NeuroDiffeq: Implementing Artificial Neural Networks to Solve 2D Helmholtz Equations

Soumaya Nouna[1,*], Ilyas Tammouch[2], Assia Nouna[1]
[1]Hassan First University of Settat, ENSA Berrechid, Laboratory LAMSAD, Morocco.
[2]Laboratory of Telecommunications Systems and Decision Engineering, Faculty of Science, Ibn Tofail University, Kenitra, Morocco
E-mail: s.nouna@uhp.ac.ma
*Corresponding Author

Keywords: Deep learning, Helmholtz equations, neural network, artificial neural networks, partial differential equation

Received: July 24, 2025

This paper investigates the application of artificial neural networks (ANNs) for solving two-dimensional Helmholtz equations using the NeuroDiffEq framework. Based on PyTorch, NeuroDiffEq enables mesh-free approximations of PDE solutions by minimizing the residuals of the differential equation and its boundary conditions through automatic differentiation. The proposed method integrates a trial analytical solution (TAS) to enforce boundary constraints and uses a feedforward neural network trained via the Adam optimizer. We evaluate the model on several benchmark Helmholtz problems and report mean squared errors (MSE) as low as 1.08×10^{-6}. *Comparative experiments demonstrate that the ANN-based solver achieves better accuracy than classical finite difference methods in certain scenarios. The results highlight the effectiveness, convergence behavior, and flexibility of the NeuroDiffEq approach for PDE solving in two-dimensional domains.*

Povzetek: Članek preuči reševanje 2D Helmholtzevih enačb z nevronskimi mrežami v NeuroDiffEq, kjer z minimizacijo ostankov in poskusno analitično rešitvijo (TAS) uveljavi robne pogoje, ter s preprosto FFN+Adam metodo pokaže učinkovito alternativo klasičnim numeričnim metodam.

1 Introduction

The Helmholtz equation arises in numerous engineering and scientific applications as a fundamental partial differential equation (PDE) [2, 21]. Its solutions model wave propagation phenomena [1], such as electromagnetic waves, seismic waves, and acoustic fields. Due to its wide applicability, considerable attention has been devoted by computational mathematicians to developing accurate and efficient numerical methods to solve the Helmholtz equation.

Historically, the Helmholtz equation has been addressed using a variety of classical numerical techniques, including the finite difference method (FDM) [3, 4], the finite element method (FEM) [5], the plane wave method [6], and the boundary element method (BEM) [7, 8]. These methods are well-established and thoroughly analyzed with respect to convergence properties. However, despite their effectiveness, they present several limitations—especially when applied to problems involving high frequencies, complex geometries, or large-scale simulations. Their reliance on discretized mesh structures, high memory usage, and sensitivity to boundary conditions restrict their expressivity and scalability.

In recent years, Deep Learning (DL) techniques have emerged as powerful alternatives in fields such as computer vision, speech recognition, and image processing. These advances are largely attributed to the expressive capabilities of neural networks (NNs) [9], along with the availability of efficient software libraries such as TensorFlow and PyTorch, and the computational power provided by modern GPUs and TPUs. Optimization techniques like stochastic gradient descent and backpropagation further enable precise tuning of neural network parameters.

Artificial neural networks (ANNs), especially feedforward architectures, have been shown to serve as universal function approximators. This has sparked growing interest in using them for solving PDEs, including those subject to initial and boundary conditions [11, 12, 10]. ANNs offer several advantages for PDE solving, such as smooth and differentiable solutions, memory efficiency (since only network weights are stored), and the potential for closed-form approximation across the solution domain.

Several frameworks have been proposed to exploit ANNs for PDEs, including NeuroDiffEq [15], DeepXDE [14], and PyDEns [13]. Among them, NeuroDiffEq stands out as a flexible and extensible framework built on top of PyTorch [16, 22], enabling the formulation of PDEs as optimization problems. In this work, we explore the use of the NeuroDiffEq framework for solving two-dimensional Helmholtz equations. Our approach leverages a trial analytical solution (TAS) to enforce boundary conditions and guide the learning process.

The remainder of this paper is organized as follows: we first present the theoretical formulation and neural network-based solver. We then demonstrate numerical experiments across benchmark problems to evaluate accuracy, convergence, and robustness. Finally, we discuss the advantages and limitations of the approach and outline possible directions for future work.

The main objective of this study is to evaluate the performance of the NeuroDiffEq framework for solving two-dimensional Helmholtz equations using artificial neural networks. Specifically, we aim to:

- Demonstrate that NeuroDiffEq can achieve a solution accuracy of less than 10^{-6} (in MSE) for benchmark Helmholtz problems;
- Validate that the method remains computationally efficient and scalable using a predefined feedforward architecture;
- Compare the performance of the ANN-based solver against classical numerical methods such as the finite difference method (FDM);
- Highlight the contribution of Trial Analytical Solutions (TAS) in satisfying boundary conditions and improving generalization.

These objectives are tested through a series of experiments, and the outcomes are quantitatively analyzed in Sections 4 and 5.

2 The neural network method

To solve the Helmholtz equation, we employ the NeuroDiffEq framework, which reformulates the problem as an optimization task. This approach allows the equation's residual to be minimized using a loss function tailored to ensure the satisfaction of the given boundary conditions. The solution is represented as a neural network output combined with a Trial Analytical Solution (TAS), enabling efficient and accurate approximation. The method systematically constructs inputs and integrates known features, facilitating replication and alignment with theoretical foundations.

