of the domain decomposition-based finite basis PINNs (FBPINNs) is to decompose the computational domain Ω into J overlapping subdomains Ω_j, $\Omega = \bigcup_{j=1}^{J} \Omega_j$. As before, Ω may be a space-time domain, and in this work we will focus on domain decomposition in time. On each subdomain, we define a space of network functions $\mathcal{V}_j = \{\hat{u}_j(\mathbf{x}, \theta_j) \mid \mathbf{x} \in \Omega_j, \theta_j \in \Theta_j\}$,

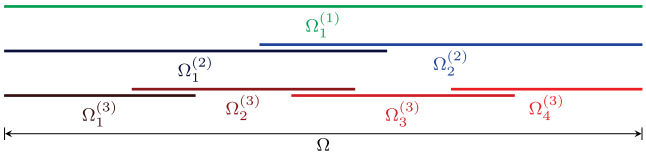


Figure 6.1: Multilevel overlapping domain decomposition of Ω with $L = 3$ levels.

where $\hat{u}_j(\mathbf{x}, \theta_j)$ denotes a PINN model, $\Theta_j = \mathbb{R}^{k_j}$ is the space of all trainable neural network parameters, and k_j is the number of network parameters.

In order to represent the global solution of a given problem, we define window functions ω_j with $\text{supp}(\omega_j) \subset \Omega_j$ such that $\{\omega_j\}_{j=1}^J$ form a partition of unity, that is, $\sum_{j=1}^J \omega_j = 1$ on Ω. Then we can define a global neural network space $\mathcal{V} = \sum_{j=1}^J \omega_j \mathcal{V}_j$, and the global FBPINN function reads $\hat{u}(\mathbf{x}, \theta) = \sum_{j=1}^J \omega_j \hat{u}_j(\mathbf{x}, \theta_j)$. It has been observed that this approach may significantly improve the performance of PINNs; cf. [22]. However, similar to classical domain decomposition methods [27], the one-level approach is not scalable to large numbers of subdomains; see [7, 8].

To improve the scalability and the performance for multiscale problems, a hierarchy of domain decompositions may be employed. Define L levels of domain decompositions, with the overlapping domain decomposition at level l denoted by $D^{(l)} = \{\Omega_j^{(l)}\}_{j=1}^{J^{(l)}}$, where $\Omega = \bigcup_{j=1}^{J^{(l)}} \Omega_j^{(l)}$ and $J^{(l)}$ is the number of subdomains at level l; cf. Figure 6.1. Even though there is generally no restriction on the overlapping domain decompositions, we choose $J^{(1)} = 1$, so the first level corresponds to a single global subdomain, and $J^{(l)} < J^{(l+1)}$ for all $l = 1, \ldots, L$.

Now, on each level l we define window functions $\omega_j^{(l)}$ to be a partition of unity, so $\sum_{j=1}^{J^{(l)}} \omega_j^{(l)} = 1$, and $\text{supp}(\omega_j^{(l)}) \subset \Omega_j^{(l)}$. Similar to the one-level case, this yields the global neural network space $\mathcal{V} = \sum_{l=1}^L \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \mathcal{V}_j^{(l)}$ and the global network function defined in terms of $\theta = \bigcup_{l=1}^L \theta^{(l)}$ and $\theta^{(l)} = \bigcup_{j=1}^{J^{(l)}} \theta_j^{(l)}$:

$$\hat{u}(\mathbf{x}, \theta) = \frac{1}{L} \sum_{l=1}^{L} \hat{u}^{(l)}(\mathbf{x}, \theta^{(l)}) \quad \text{with } \hat{u}^{(l)}(\mathbf{x}, \theta^{(l)}) = \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \hat{u}_j^{(l)}(\mathbf{x}, \theta_j^{(l)}). \tag{6.5}$$

It has been observed in [7, 8] that due to increased communication between the subdomain models, the multilevel FBPINN approach may significantly improve the performance over the one-level approach.

