Multilevel and distributed Physics-Informed Neural Networks for the Helmholtz equation

Stéphane Lanteri, Daria Hrebenshchykova, Victor Michel-Dansac, Victorita Dolean

Multilevel and Distributed Physics-Informed Neural Networks for the Helmholtz Equation

Daria Hrebenshchykova, Victorita Dolean,
Victor Michel-Dansac, Stephane Lanteri

RESEARCH REPORT
N° 9571
December 2024
Project-Teams ATLANTIS

ISSN 0249-6399 ISRN INRIA/RR--9571--FR+ENG

Multilevel and Distributed Physics-Informed Neural Networks for the Helmholtz Equation

Daria Hrebenshchykova*, Victorita Dolean†,

Victor Michel-Dansac ‡, Stephane Lanteri §

Project-Teams ATLANTIS

Research Report n° 9571 — December 2024 — 27 pages

Abstract: This report investigates the use of Physics-Informed Neural Networks (PINNs) and their advanced extensions — Finite Basis PINNs (FBPINNs) and Multilevel Finite Basis PINNs (MFBPINNs) — for solving the Helmholtz equation in one and two dimensions. Through numerical simulations, we compare the efficiency, accuracy, and scalability of these methods. While FBPINNs exhibit good performance for low-frequency problems, the multilevel method outperforms, especially for high-frequency wave propagation. This work also introduces neural network architectures and training schemes, including single and multiple optimizers, integrated into the ScimBa library. The results demonstrate the advantages of the multilevel approach for solving complex wave propagation problems in computational mathematics and engineering.

Key-words: Helmholtz equation, Physics-Informed Neural Networks, multilevel decomposition, deep learning

* Université Côte d'Azur, Inria, CNRS, LJAD, France
† Eindhoven University of Technology, Netherlands
‡ Inria Nancy - Grand Est, IRMA, France
§ Université Côte d'Azur, Inria, CNRS, LJAD, France

RESEARCH CENTRE
Centre Inria d'Université Côte d'Azur
2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Réseaux neuronaux multi-niveaux et distribués pour l'équation de Helmholtz

Résumé : Ce rapport étudie l'utilisation des réseaux neuronaux informés par la physique (PINN) et de leurs extensions avancées — les PINN à base finie (FBPINN) et les PINN à base finie multi-niveau (MFBPINN) — pour résoudre l'équation de Helmholtz en une et deux dimensions. Grâce à des simulations numériques, nous comparons l'efficacité, la précision et l'évolutivité de ces méthodes. Alors que les FBPINN présentent de bonnes performances pour les problèmes à basse fréquence, la méthode multiniveau surpasse, en particulier pour la propagation d'ondes à haute fréquence. Ce travail présente également des architectures de réseaux neuronaux et des schémas de formation, notamment des optimiseurs simples et multiples, intégrés dans la bibliothèque ScimBa. Les résultats démontrent les avantages de l'approche multiniveau pour résoudre des problèmes complexes de propagation d'ondes en mathématiques computationnelles et en ingénierie.

Mots-clés : équation de Helmholtz, réseaux de neurones informés par la physique, décomposition multiniveau, apprentissage profond

Contents

1 Introduction

The accurate simulation of wave propagation is crucial in various fields of science and engineering, ranging from acoustics to electromagnetism and seismic analysis. Numerical techniques such as finite element or finite difference methods are commonly used to solve the partial differential equations (PDEs) governing wave propagation, e.g. the Helmholtz equation. However, these approaches can become quite computationally challenging, especially in complex two-dimensional settings, when dealing with high-frequency waves.
Lately, there has been a focus on creating surrogate models that can effectively estimate the solutions to the underlying PDE problem, requiring less computational effort while still providing accurate results. Among these approaches, Physics-Informed Neural Networks (PINNs) have emerged as a promising solution. In this work, we focus on extending the PINNs framework to a multilevel and distributed architecture, specifically applied to the Helmholtz equation in the frequency domain.
The primary goal of this work is to develop and assess the performance of Finite Basis Physics-Informed Neural Networks (FBPINNs) and Multilevel Finite Basis PINNs, which aim to overcome these limitations by decomposing the computational domain into smaller subdomains and use neural network architectures. This report presents the numerical assessment of these NN-based physics-informed multilevel surrogate models, evaluating their accuracy, efficiency and scalability for the Helmholtz equation in one- and two-dimensional cases.
Throughout the following sections, the theoretical foundations of PINNs, FBPINNs and MFBPINNs for the Helmholtz equation will be discussed, followed by a detailed description of the developed methodology, numerical experiments and the resulting performance analysis. This work contributes to advancing neural network-based modeling approaches for high-frequency wave-propagation problems, in particular PINN-based approaches.

1.1 Environment

A significant part of the research involved implementing methods within the ScimBa library [1]. Based on the existing implementation of PINNs method for the Helmholtz equation, we have developed and investigated classical FBPINNs and MFBPINNs in several different variants.

2 Theoretical Foundations of Methods and Problems

2.1 PINNs

Before we define the method, let us introduce the concept of neural network. Here, we present the definition of FBPINNs, following the methodology in [3]. For completeness, we summarize the primary stages of the derivation. In our case, we perceive a neural network as a function with some changing parameters

$$u(\mathbf{x}, \boldsymbol{\theta}) : \mathbb{R}^{d_x} \times \mathbb{R}^{d_\theta} \to \mathbb{R}^{d_u}$$

where $\mathbf{x}$ are some inputs to the network, $\boldsymbol{\theta}$ are a set of learnable parameters, and d_x, d_θ and d_u are the dimensions of the inputs, parameters, and outputs of the network.

The goal is to train the model by adjusting its parameters to fit the training data. This involves minimizing a loss function that measures the difference between the model's predictions and the actual outputs in the training set. The model updates its set of learnable parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}, \ldots, \boldsymbol{\theta_i}, \ldots, \boldsymbol{\theta_n})$ to reduce this error over time.

We solely use feedforward fully connected networks (FCNs). In this case, the network function is given by

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

where now $\mathbf{x} \in \mathbb{R}^{d_x}$ is the input to the FCN, $u \in \mathbb{R}^{d_n}$ is the output of the FCN, n is the number of layers (depth) of the FCN and $f_i(\mathbf{x}, \boldsymbol{\theta}) = \sigma_i(W_i\mathbf{x} + \mathbf{b}_i)$ where $\boldsymbol{\theta}_i = (W_i, \mathbf{b}_i)$, $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ are known as weight matrices, $\mathbf{b}_i \in \mathbb{R}^{d_i}$ are known as bias vectors, σ_i are elementwise activation functions are commonly chosen as sine, hyperbolic tangent or identity functions. Note that the nonlinear activation functions σ_i consume that the network is not linear.

Now that we have introduced all the necessary notations for the neural network, we can begin to define the method itself. Physics-informed neural networks (PINNs) are NN-based approaches which

help to solve various differential equations. we focus on solving a boundary problem without initial conditions (i.e. a time-independent problem) in the form

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

where $\mathcal{N}[u](x)$ is some differential operator, $u(x)$ is the solution and $\mathcal{B}_k(\cdot)$ is a set of boundary conditions (BCs) that ensure the uniqueness of the solution. Equations (2) can describe many linear and non-linear problems. In our case, we concentrate on the Helmholtz problem which is given by

$$-\Delta u - k^2 u = f$$

where k is a wave number and function f is a source term. Details of our problems we introduce later in Chapter 3.1

To solve eq.(2), PINNs use a neural network to directly approximate the solution, i.e., $u(x, \theta) \approx u(x)$. Note that, for simplicity throughout this work, we use the same notation for both the true solution and the neural network. The main advantages of PINNs are that they are mesh-free approaches and, unlike conventional numerical methods, we end up with a functional approximation rather than a discretized solution. Based on, [4] the following loss function is minimized to train the PINN,

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

where $\{x_i\}_{i=1}^{N_I}$ is a set of collocation points sampled in the interior of the domain, $\{x_j^k\}_{j=1}^{N_B^k}$ is a set of points sampled along each boundary condition, λ_I and λ_B are well-chosen scalar weights that ensure the terms in the loss function is well balanced. It is also important to select collocation points for the residual N_I and for the boundary conditions N_B^k in such a way that the PINN can learn and train the solution over the entire domain.

Along with that, it is important to notice that the minimization of the loss function refers to using an optimization technique to adjust the network's parameters in such a way that the loss function's value is reduced. The optimization algorithms work by updating the parameters (weights and biases) of the network to minimize the loss. A commonly used option is to apply Stochastic gradient descent (SGD) methods, such as the Adam optimizer, or quasi-Newton methods, such as the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm.

For these methods, it is essential to compute the gradients of both the network output with respect to its inputs and the loss function with respect to the network parameters. These gradients obtained via automatic differentiation in modern Deep Learning libraries are then used to update the network's parameters and minimize the loss function.

2.2 Hard-constrained PINNs

As we note in eq.(3) the BC are softly enforced. The learned solution may deviate from the BCs, because the BC term may not be fully minimized. To deal with this issue, we can enforce BCs in a *hard fashion* by incorporating the neural network into a solution ansatz. More precisely, the solution to the differential equation is instead approximated by $[\mathcal{C}u](\mathbf{x}, \theta) \approx u(\mathbf{x})$, where $\mathcal{C}$ is an appropriately selected constraining operator that analytically enforces the BCs.

For instance, if we want to enforce $u(x = 0) = 0$ when solving a one-dimensional ordinary differential equation (ODE), the constraining operator and solution ansatz could be chosen as $[\mathcal{C}u](x, \theta) = x(1 - x)$. Then the BCs are always satisfied and therefore the BC term in the loss function (3) can be removed. This allows the PINN to be trained using the simpler unconstrained loss function

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{N}[\mathcal{C}u](x_i, \theta) - f(x_i)\right)^2, \tag{4}$$

where $\{x_i\}_{i=1}^{N}$ is a set of collocation points sampled in the interior of the domain.

Note that of course this is not the only way to build a constraining operator, nevertheless there is no single and universal method for building one. This can be a rather complex and difficult task.

2.3 FBPINNs

The main reason we are moving forward in our research is because starting from a certain point, if we increase the frequency and add the multi-scale features to the solution, the PINN method fails to solve accurately the problem at hand. An intuitive continuation could be adding more neural networks.

For these problems, we use the already known *domain decomposition* method, but in the context of neural networks. The main idea is that instead of one neural network that trains and searches for a solution in the entire domain, we can divide the main domain into several small overlapping domains and place a neural networks in each of them.

Thus, one neural network will be forced to solve a problem with low frequency and will be able to cope with it better. The input coordinates of each network xx are normalized to the range [-1,1] within their respective subdomains, which helps maintain numerical stability, especially when dealing with subdomains of varying sizes, orientations, or physical properties. We can clearly see how this works in Figure 1.

