Research in Industrial Projects for Students

Sponsor

Advanced Micro Devices, Inc.

Final Report

Surrogate Modeling Investigation of Scientific Applications with Neural Networks

Student Members

Emma Hayes (Project Manager), *Carnegie Mellon University*,
eehayes@andrew.cmu.edu

Pelin Erşin, *Izmir University of Economics*

Peter Matthews, *University of Warwick*

Paramjyoti Mohapatra, *Case Western Reserve University*

Academic Mentor

Elisa Negrini, enegrini@math.ucla.edu

Sponsoring Mentor

Karl Schulz, karl.schulz@amd.com

Consultants

Tamima Rashid

Tushar Chouhan

August 16, 2023

This project was jointly supported by AMD and NSF Grant DMS 1925919.

Abstract

Neural networks are a powerful tool which can be used to provide numerical solutions for scientific problems. However, the addition of physical constraints into the network's loss function can help improve accuracy under the constraints of limited training data. This is where Physics Informed Neural Networks (PINNs) become particularly useful to predict the solution to PDEs. For our project, we specifically worked with the 2D acoustic wave equation where pressure and velocity vary over both space and time under the influence of a driving force at a source point. This equation plays an important role in mechanics, and specifically seismic wave activity. For our project, we trained neural networks to asses their ability to both interpolate between training data points and extrapolate beyond training points. We used a Discontinuous Galerkin (DG) method to generate ground truth data, utilizing a variety of time steps and mesh sizes. This data was treated as ground truth and used to train and test multiple different neural network architectures to predict the solution to the 2D wave equation. Our sponsor, Advanced Micro Devices (AMD), is a company which develops high-performance processor technologies such as Central Processing Units (CPUs) and Graphics Processing units (GPUs). We were given access to AMD's HPC system, allowing us to run our code in parallel on their GPUs. This project is of interest to our sponsor because they offer CPUs and GPUs that could be useful to shorten the run time and reduce the cost of such computations.

Acknowledgments

The authors would like to thank Dr. Karl Schulz and Dr. Laurent White of AMD. Karl has provided us with invaluable resources, including code created by Laurent. He has also given us access to AMD super computers where we were able to run our code remotely, and utilize parallel computing. We are also grateful for the guidance and advice offered by Tamima Rashid and Dr. Tushar Chouhan of AMD.

We would also like to thank our Academic Mentor Dr. Elisa Negrini. Elisa has provided us with guidance and support that we would not be able to complete this project without. The technical support provided by James Kimmick and David Medina is much appreciated. Finally, we would like to thank both Dr. Susana Serna and Dr. Aida Velasco for their time, energy, and dedication in running the Research in Industrial Projects for Students (RIPS) program.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Advanced Micro Devices

Advanced Micro Devices (AMD) is a semiconductor manufacturing company based in Santa Clara, California. The company develops high-performance processor technologies such as Central Processing Units (CPUs) and Graphics Processing Units (GPUs) for business and consumer markets (see Chapter 2). Not only does AMD develop these technologies, but they also look for ways for them to be involved in traditional science. AMD has an "Artificial Intelligence (AI) for science mission" in which they seek out traditional science partners to help them develop faster ways of performing their tasks. AMD is involved in scientific computing, machine learning, and data analytics. In these fields, they look for ways their technologies can benefit the methods already in place and drive forward scientific advancement.

1.2 Motivation

In Machine Learning, developing neural networks beyond the classification tasks, particularly in scientific computing, is a notable area of interest. Specifically, in recent years interest has been growing in developing neural network-based methods to solve Partial Differential Equations (PDEs) ([20], [1]). At present, classical numerical algorithms such as Discontinuous Galerkin methods produce results with high accuracy [12]. However, these algorithms are computationally expensive, have long run times, and often produce results on a pre-determined grid.

After they have been trained, neural networks are a valuable option since they are fast to evaluate and can produce results on the full data domain. However, a standard neural network may yield less accurate results than a numerical algorithm if not enough training data is available. To achieve both accuracy and efficiency, the 2023 RIPS AMD team created a Physics Informed Neural Network (PINN), which is a neural network endowed with physical constraints.

Our team developed and analyzed the performance of certain fast surrogate models for predicting the solution of the 2D acoustic wave equation. This PDE model, which simulates wave propagation in a 2D spatial domain, is used in multiple areas such as oil and gas exploration and predictions of seismic activities. The aim of the project was to create a model that is faster than classical numerical algorithms, while retaining their accuracy. We attempted to achieve this goal using Physics Informed Neural Networks and by combining

high performance computing and traditional science. The high performance computing systems that we have access to allowed us to simulate data and train our networks on both CPUs and GPUs. Once we had networks that made sufficiently accurate predictions, we were able to benchmark the different methods of modelling the wave equation on both CPUs and GPUs. With these different methods, we trained our network on a coarse grid of data, and once trained then produced a much finer mesh of data more quickly than a classical numerical method could.