2.4 Stacking FBPINNs

We combine the multifidelity stacking PINNs and the FBPINNs as follows: In the first level, we train a standard single fidelity PINN across the full domain $\Omega^{(0)} = \Omega$. Then, for each level $l > 0$, we use a FBPINN network architecture modified to consist of multifi-

delity networks that takes as input the network from the previous level $l-1$:

$$\hat{u}^{(l)}(\mathbf{x}, \theta^{(l)}) = \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} \hat{u}_{j,MF}^{(l)}(\mathbf{x}, \hat{u}^{(l-1)}, \theta_j^{(l)}). \tag{6.6}$$

We note that Eq. (6.6) differs from Eq. (6.5) by a factor of $1/L$, because the output of FBPINNs is the sum of the networks trained at all levels, while the output of stacking FBPINNs is the sum of the networks for the final level. The networks at level l learn the correlation between the output of the $l-1$ level and the solution, and take as input the previously learned solution $\hat{u}^{(l-1)}(\mathbf{x}, \theta^{(l-1)})$:

$$\hat{u}_{j,MF}^{(l)}(\mathbf{x}, \theta_j^{(l)}) = (1-|\alpha|)\hat{u}_{j,\text{lin}}^{(l)}(\mathbf{x}, \hat{u}^{(l-1)}, \theta_j^{(l)}) + |\alpha|\hat{u}_{j,\text{nonlin}}^{(l)}(\mathbf{x}, \hat{u}^{(l-1)}, \theta_j^{(l)}).$$

3 Domain decomposition in time

In this work, we are particularly interested in cases where classical PINNs fail to learn the temporal evolution, such as a damped pendulum and the Allen–Cahn equation. We consider a domain $\Omega = \mathbf{X} \times [0, T]$ where $\mathbf{X}$ denotes the spatial domain and $T \in \mathbb{R}$. Therefore, for the stacking FBPINN approach, we consider the domain decomposition in time:

$$\Omega_j^{(l)} = \left[\frac{(j-1)T - \delta T/2}{J^{(l)} - 1}, \frac{(j-1)T + \delta T/2}{J^{(l)} - 1} \right],$$

where $\delta > 1$ is the overlap ratio. For $l = 0$, we take $\Omega_1^{(0)} = [0.5T - \delta T/2, 0.5T + \delta T/2]$. The partition of unity functions are given by $\omega_j^{(l)} = \frac{\hat{\omega}_j^{(l)}}{\sum_{j=1}^{J^{(l)}} \hat{\omega}_j^{(l)}}$, where

$$\hat{\omega}_j^{(l)}(t) = \begin{cases} 1 & l = 0, \\ [1 + \cos(\pi(t - \mu_j^{(l)})/\sigma_j^{(l)})]^2 & l > 0, \end{cases} \tag{6.7}$$

$\mu_j^{(l)} = T(j-1)/(J^{(l)} - 1)$, and $\sigma_j^{(l)} = (\delta T/2)/(J^{(l)} - 1)$. For simplicity, we take $J^{(l)} = 2^l$ in each case. An illustration of the window functions for $T = 1$ and $l = 2$ ($J^{(2)} = 4$) is given in Figure 6.2.

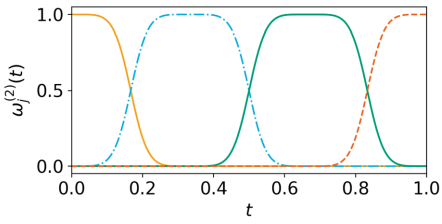


Figure 6.2: Window functions ω_j for $l = 2$ and $T = 1$.

As set up, each network only covers a small part of the time domain. To ease training, we scale the input in each domain to be in the range $[-1, 1]$ by using a scaled time $\hat{t} = t(l-1)/T - j$ as the input to network j. This scaling improves the robustness of the training.