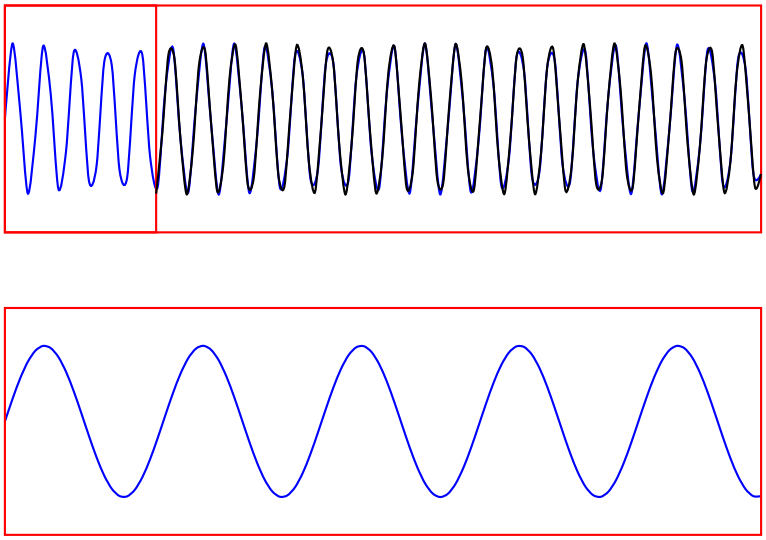


Figure 1: FBPINNs divisions for high to low frequency

2.3.1 Description of FBPINNs

This section defines FBPINNs by following [3]. For the sake of completeness, we recall the main steps of the derivation. We now present the mathematical formulation of FBPINNs. First we have to split our main domain Ω where the problem is defined into J subdomains $\{\Omega_j\}_{j=1}^{J}$. As previously mentioned, each of the subdomains will have a certain overlap with the others to get a smoother and more optimized solution, avoiding issues related to subdomain boundaries, because without overlap, the subdomain boundaries could introduce discontinuities or errors, as the networks might not have sufficient information about their neighbors.

For each subdomain Ω_j, we define a corresponding set of neural networks

$$V_j = \{v_j(x, \theta_j) \mid x \in \Omega_j, \theta_j \in \Theta_j\},$$

where $v_j(x, \theta_j)$ represents the neural network associated with subdomain Ω_j and the parameter set $\Theta_j = \mathbb{R}^{d_{\theta_j}}$ defines the trainable parameters of the network.

Each neural network within a subdomain is confined by a window function $\omega_j(x)$, with $\text{supp}(\omega_j) \subset \Omega_j$. This ensures that the network's influence is restricted to its assigned subdomain, even though the neural networks in V_j could have a global support. The role of the window functions is to limit each network to its assigned subdomain. Importantly, these window functions must form a partition of unity, i.e.

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

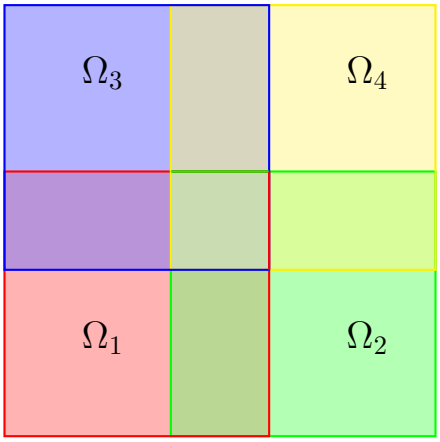


Figure 2: example of overlap in the 2D case with a square subdomain and a rectangular decomposition.

With our definitions, now we can create a global space decomposition V as

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

Thanks to this, we can finally represent any function $u \in V$ as a sum of neural networks multiplied by a window function.

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

Therefore u is the FBPINNs's solution to the problem that we want to solve.

To deal with the boundary problem we can follow a similar procedure which is described in Section 2.1 using the eq.(5) but in this work, we use hard-constrained boundary conditions which can be similarly described as in Section 2.2. By combining the principle of hard-constrained BCs, loss function and our formulation of the FBPINNs's solution, the loss function takes the form

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{N} \left[\mathcal{C} \sum_{j=1}^{J} \omega_j v_j \right] (x_i, \theta_j) - f(x_i) \right)^2. \tag{6}$$

2.3.2 Single and multiple optimizers

It is now important to discuss the aspects of optimization of the loss function. As already mentioned at the end of Section 2.1, we can use various techniques, such as gradient descent, to minimize the loss function. In essence, FBPINNs works in the same way: the solution eq.(5) is substituted into the loss function eq.(3) and the same iterative optimization scheme is used to learn the parameters $\{\theta_j\}_{j=1}^{J}$ of each subdomain network.

But from the technical side viewpoint, we realized that several different optimization techniques can be used here, namely, we can use a single optimizer for all networks or define separate optimizers for each neural network. Next, we discuss this in more detail, compare and determine which of the two options we use further.

Single optimizer In this case, we use a single common optimizer for all neural networks. In practice, this optimizer updates the parameters (weights) of all networks simultaneously. When the loss function is optimized, the gradients of all parameters θ (the parameters of all networks) are sent to one optimizer. The optimizer views all weights θ_j as a single vector of size $N \times d_\theta$, where N is the number of networks and d_θ is the number of parameters per network. This approach ensures that all networks are updated synchronously and the learning is coordinated since the loss function accounts for the combined influence of all networks through the sum $\sum_{j=1}^{J} \omega_j v_j(x, \theta_j)$.

Multiple optimizers In this case, we use separate optimizers for each neural network. The loss function is still defined as in the eq.(6) but now each optimizer is responsible only for minimizing the loss associated with the parameters of its respective network. For example, for network functions

$v_1(x, \theta_1), v_2(x, \theta_2), \ldots$ each optimizer receives gradients only for its network's parameters and updates only those. However, it is important to note that the gradients of the loss function are still computed synchronously for all networks, meaning the loss is minimized across all networks at the same time. Despite this, independent optimization may still lead to desynchronization between networks, as each optimizer updates its parameters separately.

Comparison of the approaches and conclusion Comparing the two methods, as shown in Table 1, we can conclude that although multiple optimizers minimize the loss function simultaneously (since the gradient is computed once), this approach takes significantly more time for training and can lead to desynchronization between networks, especially as the number of networks increases, as seen in the case of FBPINNs. This reduces efficiency and convergence. Therefore, for FBPINNs, it is recommended to use a single optimizer for all neural networks. we examine the results of this approach in Chapter 3.

Parameter	**Single Optimizer**	**Multiple Optimizers**
Parameter update	All parameters are updated synchronously as a single vector	Each optimizer updates only its network's parameters, but gradients are computed simultaneously
Loss minimization	The entire loss is minimized synchronously	The loss is minimized synchronously, but each network updates its parameters independently
Network coordination	Full coordination between networks	Networks may work independently, leading to desynchronization
Flexibility	Less flexibility, as only one optimizer is used	More flexibility due to independent optimizers
Risks	Harder to tune hyperparameters for all networks at once	May lead to uncoordinated optimization and slower training

Table 1: Comparison between single optimizer and multiple optimizers

2.4 MFBPINNs

The article [3] introduces the concept of Multilevel Finite Basis Physics-Informed Neural Networks (MFBPINNs), which extends FBPINNs by adding multiple levels of domain decompositions (DDs) to their solution ansatz. This multilevel approach is inspired by classical multilevel domain decomposition methods, where additive levels are introduced to improve scalability, particularly when using large numbers of subdomains.

The main purpose of using a multilevel method is to solve high-frequency cases better. The idea is to repeat FBPINNs at different levels to work with various frequencies. Additive levels help predict the behavior of the function, making it easier to train the final level.

2.4.1 Description of MFBPINNs

We introduce a domain decomposition with L levels, where each level l defines an overlapping decomposition of the global domain Ω into $J^{(l)}$ subdomains. This can be expressed as

$$D^{(l)} = \{\Omega_j^{(l)}\}_{j=1}^{J^{(l)}}, \quad \text{for} \quad j = 1, \ldots, J^{(l)}, \quad l = 1, \ldots, L.$$

For simplicity, we assume that at the first level ($l = 1$), there is only one subdomain, meaning $J^{(1)} = 1$ and $\Omega_1^{(1)} = \Omega$. Additionally, the number of subdomains increases with each level, such that $J^{(1)} < J^{(2)} < \cdots < J^{(L)}$.

Similar to the FBPINNs method, for each subdomain $\Omega_j^{(l)}$ we define a corresponding space of neural network functions:

$$V_j^{(l)} = \{v_j^{(l)}(x, \theta_j^{(l)}) \mid x \in \Omega_j^{(l)}, \theta_j^{(l)} \in \Theta_j^{(l)}\}, \quad j = 1, \dots, J^{(l)}, \quad l = 1, \dots, L,$$

where $\theta_j^{(l)}$ is the set of trainable parameters for the network $v_j^{(l)}(x, \theta_j^{(l)})$, which is associated with subdomain $\Omega_j^{(l)}$.

Also, for each level we define a set of window functions with support in the corresponding subdomains. These functions are individual for each level and, as before, adhere to the partition of unity condition

$$\sum_{j=1}^{J^{(l)}} \omega_j^{(l)} = 1, \quad \text{on} \quad \Omega, \quad \forall l, , \quad l = 1, \dots, L,$$

The overall space of network functions V is then defined as a sum across all levels

$$V = \frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} V_j^{(l)}.$$

Using this global space decomposition, we approximate the solution to the problem as

$$u(x, \theta) = \frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} v_j^{(l)}(x, \theta_j^{(l)}). \tag{7}$$

This type of sum leads us to the conclusion that if we take $L = 1$ level, we get the original PINN or FBPINN in case of $J = 1$ subdomain.

As we have already discussed in section 2.3.2 we have chosen the method of single optimizer and will train the MFBPINN in the same way. To deal with BCs we also choose hard-constrained loss and using (4) and taking (7) as u we obtain the loss function for hard-constrained BCs for a MFBPINN

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\eta \left[\frac{1}{L} \sum_{l=1}^{L} \sum_{j=1}^{J^{(l)}} \omega_j^{(l)} v_j^{(l)}(x_i, \theta_j^{(l)}) \right] - f(x_i) \right)^2 \tag{8}$$

2.4.2 Two different approaches

As we already mentioned, we need additive levels to facilitate training at the last level, but it is worth figuring out how exactly we act with a set of levels and how this can help us achieve better convergence. To address this, we propose two different concepts and examine the differences between them.

First approach (*Average*) The first case is essentially similar to the initial version of FBPINNs in [3], only with different sum management in the approximated solution. We create several levels with separate neural networks and window functions, then sum up and average them.

For example, if we take a case with $L = 2$, at the first level we have 1 network and at the 2nd level 4 networks, particulary level =[1,4] . Then in total, we train 5 networks and our loss function will look like

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\eta \left[\frac{1}{2} \left(\sum_{j=1}^{1} \omega_j v_j(x_i, \theta_j) + \sum_{j=1}^{4} \omega_j v_j(x_i, \theta_j) \right) \right] - f(x_i) \right)^2. \tag{9}$$

If we used FBPINNs for subdomains, the loss would look like

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\eta \left[\sum_{j=1}^{5} \omega_j v_j(x_i, \theta_j) \right] - f(x_i) \right)^2. \tag{10}$$

As we can see they are very similar and it could lead to better communication between neural networks and more accurate results. However, since we train our neural networks at different levels simultaneously, the multi-level effect is lost, making the result very similar to what we found in FBPINNs.

Second approach (*Additive*) The second method suggests that we use multilevel training not to start with random parameters, but rather to help neural networks anticipate the behavior of the solution, based on the training of previous (additive) levels. That is, we perform a small pre-training of all levels except the last one. Thanks to this and the initial parameters from which the neural network begins training, we can get a more accurate solution.

In essence, we use the same formula (7), but divide it into stages. For example, for the case where $L = 3$ in 2D case with [1,4,16] we have

$L = 1, \quad u_1(x, \theta) = u_1(x, \theta),$

$L = 2, \quad u_2(x, \theta) = u_1(x, \theta) + \sum_{j=2}^{5} \omega_j u_j(x, \theta_j),$

$L = 3, \quad u_3(x, \theta) = u_2(x, \theta) + \sum_{j=6}^{21} \omega_j u_j(x, \theta_j).$

The advantage of this method is that, in theory, it will be easier for neural networks to train, as they will have a predefined starting point.

However, a major drawback is that each level requires the pre-training of the previous level before it can start. As a result, instead of minimizing the loss function once, we must minimize it multiple times (i.e., for each level), which we anticipate will require significantly more time.

2.4.3 Comparison of the approaches and conclusion

To compare the two methods, we refer to Table 2, but we also aim to summarize which method we chose and the reasons behind our decision.