1.3 Approach

We examined the two dimensional acoustic wave equation, which can be characterized as follows:

$$\frac{\partial p}{\partial t} + \kappa \cdot \nabla \mathbf{v} = f_s(\mathbf{x}, t) \tag{1.1}$$

$$\frac{\partial \mathbf{v}}{\partial t} + \frac{1}{\rho}\nabla p = 0 \tag{1.2}$$

$$\mathbf{x} = (x, y) \in [0, 1] \times [0, 1]$$

$$t \in [0, 2]$$

where f_s is the driving force from sources at different spatial locations, p is the pressure at any time step, κ and ρ are material properties and $\mathbf{v}$ is the velocity at any time step. Time is represented as t and $\mathbf{x}$ refers to the vector in space.

The 2022 RIPS AMD team demonstrated that the embedding of physics information in neural networks reduces the model uncertainty while improving the accuracy using the wave equation. As the 2023 team, we focused more on the speed of computation with PINNs instead of their accuracy as compared to neural networks. Specifically, we tried to improve accuracy while balancing it with the computing time. Another difference between the two projects is that modelling the wave using the forcing term, f_s, introduced discontinuities into the problem that were not studied last year.

We used a Discontinuous Galerkin (DG) method to generate solutions to the 2D acoustic wave equation with different sources [20]. This data was considered the ground truth and was used to train PINNs that act as surrogate models for the system [19]. We assessed the network's ability to interpolate between training points using a driving forcing function that varies in space and functional form.

As the major goal for the project was to balance accuracy and speed, we analyzed the time it takes for our models to make their predictions versus the accuracy of those predictions. The developed surrogate models were evaluated based on their prediction accuracy and their performance against the ground truth at multiple receiver locations. We then improved the efficiency of elements of the learning scheme and network architecture.

AMD granted us allocation on a remote high-performance computing (HPC) system with multiple GPU accelerators per node and high-speed interconnect. These resources allowed us controlled evaluation of various run time performances, and provided an opportunity to explore performance enhancements via GPU acceleration.

1.4 Overview

We begin with Chapter 2 where we discuss the background of our project in detail. We give a literature review on the acoustic wave equation, Discontinuous Galerkin methods,

neural networks, and physics informed neural networks. An insight of our data generation process as well as a comparison on CPUs and GPUs are also in this chapter. In Chapter 3, we introduce our neural network, its architecture and the loss function we have used for it. Lastly, we share the results we have obtained so far in Chapter 4 and discuss what can be done to improve them.

Chapter 2

Background

2.1 The Acoustic Wave Equation

The acoustic wave equations - see Equation (1.1) and Equation (1.2) - govern waves that act as small perturbations relative to the background level of a field, such as the pressure of a sound wave, or waves on the surface of a body of water [5].

In general, there are two ways to use this paradigm to model a wave propagating from a source, using initial conditions or forcing terms. When modelling a wave via initial conditions, it is assumed that at $t = 0$, there is a disturbance in the pressure field given by, for example, a Gaussian impulse shown below. The wave then propagates in response to this pressure differential.

$$p(0, \boldsymbol{x}) = a \cdot \exp\left(-\frac{1}{2\sigma^2}||\boldsymbol{x} - \boldsymbol{x}_s||^2\right) \tag{2.1}$$

Here, a is the amplitude of the wave and the standard deviation, σ, controls how localised the impulse is at $t = 0$. The $f_s(\boldsymbol{x}, t)$ term in Equation (1.1) is zero as we assume there is no further influence or external force on the system. This regime leads to continuous solutions with bounded derivates, making them easier to deal with numerically.

In contrast, we could also study wave phenomena by explicitly modeling the external forces on the system via $f_s(\boldsymbol{x}, t)$. One common choice is to model these forces using the Ricker wavelet [21], shown in Equation (2.2). We used the Ricker wavelet when modelling wave propagation throughout the rest of this work.

$$f_s(\boldsymbol{x}, t) = \frac{\tau}{\pi}\left(1 - (2\pi\omega)^2(t - t_0)^2\right)\exp\left(-\frac{1}{2}\left[(2\pi\omega)^2(t - t_0)^2 + 2\tau||\boldsymbol{x} - \boldsymbol{x}_s||^2\right]\right) \tag{2.2}$$

The parameters of interest, that control the dynamics of the wave, are the steepness τ and peak frequency ω. In the idealised case of $\tau = \infty$ on an infinite grid, the resulting PDE is solved by a sinusoidal wave of frequency ω propagating from a point source $\boldsymbol{x}_s$. However, in this case the resulting forcing term f is the Dirac δ function, leading to discontinuities in the data. Thus, in practice we chose $1 << \tau < \infty$ to model waves with small sources whilst maintaining a continuous forcing term that allows us to solve the PDE numerically. Moreover, the spacial derivatives f_s are bounded uniformly by $M\sqrt{\tau}$. Picking the exact value of τ is thus a tradeoff between more idealistic wave-like dynamics (larger values) and "smoother" forcing terms that easier to deal with numerically (smaller values).