In our applications, we calculate the relative ℓ_2 error $\frac{\|u(\mathbf{x}) - \hat{u}(\mathbf{x}, \theta)\|_2}{\|u(\mathbf{x})\|_2}$ where u denotes the exact solution and $\hat{u}$ denotes the output from the multifidelity FBPINN.

4 Results

4.1 Pendulum

While a relatively simple system, accurately training a PINN to predict the movement of a pendulum for long times presents challenges [29]. The pendulum movement is governed by a system of two first-order ODEs for $t \in [0, T]$,

$$\frac{ds_1}{dt} = s_2, \tag{6.8}$$

$$\frac{ds_2}{dt} = -\frac{b}{m} s_2 - \frac{g}{L} \sin(s_1), \tag{6.9}$$

where s_1 and s_2 are the position and velocity of the pendulum, respectively. We employ the same parameters used in [29], that is, $m = L = 1$, $b = 0.05$, $g = 9.81$, and $T = 20$. We take $s_1(0) = s_2(0) = 1$. We compare the results with those for the stacking PINN from [13], which uses the same multifidelity architecture but only a single PINN on each level. As shown in Figure 6.3, the stacking FBPINN is able to reach a significantly lower relative ℓ_2 error. In addition, each network in the stacking FBPINN is significantly smaller than the networks used in the stacking PINN with the result that, at three stacking layers, the stacking FBPINN reaches a relative ℓ_2 error of $7.4 \cdot 10^{-3}$ with only 34 570 trainable parameters. In comparison, the best case stacking PINN from [13] requires four stacking levels to reach a relative ℓ_2 error of $1.3 \cdot 10^{-2}$ with 63 018 trainable parameters.

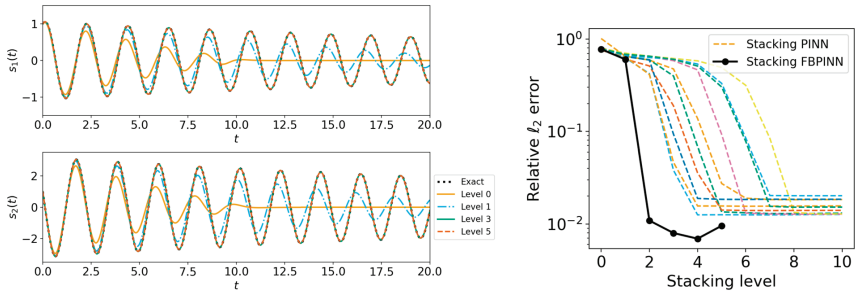


Figure 6.3: Stacking FBPINN results for the pendulum problem: **Left:** Stacking FBPINN results for an illustrative example of s_1 (top) and s_2 (bottom) as a function of time for the pendulum problem up to five stacking FBPINN levels. **Right:** Pendulum relative ℓ_2 training errors comparing the work in the current paper (solid line) with the approach from [13] (dashed lines).

4.2 Multiscale problem

We now consider a toy model problem with a low and high frequency component, inspired by [22]:

$$\begin{aligned} \frac{ds}{dx} &= \omega_1 \cos(\omega_1 x) + \omega_2 \cos(\omega_2 x), \\ s(0) &= 0, \end{aligned}$$

on domain $\Omega = [0, 20]$ with $\omega_1 = 1$ and $\omega_2 = 15$. The exact solution for this problem is $s(x) = \sin(\omega_1 x) + \sin(\omega_2 x)$.

The results are shown in Figure 6.4. After two stacking levels, the stacking FBPINN reaches a relative ℓ_2 error of $4.2 \cdot 10^{-3}$, with 7 822 trainable parameters. A comparable relative ℓ_2 error of $6.1 \cdot 10^{-3}$ is reached after 10 stacking levels with a stacking PINN with 11 179 trainable parameters. Also shown in Figure 6.4 (right) is the best case SF network from [13], which has a relative ℓ_2 error of $9.5 \cdot 10^{-2}$ with 16 833 trainable parameters. The stacking FBPINN outperforms the SF PINN with an error more than an order of magnitude lower, with less than half the trainable parameters. Additionally, the final stacking FBPINN reaches a relative ℓ_2 error of $8.3 \cdot 10^{-4}$, an order of magnitude lower than the final stacking PINN.