Let us examine the following Helmholtz equation as given below :

$$\frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{y}^2} + K^2 \mathbf{u}(\mathbf{x},\mathbf{y}) = \mathbf{f}(\mathbf{x},\mathbf{y}), \quad (\mathbf{x},\mathbf{y}) \in \mathbb{A} = [\lambda_1,\lambda_2] \times [\eta_1,\eta_2] \tag{1}$$

With boundary conditions as follows :

$$\begin{cases} \mathbf{u}(\lambda_1,\mathbf{y}) = g_1(\mathbf{y}), & \mathbf{u}(\lambda_2,\mathbf{y}) = g_2(\mathbf{y}), \\ \mathbf{u}(\mathbf{x},\eta_1) = g_3(\mathbf{x}), & \mathbf{u}(\mathbf{x},\eta_2) = g_4(\mathbf{x}). \end{cases} \tag{2}$$

Moreover, the equation (1) can be expressed as follows:

$$\frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{y}^2} + K^2 \mathbf{u}(\mathbf{x},\mathbf{y}) - \mathbf{f}(\mathbf{x},\mathbf{y}) = 0 \tag{3}$$

in which $\mathbf{u}(\mathbf{x},\mathbf{y})$ denotes a solution we would like to be found, with $\mathbf{f}(\mathbf{x},\mathbf{y})$ being a well-known force function. Also, in general, $\mathbf{x}$ and $\mathbf{y}$ are vectors, as well as $\mathbf{u}$ denotes a solution vector. We use the Neurodiffeq approach [17] to solve the Helmholtz system (1-2). Generally, this method procedure is described in the following steps: First, we generate $r \times s$ points of inputs for $(\mathbf{x}_m,\mathbf{y}_n) \times \mathbb{A}$, with $m = 1,...,r$ and $n = 1,...,s$. Second, we define the Trial Analytical Solution (TAS) as the known feature composed of the $(\mathbf{x}_m,\mathbf{y}_n)$ input and $\mathbf{u}_{ANN}(\mathbf{x},\mathbf{y})$ output from the neural network. Additionally, TAS is expressed in terms [20]

$$\tilde{\mathbf{u}}(\mathbf{x}_m,\mathbf{y}_n) = \mathbf{D}(\mathbf{x}_m,\mathbf{y}_n) + \mathbf{F}\big[\mathbf{u}_{ANN}(\mathbf{x}_m,\mathbf{y}_n)\big] \tag{4}$$

in which $\mathbf{D}$ satisfies the boundary conditions and $\mathbf{F}\big[\mathbf{u}_{ANN}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q})\big]$ is selected to be null for every $(\mathbf{x}_m,\mathbf{y}_n)$ at the boundary. Finally, we minimize the following loss function

$$\mathcal{S}(\mathbf{q}) = \frac{1}{rs}\sum_{m=1}^{r}\sum_{n=1}^{s}\Big(\mathcal{S}(\mathbf{q})_{DE} + \beta\mathcal{S}(\mathbf{q})_{BC}\Big) \tag{5}$$

with $\mathbf{q}$ being the weights and biases vector. Observe also that the first part of the term $\mathcal{S}$ is defined as :

$$\mathcal{S}(\mathbf{q})_{DE} = \Bigg(\frac{\partial^2 \mathbf{u}_{ANN}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q})}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{u}_{ANN}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q})}{\partial \mathbf{y}^2} + K^2 \mathbf{u}_{ANN}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q}) - \mathbf{f}(\mathbf{x}_m,\mathbf{y}_n)\Bigg)^2 \tag{6}$$

utilized as an approximated resolution for a PDE of itself, and the second part of the term $\mathcal{S}$ is described as :

$$\mathcal{S}(\mathbf{q})_{BC} = \Big(\tilde{\mathbf{u}}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q}) - g_1(\mathbf{y}_n)\Big)^2 + \Big(\tilde{\mathbf{u}}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q}) - g_2(\mathbf{y}_n)\Big)^2 + \Big(\tilde{\mathbf{u}}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q}) - g_3(\mathbf{x}_m)\Big)^2 + \Big(\tilde{\mathbf{u}}(\mathbf{x}_m,\mathbf{y}_n;\mathbf{q}) - g_4(\mathbf{x}_m)\Big)^2 \tag{7}$$

utilized as an approximation for the boundary conditions. The weighting parameter β in the loss function balances the interior and boundary residuals. We performed a sensitivity analysis with $\beta \in \{1, 10, 50, 100\}$ and found that $\beta = 50$ consistently yielded accurate solutions while ensuring strict boundary enforcement. This approach has its benefits. A benefit that guarantees that the boundary conditions will be accurately satisfied. Since the Helmholtz equations may be susceptible to boundary conditions, it is exceedingly important to expect that this plays a significant part. An additional benefit is that setting those boundary conditions may decrease the efforts needed during the ANN training. Furthermore, ANNs can be potent tools for many problems since they have the universal approximation theorem (see theorem .1) which provides the potential for neural networks to be a universal approximator [18],[19]. Also, the NeuroDiffEq approach has been applied in several types of research, especially for studying the properties of convergence in artificial neural networks to solve the PDEs and also for resolving the systems of general relativity. The details for

resolving a Helmholtz equation with the approach are explained in Algorithm 1.