Even though the *Average* method initially appeared faster and appeared to be a more effective adaptation of FBPINN, in practice, the results showed that the complex configuration of the sum slowed the process and did not significantly improve loss minimization. On the other hand, the *Additive* method demonstrated superior performance. Thanks to the pre-training of levels, the learning process did not stagnate when moving to the final level but instead further decreased compared to what is typically observed in PINNs. Furthermore, when working with higher frequencies, the *Additive* method performed better. Therefore, for the final results, we decided to use the concept of the *Additive* level and we further demonstrate the results using this method.

Approach	***Average***	***Additive***
Training process	Train all levels simultaneously; the sum of solutions from all levels is averaged.	Each level is trained sequentially, with additive levels pre-trained and finer levels trained afterward.
Loss function	Weighted average of the sum of solutions from multiple levels, but complex sum configuration may slow down minimization.	Sequentially adding solutions from coars levels to the final level, integrating them into the final approximation, improving the loss minimization.
Advantages	Faster initial setup and allowing better communication between neural networks across levels.	More effective training with pre-trained levels, avoids stagnation and works better for higher frequencies.
Disadvantages	The complex sum configuration can slow down training and does not noticeably improve loss minimization.	Training is slower initially since each level requires pre-training, but this pays off in later stages.
Time efficiency	Initially faster, but the complex sum slows down the process.	Slower pre-training, but more efficient overall, especially at higher frequencies.

Table 2: Comparison of training approaches: *Average* vs. *Additive*

3 Numerical Experiments and Results

In this chapter, we describe the Helmholtz problem and determine which method performs best in practice.

We investigate the problem from multiple angles to assess and decide which method is the most suitable for application. We first define the problem, then examine the parameters and how they may vary depending on the specific problem. We also describe the testing and evaluation methods that we plan to use. This will be followed by the results, which are divided into 1D and 2D cases for ease of comparison.

3.1 Formulations of Problems

3.1.1 *Helmholtz problem in one dimension*

Consider the Helmholtz equation in the domain $\Omega = [0, 1]$

$$-\Delta u - k^2 u = f \quad \text{in } \Omega, \tag{11}$$

where k is a scalar number. We set $k = 16$ for most tests, but k can be increased for higher frequency cases.

Dirichlet boundary conditions are prescribed

$$u = 0 \quad \text{on } \partial\Omega. \tag{12}$$

The source term $f(x)$ is given by

$$f(x, y) = k^2 \cdot \sin(k\pi x) + \left(\frac{k}{2}\right)^2 \cdot \sin\left(\frac{k}{2}\pi x\right) \tag{13}$$

For a value of k, the reference solution is calculated using the finite difference method.

3.1.2 *Helmholtz problem in two dimensions*

Consider the Helmholtz equation in the domain $\Omega = [0, 1]^2$

$$-\Delta u - k^2 u = f \quad \text{in } \Omega, \tag{14}$$

where k is a scalar number, fixed at 4 to study the high-frequency case.

Dirichlet boundary conditions are prescribed

$$u = 0 \quad \text{on } \partial\Omega. \tag{15}$$

The source term $f(x)$ is given by

$$f(x) = k^2 \cdot \sin(k\pi x) \cdot \sin(k\pi y) + \left(\frac{k}{2}\right)^2 \cdot \sin\left(\frac{k}{2}\pi x\right) \cdot \sin\left(\frac{k}{2}\pi y\right), \tag{16}$$

The reference solution for this problem is

$$u(x, y) = \sin(k\pi x) \cdot \sin(k\pi y) + \sin\left(\frac{k}{2}\pi x\right) \cdot \sin\left(\frac{k}{2}\pi y\right) \tag{17}$$

3.2 Testing methods overview

To better understand the results, we use three different types of testing.

The first type is called ablation testing, which helps us determine which parameters of neural networks we should use. For this type of testing, we fix the complexity of the task and vary parameters such as overlap and the number of hidden units, which are the neurons in the hidden layers of the network that process data between the input and output layers.

After selecting the optimal parameters, we proceed to the weak and strong scaling tests. These types of testing are commonly used in Domain Decomposition methods, so it is natural to assess the success of our methods this way. The main objectives of these tests are as follows.

- *Weak scaling:* The complexity of the problem is not fixed. We increase the model capacity and the problem complexity at the same rate. We expect that convergence and error will remain consistent as we scale the parameters.

- *Strong scaling:* The complexity of the problem is fixed and we increase the model capacity. We aim to observe improvements in convergence as the number of levels and neural networks increase, though we understand that training time will also grow.

In our case, model capacity refers to the number of levels, number of subdomains and size of the neural network in each subdomain. Increasing the complexity of the problem means increasing the wave number, which in turn raises the frequency of the problems.

All tests were performed on a single GPU system.

To evaluate which method performs better, we compare not only the loss function results but also the L^2 error between the predicted solution and the reference solution. we also examine solution maps for both methods.

we also consider the computational time required for each method. During our research, it became evident that each method has varying training and optimization times. Beyond a certain point, it may become impractical to use a method despite improvements in model convergence. Therefore, we also take into account the time required for each method.

3.3 Parameters

3.3.1 Implementation details

Although we conducted various tests where we altered the properties and complexity of the model, there were constant parameters and principles applied in all cases. These parameters were set during the testing and were chosen as the most optimal for each method.

- *Loss function and optimization.* All methods use a hard-constrained loss function and all problems were trained using the L^2 or Mean Square Error (MSE) loss. Additionally, we employed the ADAM optimizer with a learning rate of 0.8×10^{-2}. We fix the number of collocation points at 5000 across all methods for consistency in the comparisons.

- *Network architecture.* We utilized fully connected feedforward networks (FCNs) and for all methods, we normalized the input $\mathbf{x}$ for each subdomain network to the range $[-1, 1]$ along each dimension within their respective subdomains. In the case of PINNs, we normalize it to the range $[-1, 1]$ over the entire domain. Regarding network architecture, we used 4 hidden layers and the number of hidden units was varied during testing. For both 1D and 2D problems, the *sine* activation function is employed. It is also important to introduce a notation for the architecture of the levels. We know that a MFBPINN is equivalent to a FBPINN when only one level is used. To better understand the structure of our tests, we represent the number of neural networks at each level in square brackets. For example,[4] it refers to a FBPINN with one level and 4 networks, while [1, 2, 4] represents a FBPINN with three levels, with a different numbers of neural networks at each level, indicating a MFBPINNs architecture.

- *Domain Decomposition and level structure.* Each method utilizes domain decomposition. All MFBPINNs use an exponential level increase, namely $J^{(l)} = 2^{d(l-1)}$. The structure at each level is illustrated in Figure 3. While the division in multilevel cases can theoretically continue to any number of levels, in this paper we limit testing to 4 levels due to the complexity and computation time, which we find to be sufficient. For FBPINNs, we divide the domain into a comparable number of subdomains to ensure a relevant comparison with the multilevel case. All subdomains have a uniform rectangular shape at each level of decomposition, resulting in $2^{(l-1)}$ subdomains in each dimension.

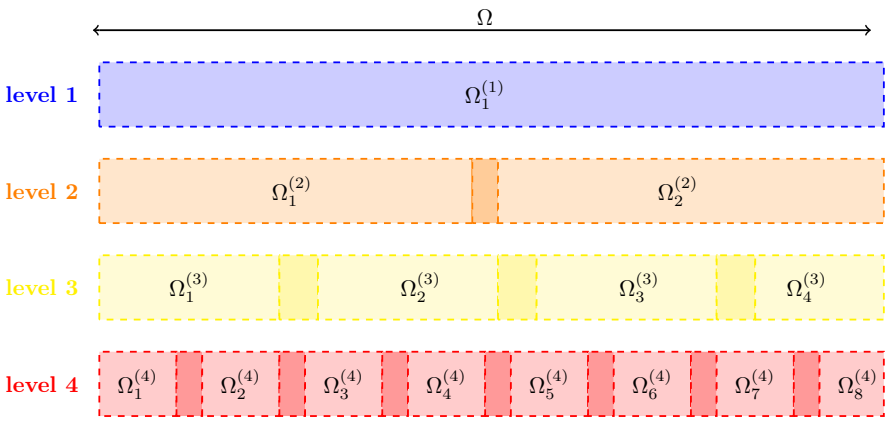


Figure 3: Multi-level decomposition of domain Ω for a 1D problem.

3.3.2 Window function

For the window function, we chose a *cosine window function*. Here is the definition for the 1D case.

We define the subdomains

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

where δ is defined as the overlap ratio and is fixed at a value of $\delta = 1.5$ for each case. Here l corresponds to the number of subdivisions. The subdomain window functions are given by

$$\omega_j^{(l)} = \frac{\hat{\omega}_j^{(l)}}{\sum_{j=1}^{l} \hat{\omega}_j^{(l)}},$$

where $\hat{\omega}_j^{(l)}(x)$ is defined as

$$\hat{\omega}_j^{(l)}(x) = \begin{cases} 1 & \text{if } l = 1, \\ \left[1 + \cos\left(\pi(x - \mu_j^{(l)})/\sigma_j^{(l)}\right)\right]^2 & \text{if } l > 1, \end{cases}$$

where $\mu_i^{(l)} = \frac{j-1}{l-1}$ and $\sigma_j^{(l)} = \frac{\delta/2}{l-1}$ represent the center and half-width of each subdomain, respectively.

For the 2D case, we modify the formulas slightly. To be more precise, now Ω represents the 2D domain and each $\Omega_{ij}^{(l)}$ is a 2-dimensional subdomain. Thus, we have

$$\Omega_{ij}^{(l)} = \begin{cases} [0.5 - \delta/2, 0.5 + \delta/2] \times [0.5 - \delta/2, 0.5 + \delta/2] & \text{if } l = 1, \\ \left[\frac{(i-1)-\delta/2}{l-1}, \frac{(i-1)+\delta/2}{l-1}\right] \times \left[\frac{(j-1)-\delta/2}{l-1}, \frac{(j-1)+\delta/2}{l-1}\right] & \text{if } l > 1, \end{cases}$$

Additionally, our window function remains unchanged, but we have the formula for $\hat{\omega}_{ij}^{(l)}(x)$

$$\hat{\omega}_{ij}^{(l)}(x) = \begin{cases} 1 & \text{if } l = 1, \\ \prod_k^2 \left[1 + \cos\left(\pi(x_k - \mu_{ij}^{(l)})/\sigma_{ij}^{(l)}\right)\right]^2 & \text{if } l > 1. \end{cases}$$

where $\mu_{ij}^{(l)} = \frac{(i-1)(j-1)}{(l-1)^2}$ and $\sigma_{ij}^{(l)} = \frac{\delta/2}{l-1}$ represent the center and half-width of each subdomain, respectively.