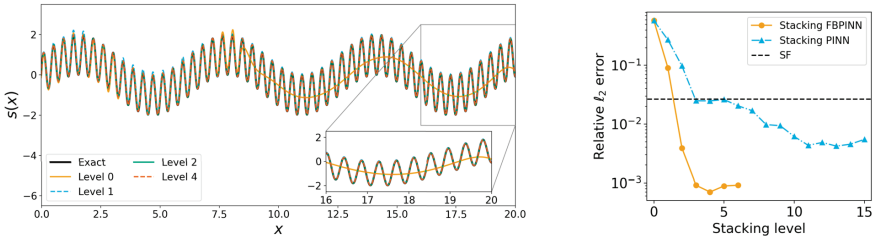


Figure 6.4: Stacking FBPINN results for the multiscale problem: **Left:** Stacking FBPINN results for the single fidelity level 0 and the first four stacking FBPINN levels. **Right:** Multiscale relative ℓ_2 training errors comparing the work in the current paper with [13].

4.3 Allen–Cahn equation

Our third example is based on the Allen–Cahn equation and is given by

$$\begin{aligned} s_t - 0.0001 s_{xx} + 5s^3 - 5s = 0, \qquad & t \in (0,1], x \in [-1,1], \\ s(x,0) = x^2 \cos(\pi x), \qquad & x \in [-1,1], \\ s(x,t) = s(-x,t), \qquad & t \in [0,1], x = -1, x = 1, \\ s_x(x,t) = s_x(-x,t), \qquad & t \in [0,1], x = -1, x = 1. \end{aligned}$$

The Allen–Cahn equation presents difficulties for PINNs when attempting to learn the full solution from $t = 0$ to 1 with a single PINN; see, for example, [32, 20, 26].

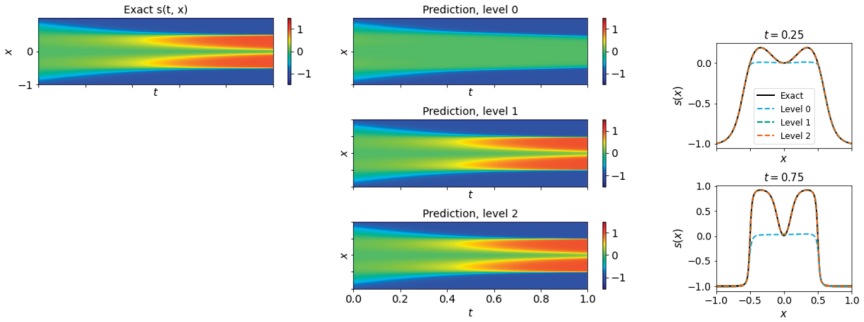


Figure 6.5: Stacking FBPINN results for the Allen–Cahn equation. **Left:** Stacking FBPINN results for the single fidelity level 0 and the first two stacking FBPINN levels. **Right:** Line plots of the results from the stacking FBPINN at $t = 0.25$ (top) and $t = 0.75$ (bottom).

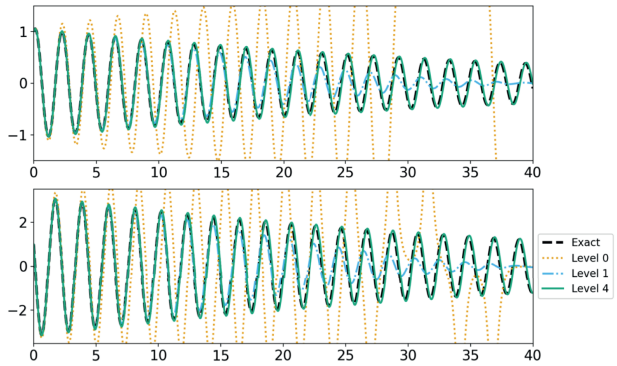
We solve the Allen–Cahn equation by dividing the time domain into subdomains, as presented in Section 3. The corresponding results for the stacking FBPINN are shown in Figure 6.5. The relative ℓ_2 error for applying two levels of the stacking FBPINN is $5.9 \cdot 10^{-3}$. Previously reported values for the relative error in literature include $1.68 \cdot 10^{-2}$ for the backward compatible PINN [20] and $2.33 \cdot 10^{-2}$ for PINNs with adaptive resampling [32].