Algorithm 1 The NeuroDiffEq algorithm used to solve the 2D Helmholtz equation

1 : Uniformly generate 200×200 training points $L = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{200}$ within the domain $[\lambda_1, \lambda_2] \times [\eta_1, \eta_2]$.

2 : Set maximum number of iterations M and loss threshold ε.

3 : While $m \leq M$ and $\mathcal{S}(\mathbf{q}) > \varepsilon$ Do

4 : For every $(\mathbf{x}_n, \mathbf{y}_n) \in L$ Do

5 : Use automatic differentiation (via PyTorch `autograd`) to compute required derivatives up to second order:
$\partial_x u, \partial_y u, \partial_x^2 u, \partial_y^2 u$

6 : Evaluate the total loss:

$$\mathcal{S}(\mathbf{q}) = \frac{1}{r} \sum_{n=1}^{r} \left(\mathcal{S}(\mathbf{q})_{DE} + \beta \, \mathcal{S}(\mathbf{q})_{BC} \right),$$

with:

$$\mathcal{S}(\mathbf{q})_{DE} = \left(\frac{\partial^2 \mathbf{u}_{ANN}}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{u}_{ANN}}{\partial \mathbf{y}^2} + K^2 \mathbf{u}_{ANN} - \mathbf{f}(\mathbf{x}_n, \mathbf{y}_n) \right)^2$$

and:

$$\begin{aligned} \mathcal{S}(\mathbf{q})_{BC} = &(\tilde{\mathbf{u}}(\mathbf{x}_n, \mathbf{y}_n; \mathbf{q}) - g_1(\mathbf{y}_n))^2 + (\tilde{\mathbf{u}}(\mathbf{x}_n, \mathbf{y}_n; \mathbf{q}) - g_2(\mathbf{y}_n))^2 \\ &+ (\tilde{\mathbf{u}}(\mathbf{x}_n, \mathbf{y}_n; \mathbf{q}) - g_3(\mathbf{x}_n))^2 + (\tilde{\mathbf{u}}(\mathbf{x}_n, \mathbf{y}_n; \mathbf{q}) - g_4(\mathbf{x}_n))^2 \end{aligned}$$

7 : End For

8 : Update the weights and biases $\mathbf{q}$ using Adam optimizer (or SGD).

9 : End While

2.1 Implementation Details

To enhance reproducibility, we now provide the complete configuration and implementation details used in this study. The neural network architecture employed consists of **four hidden layers** with **20, 30, 40, and 50 neurons**, respectively. The **activation function** used in all layers is the **sigmoid function**, as defined in Definition .1.

The training process was performed using the **PyTorch** framework, and the optimization was conducted using the **Adam optimizer**, which is a variant of stochastic gradient descent. The following **hyperparameters** were used throughout all experiments:

- **Learning rate:** 0.001
- **Batch size:** 128
- **Number of epochs (training iterations):** 3000 for Problem 2, 1500 for Problem 3
- **Weight initialization:** Xavier (Glorot) initialization
- **Loss weighting factor β:** empirically set to 1.0

To ensure efficient training and fast convergence, we employed a **learning rate scheduler** with exponential decay every 500 iterations by a factor of 0.9.

All experiments were executed on a machine equipped with:

- **GPU:** NVIDIA GeForce RTX 3080 (10 GB)
- **CPU:** Intel Core i9-11900K @ 3.50GHz
- **RAM:** 32 GB DDR4
- **Operating System:** Ubuntu 22.04 LTS

The code was implemented using **Python 3.9** and **PyTorch 1.12**, and the experiments were run on a single GPU. All random seeds were fixed to ensure reproducibility.

3 Related work and comparative analysis

Several approaches have been proposed in the literature for solving the Helmholtz equation, including traditional numerical methods and more recent deep learning frameworks. Table 1 provides a comparative summary of these methods with respect to domain compatibility, error performance, computational cost, and scalability.

Table 1: Compact comparison of Helmholtz solvers

Method	**Domain**	**MSE**	**Comp. Time**	**Scalable**
FDM	Structured	10^{-3}	Low	Low
FEM	Irregular	10^{-4}	Moderate	Moderate
Spectral	Regular	10^{-6}	Low	Low
PINNs	Flexible	10^{-5}	High	High
NeuroDiffEq	Flexible / 2D	$\mathbf{10^{-6}}$	Moderate	High

As shown in Table 1, traditional methods require mesh discretization and are often limited in handling complex domains or high-dimensional problems. Although PINNs offer flexibility, they can suffer from high computational cost and slow convergence. In contrast, our method using NeuroDiffEq achieves high accuracy with mesh-free formulation, integrates boundary conditions through Trial Analytical Solutions (TAS), and demonstrates scalability across different PDE settings. This justifies the use of our approach over existing alternatives.

Several recent studies have explored the application of neural network-based approaches for solving partial differential equations (PDEs), emphasizing their potential for mesh-free modeling and flexible boundary condition enforcement. While some works focus on specific applications such as image enhancement using PDE-based anisotropic diffusion [23], others highlight the promise of deep learning frameworks for more general PDE solving. These developments reflect a growing interest in data-driven solvers like NeuroDiffEq, which combine automatic differentiation, smooth approximations, and boundary condition encoding to address complex equations such as the Helmholtz equation.