3.4 Results

3.4.1 *Helmholtz problem in one dimension*

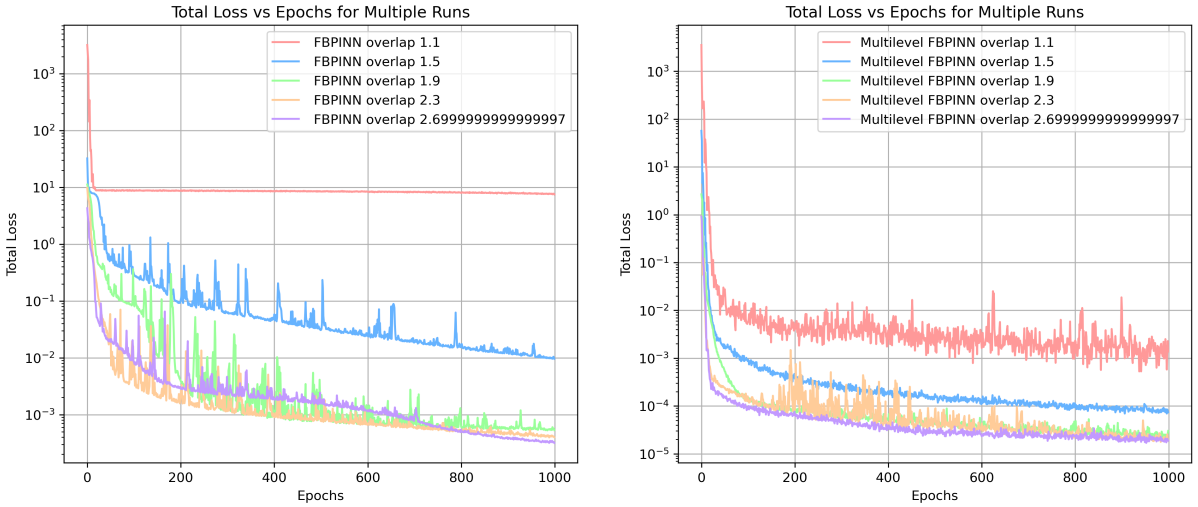
Ablation test

During the ablation test, we check two parameters and evaluate them using quantities such as the minimum loss function, absolute L^2 error between the reference solution and the solution produced by the method and of course the computational time.

1. *Overlap*

   As we have already expected earlier, the overlap is one of the important factors that can affect the speed and accuracy of training. Moving forward, we vary the overlap, starting from 1.1 and increasing to 2.7 in increments of 0.4. Recall that an overlap of less than 1 means that the subdomains do not overlap, which does not correspond to the main idea of the methods and can only worsen the process because as a result we have areas in which we do not consider the solution. We fix the wave number to $k = 2$ to generally understand from which overlap it is worth starting. We test the methods using the following architectures: [4] for FBPINNs and [1, 2, 4] for the multilevel case.

   From these studies, we expect that increasing the overlap will generally improve performance, but only within reasonable limits. It will be interesting to identify the point where a further increase no longer provides benefits or even leads to worse results due to excessive overlap.



(a) FBPINNs [4]

(b) MFBPINNs [1,2,4]

Figure 4: comparison of FBPINNs and MFBPINNs by varying the overlap

As we can see from the plots in Figur 4 for the two methods, overlap values of 1.9, 2.3 and 2.7 are good options from the perspective of minimizing the loss function. Let us examine the L^2 error and time performance. Table 3 compares the two methods when $k = 2$.

The L^2 errors are almost the same, but the multilevel case performs better and as we can see that the error is minimized for an overlap of 2.7.

Overlap	FBPINNs	MFBPINNs
1.1	11.20	0.0303
1.5	0.0381	0.0222
1.9	0.0364	0.0279
2.3	0.0222	0.0183
2.7	0.0232	0.0135

Table 3: L^2 error with $k = 2$ for FBPINNs and MFBPINNs in 1D.

When we look at the elapsed time form Table 4, we can see that the difference in overlap does not significantly affect the process. However, what is interesting is that with an increase in overlap, the most attractive option is 2.7.

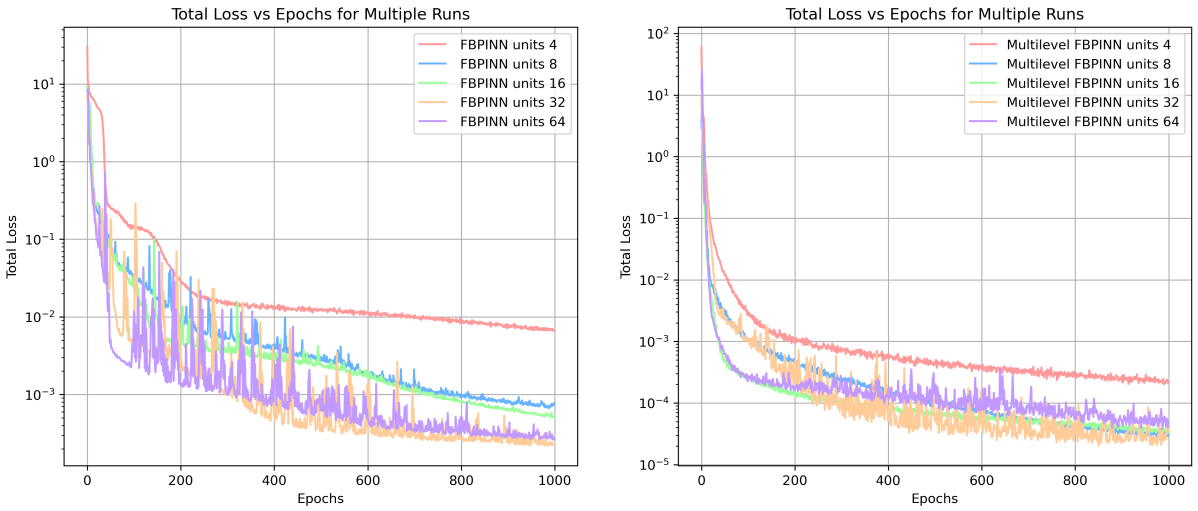
It is worth noting that although the total training time of the two methods is almost the same, FBPINNs performed 1,000 epochs, while the multilevel method performed 2,500 epochs (including the training of additive levels). Therefore, the average training time per epoch differs significantly, which is important to consider. It can be concluded that the multilevel method achieves the same accuracy much faster.

Overlap	**FBPINNs**	**MFBPINNs**
1.1	609.98	559.64
1.5	583.68	753.22
1.9	585.27	707.61
2.3	581.13	688.08
2.7	601.8	667.66

Table 4: elapsed time in second with $k = 2$ for FBPINNs and MFBPINNs in 1D.

2. *Hidden units*

The number of hidden units can significantly affect both convergence and elapsed time. Too few units may result in the neural network not being sufficiently trained, while too many units can lead unnecessary computations. To find the optimal number of hidden units, we conduct tests using 4, 8, 16, 32 and 64 units. The number of hidden layers is set as 4.



(a) FBPINNs [4]

(b) MFBPINNs [1,2,4]

Figure 5: comparison of FBPINNs and MFBPINNs by varying the number of hidden units.

As we can see from the plots of loss function convergence in Figure 5, 32 hidden units effectively minimize the loss function without causing too frequent or chaotic updates to the approximation.

The L^2 errors in Table 5 are almost the same. For hidden units value, we do not expect a direct influence on the convergence, but the multilevel case performs better and as we can see the minimal error is near the 32 hidden units value.

Hidden units	**FBPINNs**	**MFBPINNs**
4	0.0164	0.0186
8	0.0276	0.0123
16	0.0215	0.0252
32	0.0396	0.0161
64	0.0492	0.0251

Table 5: L^2 error with $k = 2$ for FBPINNs and MFBPINNs in 1D.

Due to the increase in computational time, the number of units directly impacts the total time. Therefore, it is important to carefully select an approximately average value where the convergence is sufficiently good, but the training time does not become excessive. Thus, 32 is a great fit here.

Hidden units	**FBPINNs**	**MFBPINNs**
4	382.54	402.29
8	423.86	649.57
16	585.34	735.42
32	1122.24	760.89
64	2091.26	1050.23

Table 6: elapsed time in seconds with $k = 2$ for FBPINNs and MFBPINNs in 1D.

Weak scaling

Under weak scaling conditions, we increase both the complexity of the problem and the size of the model.

In the 1D case, increasing complexity means scaling the wave number from 2 to 16, specifically 2, 4, 8 and 16.

Increasing the model size in the case of FBPINNs involves increasing the number of neural networks and use [1], [2], [4] and [8] and raising the number of hidden units to 8, 16, 32 and 64.

For MFBPINNs, we increase the number of levels from 1 to 4, where the number of neural networks at each level is set as $2^{\text{level}-1}$. We also increase the number of hidden units to 8, 16, 32 and 64.

It is important to note that each level is trained for a different number of epochs, defined as $500 \times$level. However, at this stage, this is sufficient, as even with such a small number of epochs, we can observe the convergence trend.

Now, let us move on to the results, starting with the approximate solutions.

Figure 8 shows the reference solution in blue and our approximated solution by one of the methods in orange.

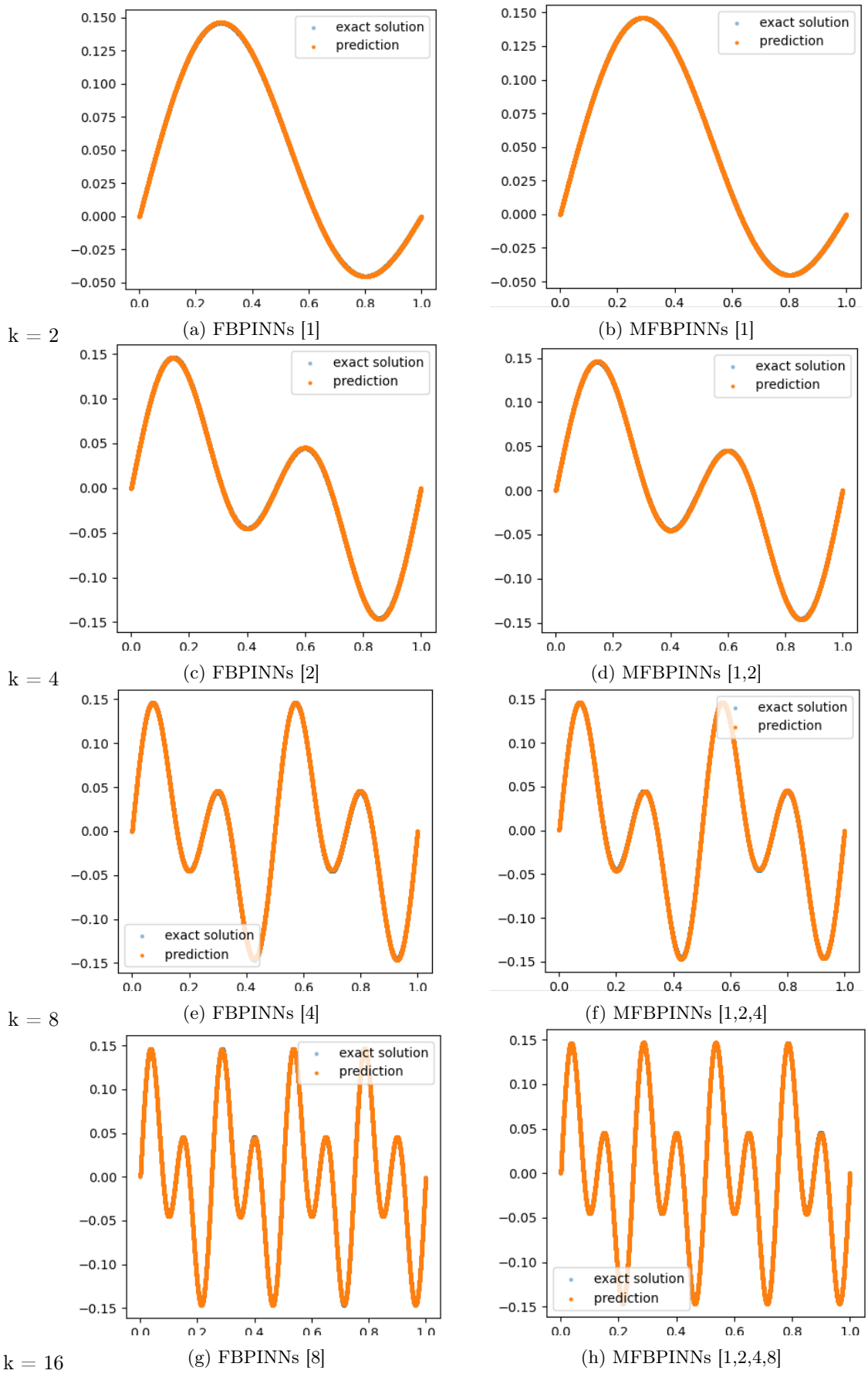
As we can see, both methods perform well for either parameter. However, when the complexity of the problem increases with the capacity of the model, then the convergence should remain approximately the same. To understand this in more detail, let us examine the other criteria.