5 Extension to DeepONets

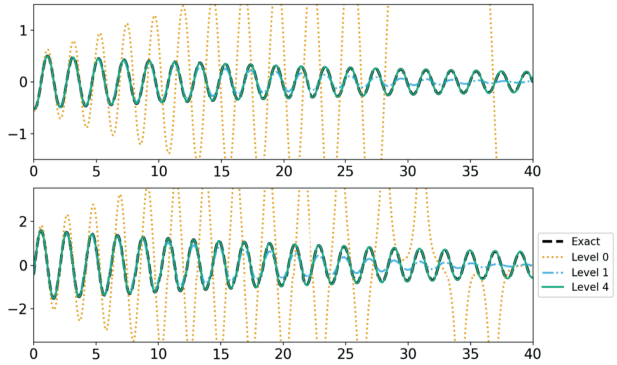
The method presented in Section 2 can be extended seamlessly to multifidelity stacking DeepONets from [14, 13]; we denote the resulting method as finite-basis DeepONets (FB-DONs). For the sake of brevity, we refer to [18, 14, 13] for details on the DeepONet approach. As an example, we present results for the pendulum problem in Section 4.1 and train a model mapping given initial conditions $(s_1(0), s_2(0))$ to the corresponding solution $(s_1(t), s_2(t))$ on the whole time interval $[0, 20]$. This is referred to as operator learning since we learn a mapping between the initial conditions and the solution space instead of a single solution. One each level l, $l > 0$, we train 2^l DeepONets with partition of unity functions as defined in Eq. (6.7). As training data, we employ 50 000 randomly chosen pairs $(s_1(0), s_2(0)) \in [-2, 2] \times [-1.2, 1.2]$, and the loss is given by Eq. (6.2) and the differential equations in Eqs. (6.8) and (6.9). After training, the resulting FB-DON model is then able to predict the solution for any initial condition in the training range, as shown in Figure 6.6. Training parameters are given in Table 6.2.

6 Discussion

In this paper, we have introduced the stacking FBPINN and FB-DON approaches. For the considered time-dependent problems, stacking FBPINNs yielded more accurate results than stacking PINNs alone, and in some cases, they additionally required fewer



(a) $s_1(0) = 1, s_2(0) = 1$



(b) $s_1(0) = -0.5, s_2(0) = -0.4$

Figure 6.6: Stacking FB-DON results for the pendulum system at two different sets of initial conditions.

total trainable parameters. This indicates that a domain decomposition in time can greatly improve the performance of stacking PINNs. In contrast to prior work on stacking PINNs and DeepONets, stacking FBPINNs and FB-DONs use a sum of subdomain networks weighted by the partition of unity functions on the corresponding level. In contrast to multilevel FBPINNS in [8], in which the subdomain networks are summed across all levels and trained simultaneously, the architecture and training of stacking FBPINNs and FB-DONs is sequential with respect to the levels; the idea is similar to multiplicative coupling as discussed in [7] but implemented differently using the stacking approach. This difference allows for stacking FBPINNs and FB-DONs to consider different equations on different levels, akin to simulated annealing, as considered in [13], or to consider different physical models at different length scales. We leave this for future work. The extension to stacking FB-DONs allows for use of physics-informed FB-DONs as surrogate models in place of traditional numerical solvers. The computation of a solution using a trained stacking FB-DONs is very efficient: it requires only one forward pass of the networks and, therefore, the computational time compared with classical

numerical solvers can be greatly reduced. One advantage of the FB-DON approach is that it can be used in conjunction with existing methods for increasing accuracy of physics-informed DeepONets, including long-time integration [29] and adaptive weighting schemes [30, 15, 23].