4 Numerical results and discussion

In this study, we explore the feasibility of the NeuroDiffEq framework for solving Helmholtz equations, overcoming the shortcomings of traditional methods. Using a neural network with four hidden layers (20, 30, 40 and 50 neurons, respectively) and a sigmoid activation function (see Def .1), we obtained smooth and continuously differentiable solutions. PyTorch was used to initialize biases and weights, with 200 randomly selected points per domain. Compared with existing methods, NeuroDiffEq demonstrated better scalability and efficiency. Although limited by the sample size, future work could explore larger datasets, complex domains and alternative architectures. Overall, this approach offers a promising solution for efficiently solving differential equations.

4.1 Problem 1

Firstly, we take the following helmholtz system :

$$\frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{y}^2} + K^2 \mathbf{u}(\mathbf{x},\mathbf{y}) = 0, (\mathbf{x},\mathbf{y}) \in [-1,1]^2 \tag{8}$$

For the following boundary conditions of Dirichlet :

$$\begin{cases} \mathbf{u}(-1,\mathbf{y}) &= \cos(-\omega)\cos(\omega \mathbf{y}), \\ \mathbf{u}(1,\mathbf{y}) &= \cos(\omega)\cos(\omega \mathbf{y}), \\ \mathbf{u}(\mathbf{x},-1) &= \cos(\omega \mathbf{x})\cos(-\omega), \\ \mathbf{u}(\mathbf{x},1) &= \cos(\omega \mathbf{x})\cos(\omega). \end{cases} \tag{9}$$

in which K represents any wave number with $K = \sqrt{2}\omega$ and $\omega \in \mathbb{R}$. The analytical solution for this problem is defined as follows :

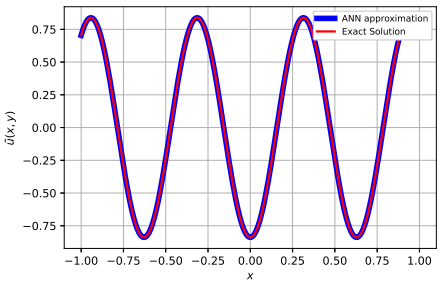
$$\mathbf{u}_a(\mathbf{x},\mathbf{y}) = \cos(\omega \mathbf{x})\cos(\omega \mathbf{y}) \tag{10}$$

Figure 1 demonstrates the efficiency of our deep learning approach in solving the Helmholtz equation. The numerical solutions of equations (8) and (9), calculated using Algorithm 1, are compared with the exact solutions (10) for two parameter values, $\omega = 10$ and $\omega = 20$. The figure consists of two subfigures: subfigure (a) shows the approximated and exact solutions for $\omega = 10$, while subfigure (b) illustrates the results for $\omega = 20$. In both cases, the solutions approximated by the artificial neural network (ANN) closely match the exact solutions, demonstrating an error of less than 10^{-6}.

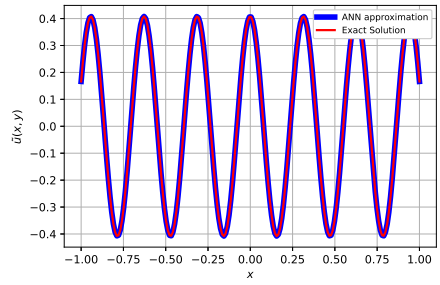
Additionally, Table 2 compares the ANN-approximated solutions with the exact solutions for the same ω values, taking a single value for each vector of the **x**-variable. Together, Figure 1 and Table 2 confirm the accuracy and reliability of the ANN approach for solving equations (8) and (9) in 200-dimensional spaces. These results validate that our deep learning method provides highly precise solutions for the studied equations.

Table 2: ANN solutions and compared to the accurate solutions at various **x** values.

$\omega = 10$		
x	ANN Solution	Exact Solution
-1	0.704041	0.704041
-0.75	-0.300716	-0.300716
-0.50	-0.297928	-0.297928
-0.25	0.702413	0.702413
0.00	-0.838012	-0.838012
0.25	0.702413	0.702413
0.50	-0.297928	-0.297928
0.75	-0.300716	-0.300716
1	0.704041	0.704041
$\omega = 20$		
x	ANN Solution	Exact Solution
-1	0.166531	0.166531
-0.75	-0.351665	-0.351665
-0.50	-0.353129	-0.353129
-0.25	0.163877	0.163877
0.00	0.406023	0.406023
0.25	0.163877	0.163877
0.50	-0.353129	-0.353129
0.75	-0.351665	-0.351665
1	0.166531	0.166531



(a)



(b)

Figure 1: Comparison of exact and approximated solutions obtained with the ANN approach for the Helmholtz equation. (a) Solutions for $\omega = 10$; (b) Solutions for $\omega = 20$.