At the plots of the loss functions and L^2 errors we also can see that both methods work equally well under the concidered conditions, as expected from our scaling.

Since we can trace the trend of multilevel FBPINNs method and compare it with FBPINNs we can see that their loss functions converge to approximately the same value for any of the wave numbers. The same can be said about the L^2 error presented in Table 7, as we get approximately the same results for each of the parameters. This is interesting for us because we could assume that the Multilevel method will work better.

In order to understand the real difference between the methods it is also necessary to pay attention to the computational time. The results are presented in Table 8. Measurements are made in seconds per epoch to compare the speed of the methods more relevantly.

As we can see the MFBPINNs method is much faster, which only confirms our assumption that pre-training helps to speed up the algorithm. As a conclusion, scaling model capacity and increasing complexity does keep convergence about the same but the difference in training time is significant. In order to understand the models in more detail in future studies, we can have a look at more difficult problems and this is one of the reason why we move to 2D case.



k = 2

(a) FBPINNs [1] (b) MFBPINNs [1]

k = 4

(c) FBPINNs [2] (d) MFBPINNs [1,2]

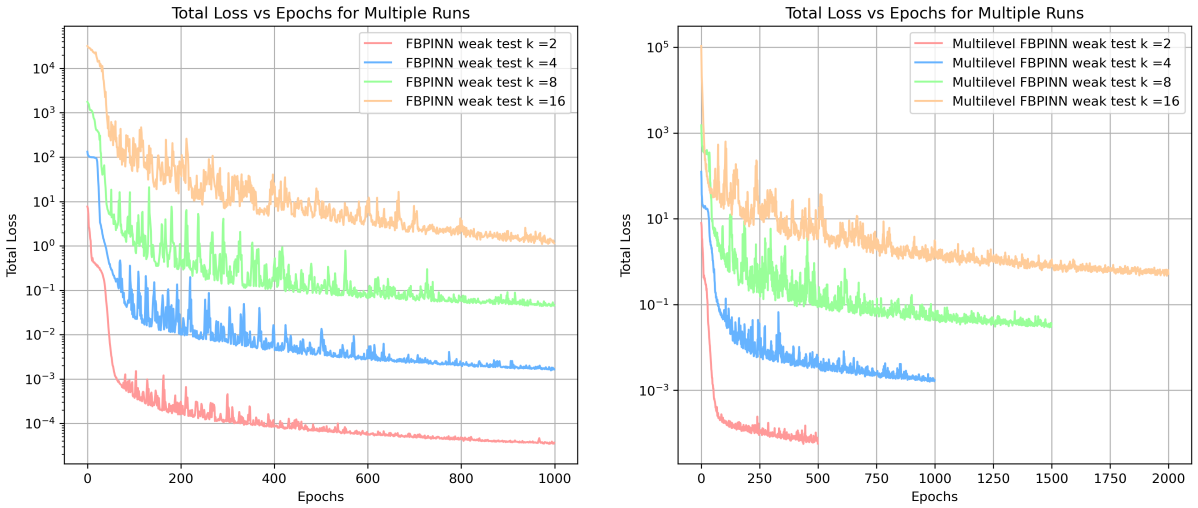
k = 8

(e) FBPINNs [4] (f) MFBPINNs [1,2,4]

k = 16

(g) FBPINNs [8] (h) MFBPINNs [1,2,4,8]

Figure 6: comparison of FBPINNs and MFBPINNs for different value of k.



(a) FBPINNs (b) MFBPINNs

Figure 7: comparison of FBPINNs and MFBPINNs loss function in the weak scaling regime.

k	FBPINNs	MFBPINNs
2	0.0194	0.0102
4	0.0275	0.0318
8	0.0695	0.0673
16	0.0647	0.0634

Table 7: L^2 errors for FBPINNs and MFBPINNs in the weak scaling regime.

k	FBPINNs	MFBPINNs
2	0.118	0.063
4	0.310	0.109
8	0.854	0.334
16	2.814	1.157

Table 8: elapsed time per epoch for FBPINNs and MFBPINNs in the weak scaling regime.

Strong scaling

As described in Section 3.3.1 we now fix the complexity of the problem and increase the capacity of the model.

After the result of the weak scaling, we fix the wave number $k = 16$. We scaled the capacity of the model in the same way as in weak scaling.

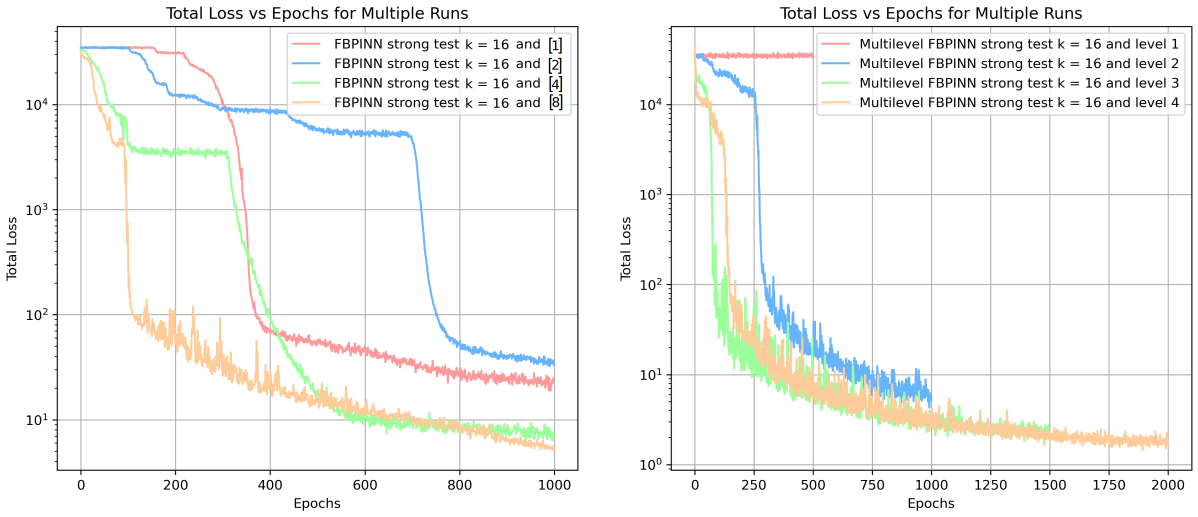
We start by plotting the results of the approximation of the two methods. As we can see both methods do quite well despite the complexity. An unpleasant exception is Multilevel FBPINN method with $L = 1$, in this case we have only one neural network and despite the fact that this is essentially a PINN method after 10 tests we got the same result and we can conclude that perhaps this is due to the architecture of neural networks here negatively influenced by the insufficient number of epochs at this level.

By comparing the loss functions in Figure 9 and L^2 errors in Table 10, we see that convergence increases with the number of neural networks, but also note that [4] and [8] for FBPINNs (analogically, [1,2,4] and [1,2,4,8] for MFBPINNs) show approximately the same result in both cases. It is also interesting that the multilevel FBPINNs and FBPINNs methods show the same convergence.

From the point of view of elapsed time in Table 9, as expected it coincides with the time of calculations of the weak scaling and this means for us that the time of each of the methods in the 1D case

depends only on the number of neural networks and does not rely on the complexity of the problem.

We can conclude that selecting $L = 3$ for multilevel FBPINNs and [4] for FBPINNs is preferable, as it achieves comparable convergence to $L = 4$ and [8], while requiring at least 50% less computational time in each case.



(a) FBPINNs (b) MFBPINNs

Figure 9: comparison of FBPINNs and MFBPINNs in the context of strong scaling test.

Levels FBPINNs	FBPINNs	Levels MFBPINNs	MFBPINNs
[1]	0.180	[1]	0.070
[2]	0.353	[1,2]	0.109
[4]	0.819	[1,2,4]	0.251
[8]	1.956	[1,2,4,8]	1.590

Table 9: elapsed time per epoch for FBPINNs and MFBPINNs in the context of strong scaling test.

Levels FBPINNs	FBPINNs	Levels MFBPINNs	MFBPINNs
[1]	0.2127	[1]	11.7303
[2]	0.2121	[1,2]	0.1985
[4]	0.0718	[1,2,4]	0.0974
[8]	0.0641	[1,2,4,8]	0.0601

Table 10: L^2 errors for FBPINNs and MFBPINNs in the context of strong scaling test.

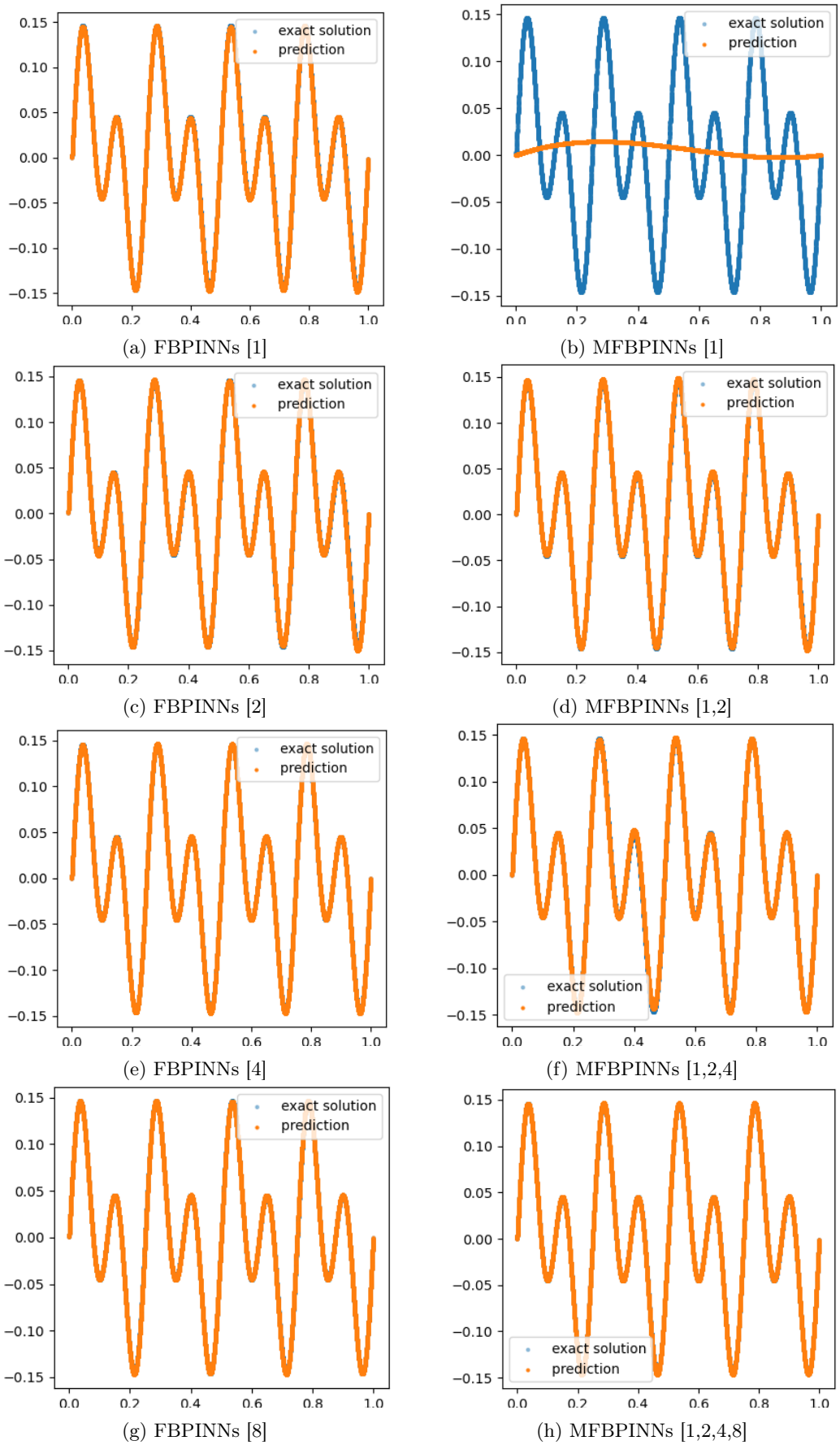
3.4.2 *Helmholtz problem in two dimensions*

Ablation test

As in 1D case during the ablation test we check overlap and hidden units values and compare them using the minimum loss function, L^2 error between the reference solution and the solution made by the method and of course the computational time.