7 Training parameters

Table 6.1: Training parameters for the FBPINN results in the paper. The learning rate is set using the `exponential_decay` function in Jax [5] with the given learning rate and decay rate and 2 000 decay steps. The training parameters used for the stacking PINN results are given in [13].

	Section 4.1	**Section 4.2**	**Section 4.3**
Level 0 learning rate & decay rate	$5 \cdot 10^{-3}, 0.99$	$10^{-3}, 0.99$	$10^{-4}, 0.99$
Level 0 network width	100	32	100
Level 0 network layers	3	3	6
Level 0 iterations	200 000	200 000	200 000
Nonlinear network width	32	16	200
Nonlinear network layers	3	4	4
Linear network size	[2, 4, 2]	[1, 5, 1]	[1, 5, 1]
MF learning rate & decay rate	$5 \cdot 10^{-3}, 0.99$	$5 \cdot 10^{-3}, 0.95$	$5 \cdot 10^{-3}, 0.95$
BC batch size	1	1	128
Residual batch size	400	400	1024
Iterations	200 000	300 000	300 000
$\lambda_r, \lambda_{bc}, \lambda_a$	1.0, 1.0, 1.0	10.0, 1.0, 1.0	$10.0, 1.0, 10^{-5}$
Level 0 activation function	swish	swish	tanh
MF activation function	swish	swish	swish

Table 6.2: Training parameters for the FBDeepONet results in the paper. The learning rate is set using the `exponential_decay` function in Jax [5] with the given learning rate and decay rate and 2 000 decay steps.

	Section 5
Level 0 learning rate & decay rate	$5 \cdot 10^{-3}, 0.9$
Level 0 branch and trunk width	100
Level 0 branch and trunk layers	5
Level 0 iterations	100 000
Nonlinear branch and trunk width	100
Nonlinear branch and trunk layers	3
Linear branch and trunk width	10
Linear branch and trunk layers	1
MF learning rate & decay rate	$5 \cdot 10^{-3}, 0.9$
BC batch size	1 000
Residual batch size	10 000
Iterations	200 000
$\lambda_r, \lambda_{bc}, \lambda_a$	1.0, 1.0, 1.0
Level 0 activation function	sin
MF activation function	sin

Bibliography

[1] M. Ainsworth and J. Dong. Galerkin neural networks: a framework for approximating variational equations with error control. *SIAM Journal on Scientific Computing*, 43(4):A2474–A2501, 2021.

[2] M. Ainsworth and J. Dong. Galerkin neural network approximation of singularly-perturbed elliptic systems. *Computer Methods in Applied Mechanics and Engineering*, 402:115169, 2022.

[3] Z. Aldirany, R. Cottereau, M. Laforest, and S. Prudhomme. Multi-level neural networks for accurate solutions of boundary-value problems. arXiv:2308.11503, 2023.

[4] I. Babuška and J. E. Osborn. Generalized finite element methods: their performance and their relation to mixed methods. *SIAM Journal on Numerical Analysis*, 20(3):510–536, June 1983.

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[6] S. De, M. Reynolds, M. Hassanaly, R. N. King, and A. Doostan. Bi-fidelity modeling of uncertain and partially unknown systems using deeponets. arXiv:2204.00997, 2022.

[7] V. Dolean, A. Heinlein, S. Mishra, and B. Moseley. Finite basis physics-informed neural networks as a Schwarz domain decomposition method. arXiv:2211.05560, November 2022.

[8] V. Dolean, A. Heinlein, S. Mishra, and B. Moseley. Multilevel domain decomposition-based architectures for physics-informed neural networks. arXiv:2306.05486 [cs, math], June 2023.