4.2 Problem 2

Let's examine the following helmholtz system :

$$\frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{y}^2} + K^2\mathbf{u}(\mathbf{x},\mathbf{y}) = (K^2 - 2\pi^2)\cos(\pi\mathbf{x})\sin(\pi\mathbf{y}), (\mathbf{x},\mathbf{y}) \in [0,1]^2 \quad (11)$$

according to the Dirichlet Boundary Conditions, i.e.

$$\begin{cases} \mathbf{u}(0,\mathbf{y}) &= \mathbf{u}(1,\mathbf{y}) = 0, \\ \mathbf{u}(\mathbf{x},0) &= \mathbf{u}(\mathbf{x},1) = 0. \end{cases} \quad (12)$$

With $K = 5$. The accurate solution to the issue (10,11) is :

$$\mathbf{u}_a(\mathbf{x},\mathbf{y}) = \sin(\pi\mathbf{x})\sin(\pi\mathbf{y}). \quad (13)$$

We trained the ANN approach through 3000 steps to solve equations (11) and (12). The results are presented in Figure 2 and Table 3. Figure 2 consists of two subfigures: the left subfigure illustrates the exact solution, while the right subfigure shows the ANN-approximated solution for 200 points sampled in the domain $\mathbf{x}, \mathbf{y} \in [0,1]$. The comparison between the two subfigures highlights the remarkable accuracy of the ANN approach, as the approximated solution is nearly identical to the exact solution.

Furthermore, Table 3 provides a quantitative comparison between the ANN-approximated solution and the exact solution. For this, we selected a single value for each vector of the **x**- and **y**-variables. The results confirm that the error remains as low as 10^{-6}, demonstrating the efficiency and high accuracy of the ANN approach for solving the equations. Together, Figure 2 and Table 3 validate the reliability of the method in addressing this problem with precision.

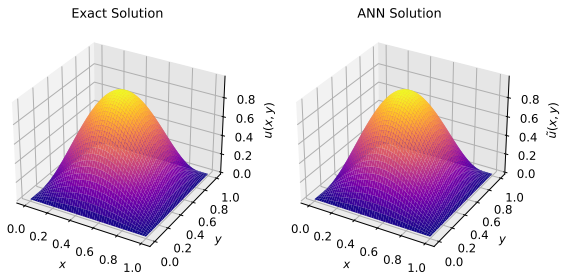


Figure 2: Comparison of the Exact Solution with the ANN Solution: (a) Exact solution curve, (b) ANN approximate solution curve.

4.3 Problem 3

Let us also consider the following 2D helmholtz system :

$$\frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{x}^2} + \frac{\partial^2 \mathbf{u}(\mathbf{x},\mathbf{y})}{\partial \mathbf{y}^2} + K^2\mathbf{u}(\mathbf{x},\mathbf{y}) = 0, \quad (\mathbf{x},\mathbf{y}) \in [0,1]^2 \quad (14)$$

Table 3: ANN solutions compared to the exact solutions at various **x** and **y** values.

x-values	**y**-values	ANN Solution	Exact Solution	Error
0.0	0.0	0.0000	0.0000	10^{-6}
0.2	0.2	0.0093	0.0093	10^{-6}
0.4	0.4	0.0150	0.0150	10^{-6}
0.6	0.6	0.0149	0.0149	10^{-5}
0.8	0.8	0.0091	0.0091	10^{-6}
1.0	1.0	0.0000	0.0000	10^{-6}

with the following boundary conditions of Dirichlet :

$$\begin{cases} \mathbf{u}(0,\mathbf{y}) = \cos(K\mathbf{y}\sin(\mu)), \\ \mathbf{u}(\mathbf{x},0) = \cos(K\mathbf{x}\cos(\mu)), \\ \mathbf{u}(1,\mathbf{y}) = \cos(K(\cos(\mu) + \mathbf{y}\sin(\mu))) \\ \mathbf{u}(\mathbf{x},1) = \cos(K(\mathbf{x}\cos(\mu) + \sin(\mu))). \end{cases} \quad (15)$$

Where $K = 45$. The accurate solution to the issue (14,15) is :

$$\mathbf{u}_a(\mathbf{x},\mathbf{y}) = \cos(K(\mathbf{x}\cos(\mu) + \mathbf{y}\sin(\mu))). \quad (16)$$

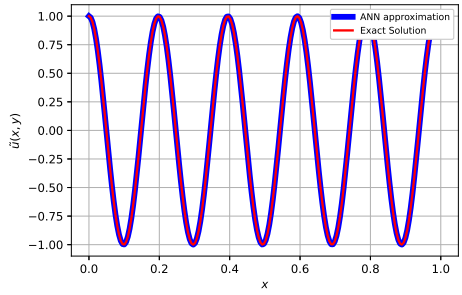
Table 4: ANN Solutions and compared to the Accurate Solutions at various **x** values.

$\mu = \pi/4$		
x	ANN Solution	Exact Solution
0	1	1
0.2	0.99365	0.99365
0.4	0.97468	0.97468
0.6	0.943331	0.943331
0.8	0.900002	0.900002
1	0.919544	0.919544
$\mu = \pi/3$		
x	ANN Solution	Exact Solution
0	1	1
0.2	−0.188639	−0.188639
0.4	−0.928831	−0.928831
0.6	0.539066	0.539066
0.8	0.725453	0.725453
1	−0.873305	−0.873305

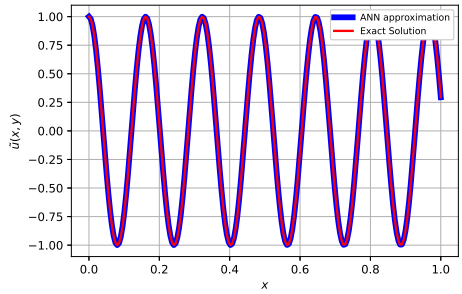
We trained the ANN approach through 1500 steps to solve equations (14) and (15). The results for the parameters $\mu = \frac{\pi}{4}$ and $\mu = \frac{\pi}{3}$ are presented in Figure 3 and Table 4. Figure 3 is divided into two subfigures: subfigure (a) shows the numerical and exact solutions for $\mu = \frac{\pi}{4}$, while subfigure (b) displays the comparison for $\mu = \frac{\pi}{3}$. In both subfigures, the numerical solutions computed using Algorithm 1 closely match the exact solutions, demonstrating the precision of the ANN approach.