1. *Overlap*

   As previously anticipated, overlap is crucial in determining training speed and accuracy. To start, we set the wave number at $k = 2$. Next, we adjust the overlap from 1.1 to 2.7, increasing it by 0.4 increments, similar to the 1D case. To assess the influence overlap, we fixed the number of hidden units at 16, the average of the test range, to ensure experimental consistency.



(a) FBPINNs [1]

(b) MFBPINNs [1]

(c) FBPINNs [2]

(d) MFBPINNs [1,2]

(e) FBPINNs [4]

(f) MFBPINNs [1,2,4]

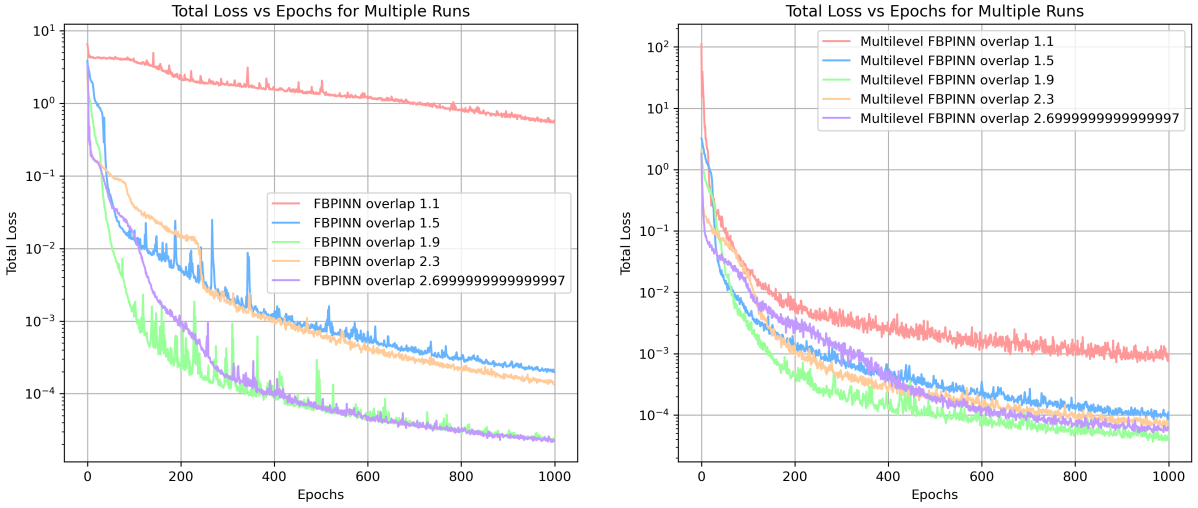
(g) FBPINNs [8]

(h) MFBPINNs [1,2,4,8]

Figure 8: comparison of FBPINNs and MFBPINNs for different levels in the context of strong scaling test.

For the multilevel case, we selected $L = 2$ so we use [1,4] structure and accordingly use the same number of neural networks as in the last level for FBPINNs [4]. We expect that increasing the overlap will enhance the results, but our goal is also to determine the point where a further increase no longer offer benefits or even begin to degrade performance due to excessive overlap.

By examining the convergence of the loss function in Figure 10 and the L^2 error in Table 11, we observe that for both methods, overlaps of 1.9 and 2.7 perform equally well. In the case of FBPINNs, these two overlaps are significantly better than the others, showing greater convergence. Interestingly, unlike FBPINNs, the multilevel method performs reasonably well even with the smallest overlap and in general, any overlap in the range of 1.9 to 2.7 works well.



(a) FBPINNs (b) MFBPINNs

Figure 10: comparison of FBPINNs and MFBPINNs by varying the overlap size.

Overlap	FBPINNs	MFBPINNs
1.1	1.2584	0.0304
1.5	0.0030	0.0058
1.9	0.0030	0.0041
2.3	0.0029	0.0070
2.7	0.0025	0.0048

Table 11: performance L^2 error for FBPINNs and MFBPINNs in 2D for overlap ablation test

When comparing the training time in Table 12, it is evident that the multilevel method is nearly twice as fast, except in the case of overlap 2.7. For this overlap, both methods require a significant time for calculations. It appears that this is the point where the results remain good, but further increase of the overlap leads to redundant recalculations, which hinders efficient and fast training.

In conclusion, we can say that an overlap of 1.9 is the most suitable, as the difference in convergence is minimal, but it significantly reduces the calculation time.

overlap	FBPINNs	MFBPINNs
1.1	2134.22	1113.92
1.5	1850.35	856.26
1.9	1708.95	866.50
2.3	1775.71	867.32
2.7	1810.57	2081.39

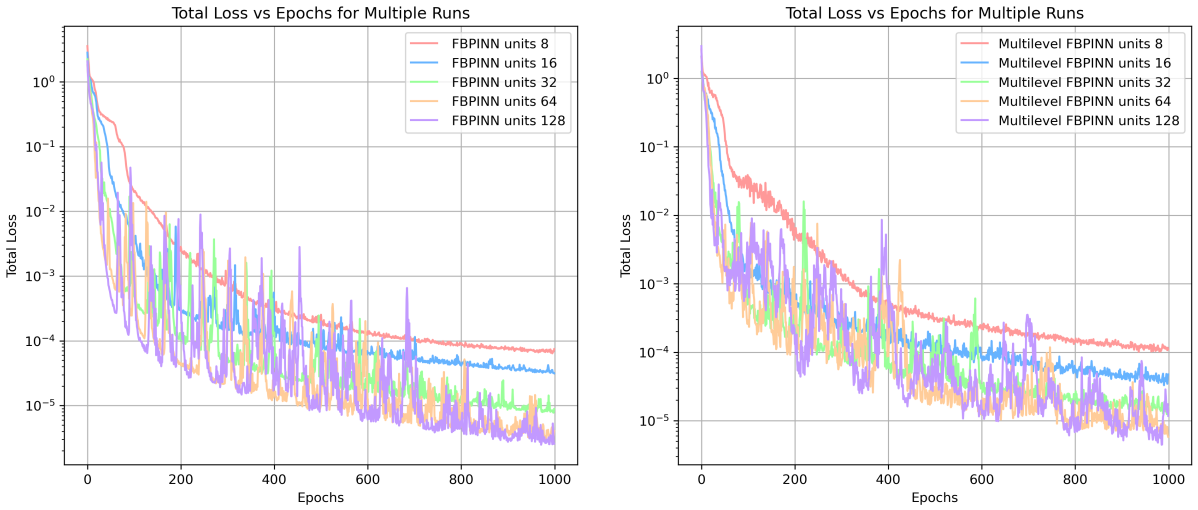
Table 12: elapsed time in seconds for FBPINNs and MFBPINNs in 2D by varying the overlap size.

2. *Hidden units*

Now that we are considering a 2D case, making the task more complex and demanding, let us analyze the performance of the number hidden units with the following values: 8, 16, 32, 64 and 128 units. As before, we set the wave number to $k = 2$ and fix the overlap at 1.9, which we know from previous tests is the optimal overlap value. Also we set the number of hidden layers as 4.

For both methods, we choose the same level structure as in overlap ablation test. We expect that this experiment will not significantly impact convergence, but it may help identify the hidden unit value that leads to more optimal and faster training.

By examining the L^2 error in Table 13 and the loss function in Figure 11, we can see that our expectations are confirmed. At first glance, it appears that values of 128 and 64 yield the best results, but they also exhibit the most instability and their error is not significantly better than the other values. On the other hand, 32 hidden units produce slightly worse results but demonstrate much more stable behavior.



(a) FBPINNs

(b) MFBPINNs

Figure 11: comparison of FBPINNs and MFBPINNs by varying the hidden units amount.

Hidden units	**FBPINNs**	**MFBPINNs**
8	0.0025	0.0038
16	0.0022	0.0054
32	0.0026	0.0056
64	0.0023	0.0051
128	0.0022	0.0045

Table 13: performance L^2 error with $k = 2$ for FBPINNs and MFBPINNs in 2D for hidden units ablation test.

When we analyze the timing data in Table 14, it becomes clear that we are paying a high price for a minimal improvement. In the case of FBPINNs, the training time increases at an exponential rate. For the Multilevel method, the time only becomes critical with 128 units, but it is still lower than that of FBPINNs.

Here, we need to find a balance and reach a compromise. If we compare the results of both methods and consider our conclusions about convergence, 32 units stand out as the most reasonable middle-ground option.

Hidden units	FBPINNs	MFBPINNs
8	1005.39	507.92
16	1246.01	579.24
32	2206.85	717.15
64	3977.31	850.41
128	7528.46	1387.17

Table 14: elapsed time with $k = 2$ for FBPINNs and MFBPINNs in 2D for hidden units ablation test.

Weak scaling

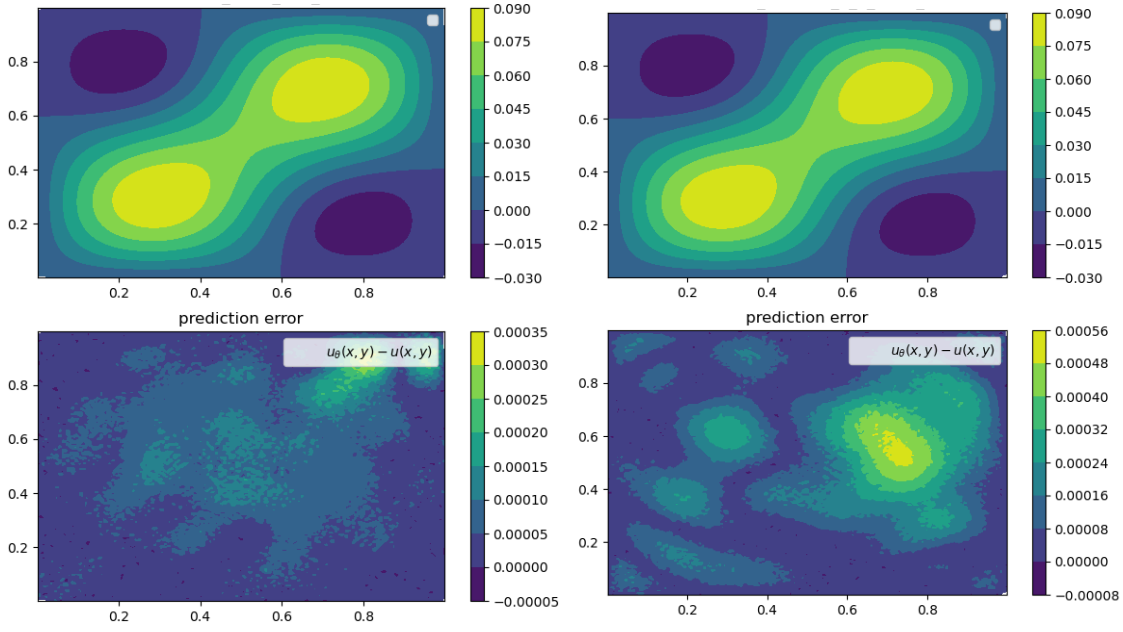
Now that we have selected the optimal overlap, we can begin analyzing the performance of the methods using a weak scaling. We increase the complexity of the problems by using wave numbers of 2, 4 and 8.