[9] W. E, B. Engquist, X. Li, W. Ren, and E. Vanden-Eijnden. Heterogeneous multiscale methods: a review. *Communications in Computational Physics*, 2(3):367–450, June 2007.

[10] W. E and B. Yu. The Deep Ritz Method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, March 2018.

[11] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber. Combining machine learning and domain decomposition methods for the solution of partial differential equations—a review. *GAMM-Mitteilungen*, 44(1):e202100001, 2021, 28 pp.

[12] T. Y. Hou and Y. Efendiev. *Multiscale Finite Element Methods: Theory and Applications*. Springer, New York, NY, 2009.

[13] A. A. Howard, S. H. Murphy, S. E. Ahmed, and P. Stinis. Stacked networks improve physics-informed training: applications to neural networks and deep operator networks. *Foundations of Data Science*, 7(1):134–162, March 2025.

[14] A. A. Howard, M. Perego, G. E. Karniadakis, and P. Stinis. Multifidelity deep operator networks for data-driven and physics-informed problems. *Journal of Computational Physics*, 493:112462, November 2023.

[15] A. A. Howard, S. Qadeer, A. W. Engel, A. Tsou, M. Vargas, T. Chiang, and P. Stinis. The conjugate kernel for efficient training of physics-informed deep operator networks. In *ICLR 2024 Workshop on AI4DifferentialEquations in Science*, 2024.

[16] T. J. R. Hughes. Multiscale phenomena: Green's functions, the Dirichlet-to-Neumann formulation, subgrid scale models, bubbles and the origins of stabilized methods. *Computer Methods in Applied Mechanics and Engineering*, 127(1):387–401, November 1995.

[17] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, September 1998. Conference Name: IEEE Transactions on Neural Networks.

[18] L. Lu, P. Jin, and G. E. Karniadakis. DeepONet: learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, March 2021. arXiv:1910.03193.

[19] L. Lu, R. Pestourie, S. G. Johnson, and G. Romano. Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport. *Physical Review Research*, 4(2):023210, 2022.

[20] R. Mattey and S. Ghosh. A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations. *Computer Methods in Applied Mechanics and Engineering*, 390:114474, 2022.

[21] X. Meng and G. E. Karniadakis. A composite neural network that learns from multi-fidelity data: application to function approximation and inverse PDE problems. *Journal of Computational Physics*, 401:109020, 2020.

[22] B. Moseley, A. Markham, and T. Nissen-Meyer. Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *Advances in Computational Mathematics*, 49(4):62, July 2023.

[23] S. Qadeer, A. Engel, A. Tsou, M. Vargas, P. Stinis, and T. Chiang. Efficient kernel surrogates for neural network-based regression. arXiv:2310.18612, 2023.

[24] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. A. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. arXiv:1806.08734, May 2019.

[25] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[26] F. M. Rohrhofer, S. Posch, C. Gößnitzer, and B. C. Geiger. On the role of fixed points of dynamical systems in training physics-informed neural networks. arXiv:2203.13648, 2022.

[27] A. Toselli and O. Widlund. *Domain Decomposition Methods—Algorithms and Theory*. Springer Series in Computational Mathematics, volume 34. Springer, Berlin, 2005.

[28] S. Wang, X. Yu, and P. Perdikaris. When and why PINNs fail to train: a neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, January 2022.

[29] S. Wang and P. Perdikaris. Long-time integration of parametric evolution equations with physics-informed DeepONets. *Journal of Computational Physics*, 475:111855, 2023.

[30] S. Wang, H. Wang, and P. Perdikaris. Improved architectures and training algorithms for deep operator networks. *Journal of Scientific Computing*, 92(2):35, 2022.

[31] Y. Wang and C.-Y. Lai. Multi-stage neural networks: function approximator of machine precision. arXiv:2307.08934, 2023.

[32] C. L. Wight and J. Zhao. Solving Allen–Cahn and Cahn–Hilliard equations using the adaptive physics informed neural networks. arXiv:2007.04542, 2020.