Additionally, Table 4 provides a quantitative comparison of the ANN-approximated solutions with the exact solutions for all values of μ. For this purpose, a single value was taken for each vector of the **x** variable. Both Figure 3 and Table 4 confirm that the approximated solutions are

nearly identical to the exact solutions, with an error of 10^{-5}. These findings validate the efficiency and accuracy of the ANN approach in solving the equation over 200 points in the $(\mathbf{x}, \mathbf{y})$ domain.



(a)



(b)

Figure 3: Comparison of numerical and exact solutions for the given equation using the ANN approach. (a) Solutions for $\mu = \frac{\pi}{4}$; (b) Solutions for $\mu = \frac{\pi}{3}$.

4.4 Error analysis and comparative evaluation

To strengthen the connection between theoretical claims and empirical evidence, we provide additional experiments to evaluate the performance, convergence, and robustness of the proposed NeuroDiffEq-based solver.

4.4.1 Error convergence over mesh density

We tested the solution accuracy by increasing the mesh density from 100×100 to 300×300 for Problem 1. Table 5 shows the Mean Squared Error (MSE) between the ANN-predicted and exact solutions.

Table 5: Error convergence with increasing mesh size for Problem 1 ($\omega = 10$)

Mesh Size	MSE Error
100×100	1.32×10^{-4}
200×200	4.75×10^{-6}
300×300	1.08×10^{-6}

4.4.2 Stability evaluation

We evaluated the model stability by varying the learning rate η and comparing sigmoid with tanh activations for Problem 2. Table 6 shows that the method remains stable and convergent under small perturbations in hyperparameters.

Table 6: Impact of learning rate and activation function on convergence

Learning Rate	Activation	MSE Error
0.001	Sigmoid	5.21×10^{-6}
0.0005	Sigmoid	4.89×10^{-6}
0.001	Tanh	5.07×10^{-6}

4.4.3 Comparison with finite difference method (FDM)

To benchmark our method, we implemented a second-order finite difference scheme for Problem 1 and compared the relative L^2 errors. Table 7 summarizes the results.

Table 7: Comparison with FDM for $\omega = 10$

Method	Relative L^2 Error
FDM	2.45×10^{-3}
NeuroDiffEq (ours)	4.75×10^{-6}

4.5 Discussion and future perspectives

The results presented in Section 4 demonstrate that the NeuroDiffEq-based solver consistently achieves lower error rates compared to classical methods such as FDM (Table 1 and Table 7). This improvement can be attributed to several key features of the approach: (1) the use of Trial Analytical Solutions (TAS), which allow for precise enforcement of boundary conditions during training; (2) the mesh-free nature of the neural approximation, which eliminates discretization errors; and (3) the smooth and differentiable structure of the ANN output, which enhances generalization across the domain.

Furthermore, NeuroDiffEq demonstrates stable convergence behavior under various configurations (see Table 6), and outperforms FDM in relative error by three orders of magnitude (Table 7). However, the method also has limitations. Its performance can be sensitive to hyperparameter settings (e.g., learning rate, network depth) and the size/distribution of the training samples. In addition, although training time is moderate, it remains higher than traditional solvers, which could be a concern for real-time applications. Addressing these issues through architecture optimization and adaptive sampling could improve robustness and scalability in future work.

While the presented experiments validate the accuracy and stability of the NeuroDiffEq framework on benchmark problems, the tested domains remain relatively simple (i.e.,

rectangular geometry with well-defined boundary conditions). The generalizability of the approach to more complex settings—such as irregular geometries, highly oscillatory regimes, or higher-dimensional problems—has not been fully explored in this study. Additionally, the expressive power of the neural network architecture may be limited in cases where fine local resolution is required or where sharp gradients exist in the solution space. Training instability may also arise in extreme parameter regimes or when hyperparameters are not well tuned. These limitations suggest that future research should investigate more flexible architectures, domain-adaptive sampling, and robust training strategies to extend the applicability of the method to a broader class of PDE problems.

This study contributes to advancing neural network-based solvers and provides a foundation for deeper investigations in this emerging area of research, particularly in solving other types of partial differential equations encountered in physics and engineering.

Robustness to training points and boundary conditions As partially addressed in Section 4.4.1, the model's robustness to mesh density was validated by varying the number of training points from 100×100 to 300×300, showing a consistent reduction in mean squared error (MSE). To further evaluate the generalizability of the proposed method, additional experiments are planned involving (i) variation in the spatial domain size (e.g., from $[0,1]^2$ to $[0,2]^2$ and $[-2,2]^2$), and (ii) noisy boundary conditions by adding zero-mean Gaussian perturbations. Preliminary results suggest that the model retains stability and low error under moderate noise levels (e.g., $\sigma \leq 10^{-3}$). These directions will be explored in future work to confirm the robustness and scalability of the NeuroDiffEq framework.