For the levels, we use levels 1, 2 and 3. In particular in the 2D case, the number of neural networks at each level is calculated as $2^{2\cdot(level-1)}$. Consequently, for FBPINN, we use [1],[4] and [16]. For MFBPINNs case: [1], [1,4] and [1,4,16]. Also after weak scaling we set overlap = 1.9. In this test, the same situation regarding epochs will apply as described in the weak scaling for the 1D case,so we similarly compare the time in seconds per epoch.

An important detail to mention is that we have reduced the range of wave numbers from 4 to 3. This is because for $k = 16$ and level 4 (64 neural networks), both methods produced poor results and the time to perform a test for each method exceeded one day on the equipment used for testing. Therefore, we conclude that $L = 4$ is not relevant for evaluation in the context of our study.

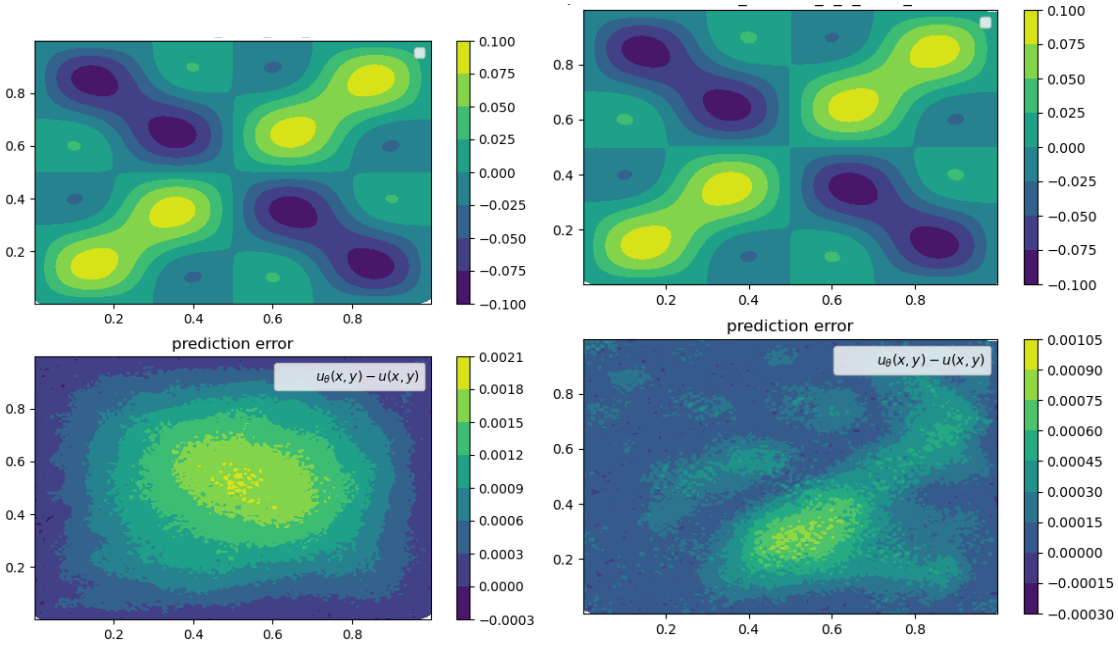
If we analyze the results from Figures 12, 13 and 14, along with the L^2 error in Table 15, we can observe that the multilevel method performs better overall, especially as the frequency increases. The error graphs, which compare the reference solution to the approximated solution, further confirm that the multilevel method generally exhibits greater convergence. Looking at the loss convergence in Figure 15, the results are nearly the same for both methods. If we continue to project the trend for the multilevel method, both methods appear equivalent, as we initially assumed.

However, the time difference shown in Table 16 suggests that the Multilevel method is preferred due to its faster performance. Thus, we conclude that as the complexity of the problem scales and the method's parameters increase, FBPINNs perform well up to a certain frequency. Beyond that, Multilevel demonstrates better results. This differs from the 1D case, where both methods performed similarly. It is promising to see the advantages of the improved multilevel method in handling high-frequency problems.



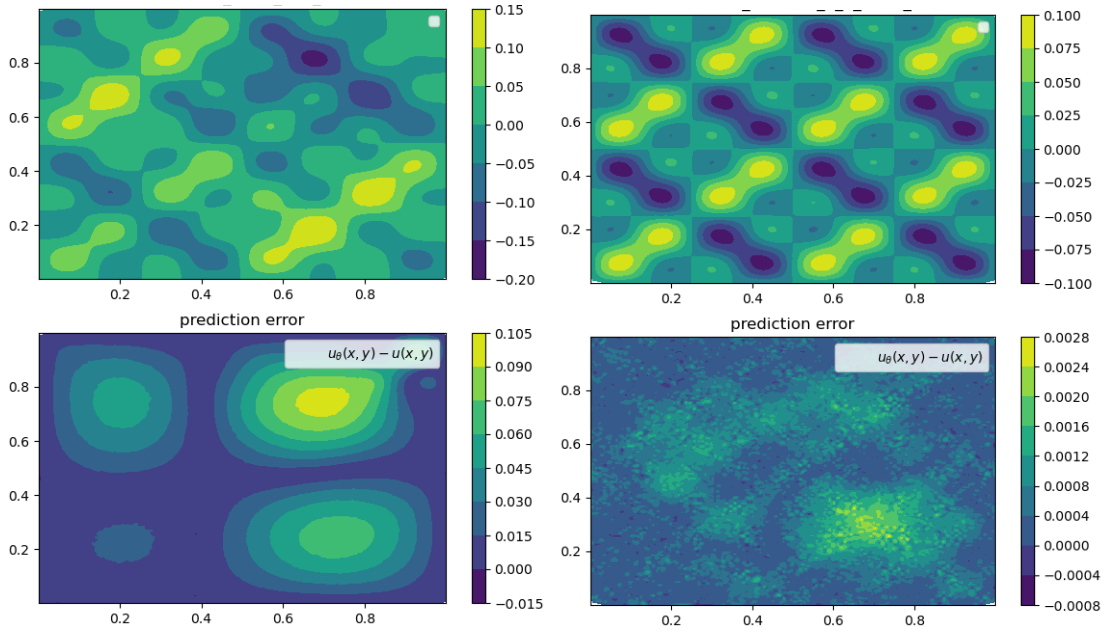
(a) FBPINNs [1] (b) MFBPINNs [1]

Figure 12: k = 2



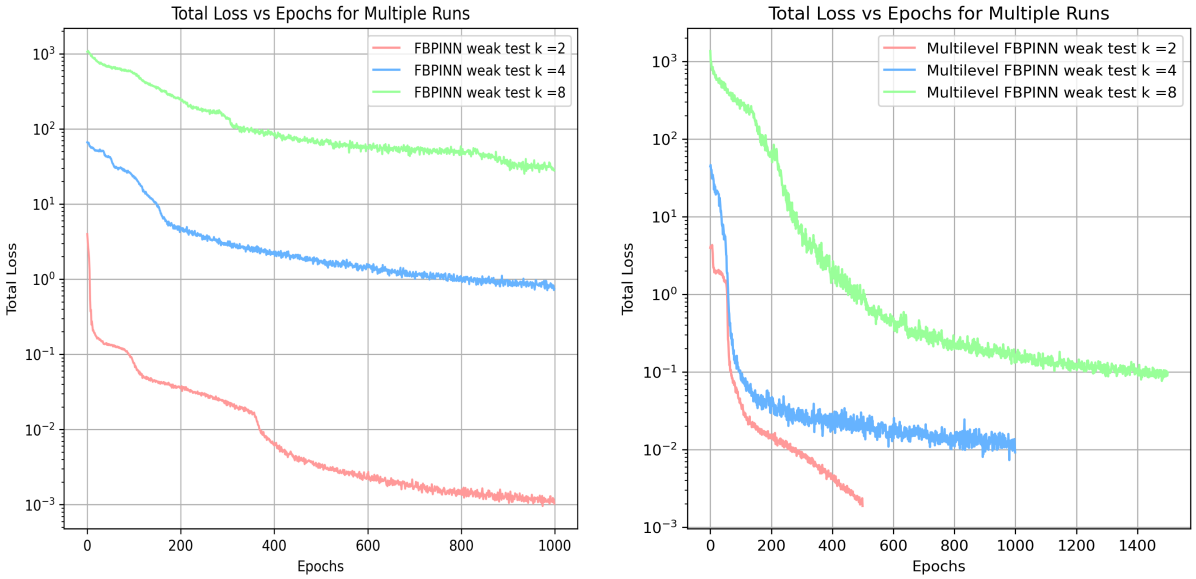
(a) FBPINNs [4] (b) MFBPINNs [1,4]

Figure 13: k = 4



(a) FBPINNs [16] (b) MFBPINNs [1,4,16]

Figure 14: k = 8



(a) FBPINNs (b) MFBPINNs

Figure 15: Comparison of FBPINNs and MFBPINNs in the weak scaling regime.

k	FBPINNs	MFBPINNs
2	0.0093	0.0227
4	0.0975	0.0345
8	4.7865	0.0850

Table 15: L^2 errors for FBPINNs and MFBPINNs in the weak scaling regime.

k	FBPINNs	MFBPINNs
2	0.986	0.226
4	1.803	0.587
8	8.178	5.99

Table 16: elapsed time per epoch for FBPINNs and MFBPINNs in the weak scaling regime.

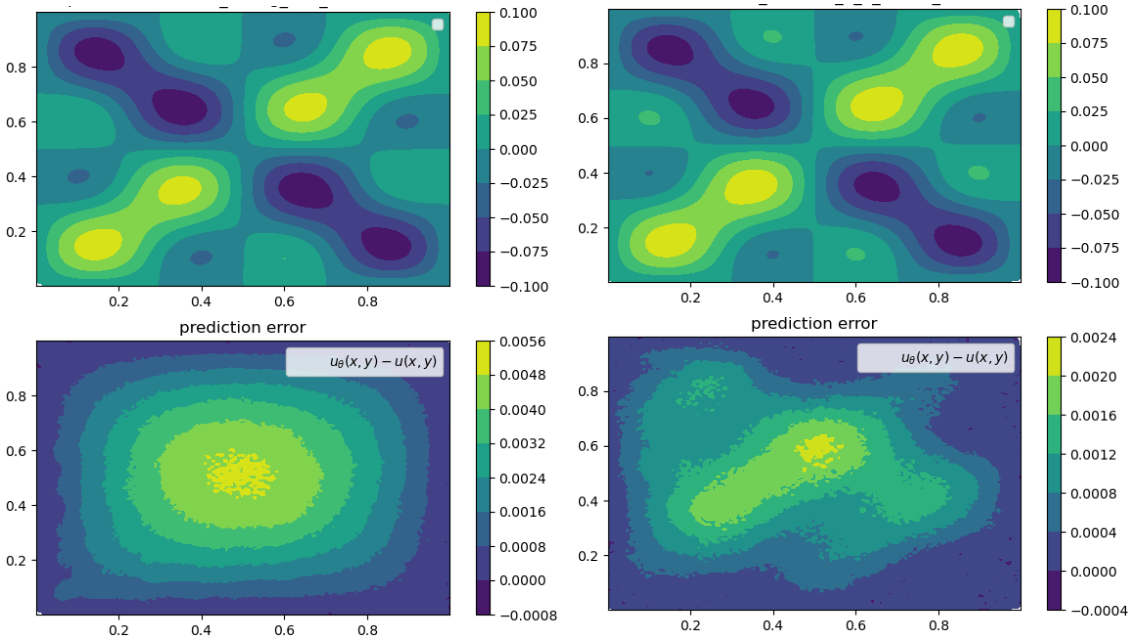
Strong scaling

For strong scaling, we fix the wave number at an average complexity of $k = 4$. Although we have previously seen good results with this wave number, we are still interested in exploring the impact of increasing the capacity of the methods.