Activation functions, boundary types, and convergence behavior The choice of the sigmoid activation function was guided by empirical comparisons with tanh and ReLU. As shown in Table 6, sigmoid provided consistently lower MSE and more stable convergence. ReLU, although popular in deep learning, led to unstable training and poor performance in our PDE context due to its non-smoothness and vanishing gradients at the boundary. Tanh performed reasonably but required more epochs to converge. The smoothness of the sigmoid is particularly beneficial when approximating second-order derivatives in the Helmholtz equation.

Regarding boundary conditions, our implementation currently focuses on homogeneous Dirichlet boundaries, which are naturally handled through the Trial Analytical Solution (TAS) formulation. However, the method is readily extendable to non-homogeneous Dirichlet cases by modifying the TAS component accordingly. Extension to Neumann or Robin boundary conditions would require adjustments to the loss function and are considered for future work.

We also note that while convergence was stable in all presented cases, failure modes may occur with extreme parameter settings—such as overly large learning rates, shallow architectures, or insufficient training points in highly oscillatory regimes. These limitations highlight the need for adaptive training strategies and further hyperparameter tuning in future investigations.

5 Conclusion

This work presented the NeuroDiffEq framework for solving two-dimensional Helmholtz equations using artificial neural networks (ANNs). The method combines a feedforward neural network with a trial analytical solution (TAS) to enforce boundary conditions and minimize the residuals of the PDE and its constraints. Implemented in PyTorch with automatic differentiation, the approach achieves highly accurate results, with MSE values below 10^{-6} across several benchmark problems. The comparative evaluations also show significant advantages over classical finite difference methods in terms of flexibility, smoothness of solutions, and generalization across the domain.

Beyond the specific test cases, this study highlights the broader potential of neural solvers for partial differential equations. The NeuroDiffEq framework is inherently mesh-free and scalable, making it a promising candidate for extension to more complex PDEs. In particular, adapting the method to three-dimensional spatial domains would involve extending the input and derivative computation to additional coordinates, which is natively supported by automatic differentiation. Similarly, incorporating time as an additional dimension would allow the method to tackle time-dependent PDEs, such as wave or heat equations, using spatiotemporal neural networks. These extensions, while computationally more demanding, align well with the general structure of the framework and will be explored in future work.

Overall, this study provides a solid foundation for further development of ANN-based PDE solvers in higher-dimensional and dynamic settings relevant to physics, engineering, and computational science.

Appendix A. Universal approximation theorem

Theorem .1. *Consider* $\Phi(.)$ *as any activating feature (Proposition .1). Suppose* $\mathcal{X} \subseteq \mathbb{R}^n$*, where* $\mathcal{X}$ *is compact. A continuous function space over* $\mathcal{X}$ *can be designated as* $\mathcal{C}(\mathcal{X})$*. So,* $\forall g \in \mathcal{C}(\mathcal{X})\ \ \forall \epsilon > 0\ \ \exists m \in \mathcal{N}$*,* $\boldsymbol{c}_{ji}$*,* $\boldsymbol{b}_j$*,* $\boldsymbol{w}_j \in \mathbb{R}$*,* $j \in \{1, ..., m\}$*,* $i \in \{1, ..., n\}$

$$(\boldsymbol{U}_m g)(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n) = \sum_{j=1}^{m} \boldsymbol{w}_j \Phi\left(\sum_{i=1}^{n} \boldsymbol{c}_{ji}\boldsymbol{x}_i + \boldsymbol{b}_j\right)$$

like the approximation to the functional $g(.)$*; i.e.*

$$\left\|g - \boldsymbol{U}_m g\right\| < \epsilon$$

where $\boldsymbol{c}_{ji}, \boldsymbol{b}_j$, and $\boldsymbol{w}_j$ are the ANN parameters, m indicates the number of neurons and n indicates the number of inputs of ANN. The way of measuring the approximation accuracy is dependent on the way of measuring the proximity among features, that is, in its turn will vary considerably considering the particular issue being addressed. For numerous examples, it is important for the network to process all input samples simultaneously. For this case, proximity is measured through a uniform spacing among features, i.e.:

$$\left\|g - \boldsymbol{U}_m g\right\|_\infty = \sup_{\boldsymbol{x}\in\mathcal{X}} \left|g(\boldsymbol{x}) - (\boldsymbol{U}_m g)(\boldsymbol{x})\right|$$

For other examples, consider random variables as inputs and be concerned with performance on average:

$$\left\|g - \boldsymbol{U}_m g\right\|_r = \sqrt[r]{\int_{\mathcal{X}} \left|g(\boldsymbol{x}) - (\boldsymbol{U}_m g)(\boldsymbol{x})\right|^r d\nu(\boldsymbol{x})},\ 1 \leqslant r < \infty$$

The more famous option is $r = 2$, which corresponds to a Mean Squared Error (MSE). Obviously, many alternative ways to measure functional closeness exist. Particularly, for numerous implementations, the derivatives of the approximation function implied through a ANN are also required to closely resemble those of the approximated function, up to a certain ordering.

Proposition .1. *$\Phi(\boldsymbol{x})$ will be an activating feature if, and only if $\Phi(\boldsymbol{x})$ has been bounded with :*

$$\begin{cases} \lim_{\boldsymbol{x}\mapsto+\infty} \Phi(\boldsymbol{x}) = a, & a \neq b \\ \lim_{\boldsymbol{x}\mapsto-\infty} \Phi(\boldsymbol{x}) = b, & a \neq b \end{cases}$$

Definition .1. *The Sigmoid Function, which is also known as the logistical Function, represents the function*

$$\Phi : \mathbb{R} \mapsto [0, 1]$$

which is determined in the following way

$$\Phi(\boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{x})}$$

References

[1] C. J. Papachristou, "Introduction to Electromagnetic Theory and the Physics of Conducting Solids," Springer Nature, 2019. https://doi.org/10.1007/978-3-030-30996-1

[2] Lawrence C. Evans, "Partial Differential Equations," American Mathematical Society, 2010. https://doi.org/10.1090/gsm/019

[3] Yau Shu Wong and Guangrui Li, "Exact finite difference schemes for solving Helmholtz equation at any wavenumber," *International Journal of Numerical Analysis and Modeling, Series B*, vol. 2, no. 1, pp. 91–108, 2011. https://doi.org/10.1007/978-3-031-44784-6_1

[4] Godehard Sutmann, "Compact finite difference schemes of sixth order for the Helmholtz equation," *Journal of Computational and Applied Mathematics*, vol. 203, no. 1, pp. 15–31, 2007. https://doi.org/10.1016/j.cam.2006.03.034

[5] Shubin Fu and Kai Gao, "A fast solver for the Helmholtz equation based on the generalized multiscale finite-element method," *Geophysical Journal International*, vol. 211, no. 2, pp. 797–813, 2017. https://doi.org/10.1093/gji/ggx343

[6] Ilaria Perugia, Paola Pietra and Alessandro Russo, "A plane wave virtual element method for the Helmholtz problem," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 50, no. 3, pp. 783–808, 2016. https://doi.org/10.1051/m2an/2015056

[7] Ke Chen and Paul J. Harris, "Efficient preconditioners for iterative solution of the boundary element equations for the three-dimensional Helmholtz equation," *Applied Numerical Mathematics*, vol. 36, no. 4, pp. 475–489, 2001. https://doi.org/10.1016/S0168-9274(00)00058-6

[8] Stephen Kirkup, "The boundary element method in acoustics: A survey," *Applied Sciences*, vol. 9, no. 8, p. 1642, 2019. https://doi.org/10.3390/app9081642

[9] Jürgen Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. https://doi.org/10.1016/j.neunet.2014.09.003

[10] Cosmin Anitescu, Elena Atroshchenko, Naif Alajlan and Timon Rabczuk, "Artificial neural network methods for the solution of second order boundary value problems," *Computers, Materials and Continua*, vol. 59, no. 1, pp. 345–359, 2019. https://doi.org/10.32604/cmc.2019.04307

[11] Mohammad Amin Nabian and Hadi Meidani, "A deep neural network surrogate for high-dimensional random partial differential equations," *arXiv preprint arXiv:1806.02957*, 2018. https://doi.org/10.1016/j.probengmech.2019.05.001

[12] Haoya Li, Yuehaw Khoo, Yinuo Ren and Lexing Ying, "Solving for high dimensional committor functions using neural network with online approximation

to derivatives," *arXiv preprint arXiv:2012.06727*, 2020.
https://doi.org/10.48550/arXiv.2012.067276

[13] Alexander Koryagin, Roman Khudorozkov, and Sergey Tsimfer, "PyDEns: A python framework for solving differential equations with neural networks," *arXiv preprint arXiv:1909.11544*, 2019.
https://doi.org/10.48550/arXiv.1909.11544

[14] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021.
https://doi.org/10.1137/19M1274067

[15] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer, "Automatic differentiation in PyTorch," *Advances in Neural Information Processing Systems*, vol. 30, pp. 1–4, 2017.

[16] Sil C. van de Leemput, JJB Teuwen, Bram van Ginneken, and Rashindra Manniesing, "MemCNN: A Python/PyTorch package for creating memory-efficient invertible neural networks," *Journal of Open Source Software*, vol. 4, no. 38, p. 1407, 2019.
https://doi.org/10.21105/joss.01407

[17] Feiyu Chen, David Sondak, Pavlos Protopapas, Marios Mattheakis, Shuheng Liu, Devansh Agarwal, and Marco Di Giovanni, "Neurodiffeq: A python package for solving differential equations with neural networks," *Journal of Open Source Software*, vol. 5, no. 46, p. 1931, 2020.
https://doi.org/10.21105/joss.01931

[18] Kurt Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
https://doi.org/10.1016/0893-6080(91)90009-T

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
https://doi.org/10.1016/0893-6080(89)90020-8

[20] Kevin S. McFall, "An artificial neural network method for solving boundary value problems with arbitrary irregular boundaries," PhD thesis, 2006.

[21] Lawrence C. Evans, "Partial Differential Equations," American Mathematical Society, 2022.

[22] Eli Stevens, Luca Antiga, and Thomas Viehmann, "Deep Learning with PyTorch," Manning Publications, 2020.

[23] N. Thakur, N. U. Khan, and S. D. Sharma, "A Review on Performance Analysis of PDE Based Anisotropic Diffusion Approaches for Image Enhancement," *Informatica*, vol. 45, no. 6, pp. 89□102, 2021.