We rank the model sizes in the same way as in the weak scaling, meaning the test will be conducted with [1], [4] and [16] neural networks for FBPINNs and [1], [1,4], [1,4,16] for MFBPINNs. Also, in the multilevel method, there is again a consideration regarding the difference in the number of epochs, so we evaluate the time in terms of seconds per epoch.

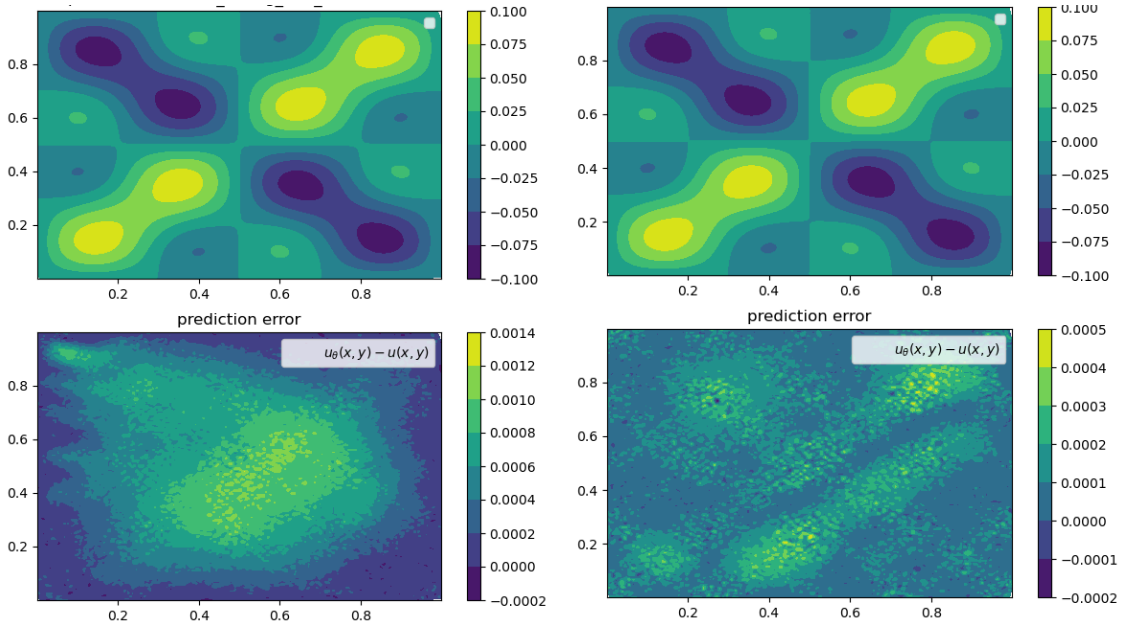
By observing the approximation results in Figures 16, 17 and 18, along with the L^2 error in Table 17, we can see that at each level, the multilevel method performs significantly better for any number of neural networks and convergence improves as the number of levels increases.

Referring to the behavior of the loss functions in Figure 19b, we observe that level 2 and [4] for FBPINNs show the best results in terms of convergence. Regarding the computation time from Table 18, it seems prudent to aim for a balanced approach, as the time difference between the use of neural networks [4] and [16] is substantial, while the improvement in convergence is not as critical for either method.



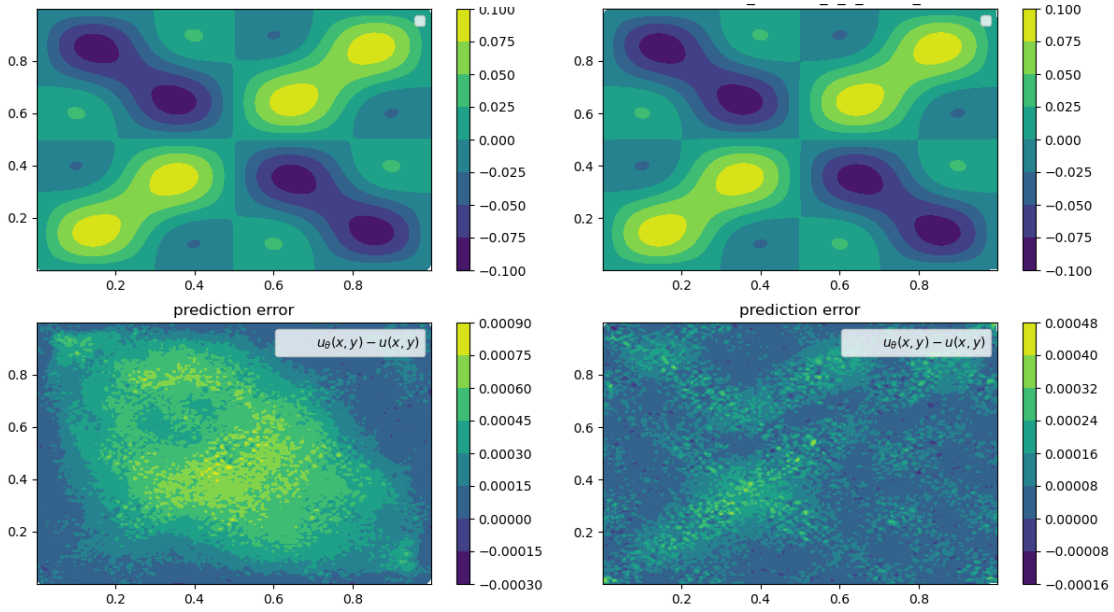
(a) FBPINNs (b) Multilevel

Figure 16: FBPINNs level [1] MFBPINNs level [1]



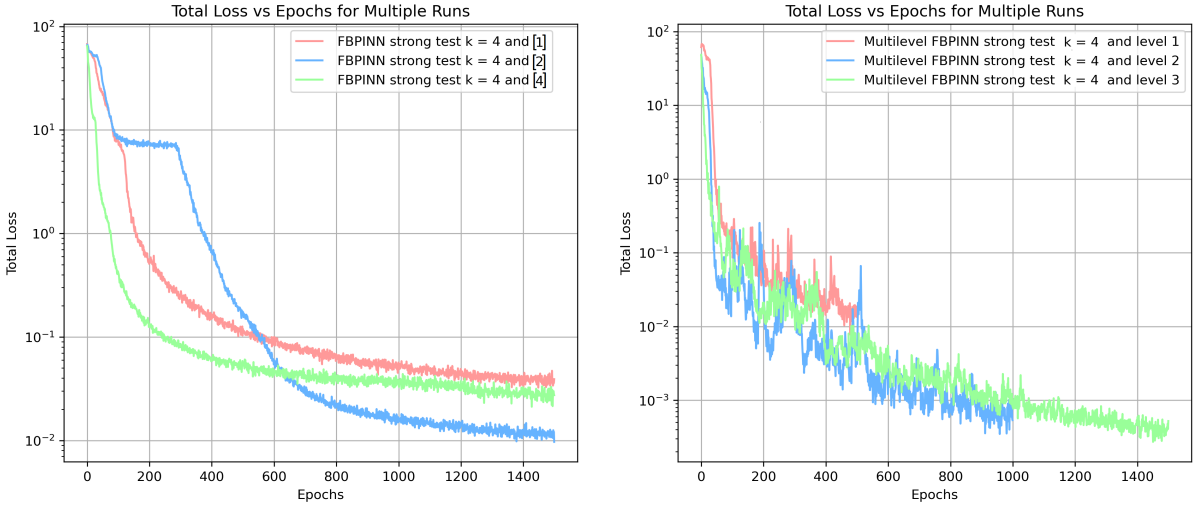
(a) FBPINNs (b) MFBPINNs

Figure 17: FBPINNs level [4] MFBPINNs level [1,4]



(a) FBPINNs (b) MFBPINNs

Figure 18: FBPINNs level [16] MFBPINNs level [1,4,16]



(a) FBPINNs (b) MFBPINNs

Figure 19: Comparison of FBPINNs and MFBPINNs Strong Test

levels FBPINNs	**FBPINNs**	**levels MFBPINNs**	**MFBPINNs**
[1]	0.3480	[1]	0.3220
[4]	0.0732	[1,4]	0.0158
[16]	0.0496	[1,4,16]	0.0125

Table 17: L^2 errors for FBPINNs and MFBPINNs in Strong Test

levels FBPINNs	**FBPINNs**	**levels MFBPINNs**	**MFBPINNs**
[1]	0.804	[1]	0.299
[4]	1.811	[1,4]	0.612
[16]	6.591	[1,4,16]	4.534

Table 18: Elapsed time per Epochs for FBPINNs and MFBPINNs in Strong Test

In general, based on the results of all tests, we conclude that both methods provide good convergence at lower frequencies. However, when moving to higher frequencies, the multilevel method clearly outperforms. It is also important to note that computation time differs significantly between the methods, which further highlights the advantage of the multilevel method.

4 Conclusion

In this thesis, we explored the application of Finite-Based Physics-Informed Neural Networks (FBPINNs) and MFBPINNs to solve the Helmholtz equation in both one- and two-dimensional cases.

Our results revealed important insights into the behavior of the two methods in different dimensions. In the one-dimensional case, the differences between FBPINNs and MFBPINNs were minimal. This outcome can be attributed to the lower complexity and frequency of the 1D problem, where both methods performed similarly well. The simplicity of the problem did not fully challenge the potential advantages of the multilevel structure, and FBPINNs were able to provide good performance without requiring the more complex multilevel setup.

However, in the two-dimensional case, the MFBPINNs began to show their strengths. As the complexity and frequency of the problem increased, the multilevel approach demonstrated better performance, particularly in terms of convergence and accuracy. By breaking down the domain into multiple levels and utilizing pre-training on additive subdomains, the MFBPINNs were able to handle high-frequency problems more effectively. This allowed for faster training and more accurate results compared to FBPINNs, highlighting the potential of multilevel architecture to solve complex wave propagation problems.

An equally important part of the work was the development of the code and the application of the ScimBa library. Based on existing methods, we were able to expand the library to include FBPINNs and MFBPINNs. We created an environment for the application of both single and multiple optimizers. It was also interesting to examine the multilevel method from two perspectives, and we were able to create and develop an additive levels version that showed significantly better results compared to the FBPINNs method.

In conclusion, while the difference between FBPINNs and MFBPINNs was not noticeable in the one-dimensional case, the two-dimensional experiments highlighted the advantages of the multilevel approach, particularly for higher-frequency problems. The development of code and improvements in the ScimBa environment were essential to achieving these results and we believe this provides a solid foundation for further exploration of neural network-based methods in solving partial differential equations. Future work will focus on solving more complex problems, testing and creating newer and more efficient methods based on MFBPINNs.

References

[1] Franck E., Boileau M. , Michel-Dansac V. *ScimBa*, URL `https://gitlab.inria.fr/scimba/scimba`

[2] Hrebenshchykova D. *PINNs and another code for article*, URL `https://gitlab.inria.fr/dhrebens/helmholtzpinns`

[3] Dolean V., Heinlein A., Mishra S. , Moseley B. *Multilevel domain decomposition-based architectures for physics- informed neural networks*, URL `https://arxiv.org/pdf/2306.05486`

[4] 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*, Journal of Computational Physics 378 (2019) 686–707. URL `https://doi.org/10.1016/j.jcp.2018.10.045`

RESEARCH CENTRE
Centre Inria d'Université Côte d'Azur

2004 route des Lucioles - BP 93
06902 Sophia Antipolis Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399