2.2 Discontinuous Galerkin Methods

Discontinuous Galerkin (DG) methods are a set of powerful numerical techniques used for approximating solutions of partial differential equations (PDEs). We used an implementation of these methods that is very similar to the treatment of Cockburn and Shu in [3]. This generates the data that we considered "ground truth" for the purpose of training neural networks.

In the DG method considered here, the space domain $[0,1] \times [0,1]$ is partitioned into a rectangular $n \times n$ grid of smaller blocks. In each block, the solution, p, is approximated by a polynomial $\psi(\mathbf{x})$, with coefficients W_j that evolve over time [20]. In other words, we can approximate the solution on each block as

$$p(\boldsymbol{x}, t) = \sum_{j=1}^{n} W_j(t)\psi_j(\boldsymbol{x}) \tag{2.3}$$

If the basis, $\{\psi_j\}_{1 \leq j \leq n}$, of polynomials is chosen to be orthogonal, then the solution $p(t, \boldsymbol{x})$ must follow Equation (2.4)

$$\frac{dW_j}{dt} = \frac{1}{\int_\Omega |\psi_j(\boldsymbol{x})|^2 d\boldsymbol{x}} \left[\int_\Omega \nabla\psi_j v d\boldsymbol{x} - \int_{\partial\Omega} \psi_j h(\boldsymbol{w}^+, \boldsymbol{w}^-, \boldsymbol{n}) dS + \int_\Omega \psi_j(\boldsymbol{x}) f_s(\boldsymbol{x}, t) dx \right] \tag{2.4}$$

Where $v(\boldsymbol{x}, t)$ is the velocity field of the solution, and $S = (\boldsymbol{w}^+, \boldsymbol{w}^-)$ is the field at the boundary between two cells. The numerical flux, h, allows for the exchange of information between blocks by satisfying the following equations.

$$h(\boldsymbol{w}^+, \boldsymbol{w}^-, \boldsymbol{n}) = -h(\boldsymbol{w}^-, \boldsymbol{w}^+, -\boldsymbol{n}) \tag{2.5}$$

$$h(\boldsymbol{w}, \boldsymbol{w}, n) = v(\boldsymbol{w}) \cdot \boldsymbol{n} \tag{2.6}$$

In our implementation, the full solution $p(\boldsymbol{x}, t)$ was obtained for any point in time by integrating these equations with a second-order Runge-Kutta scheme, as in [20].

2.3 Data

For this project we received access to the "WaveSims" codebase, authored by Laurent White of AMD. "WaveSims" implements the DG method described above to solve the wave equation for a variety of grid sizes and input parameters. After receiving this code, we first altered it to be able to generate data in parallel on multiple CPUs. The initial version of the code generated all of the data for different sources in sequence, but we were able to edit it to be able to utilize AMD's multiple CPUs. This sped up the data generation process and allowed us to compile data from a variety of source locations more quickly. We used this to create and collect the training data for our network.

For the first iteration, we ran a simulation on the spatial domain $[0,1] \times [0,1]$ for $t \in [0,2]$. Initial conditions were set to be 0 and a source-wave was generated by the Ricker wavelet centered at the point $(x_s, y_s) = (0.4, 0.2)$ with the parameters $t_0 = 0.2, \tau = 5000, \omega = 5$. We collected pressure p, x-velocity u, and y-velocity v at each point on a 32×32 grid of points every 0.012 seconds. This gave us $n = 3344841$ data points for which we had the inputs (t, x, y) and outputs (p, u, v) for our network.

In subsequent experiments we repeated the some process to obtain data for several different source points, and over grids of different sizes.

2.4 Neural Networks

First introduced by McCulloch and Pitts in 1943, neural networks (NN) are a computing technique to resemble the human brain for problem solving [4]. The human brain has computing units called "neurons" which work in parallel and exchange information using their connections. Artificial neural networks work in a similar way, consisting of computing units where each unit is connected by weights. These simple units calculate the sum of weights from inputs and by using an activation function, find out the output [25].

In artificial neural networks the units, neurons, are arranged in layers. Neurons in the same layer behave in the same manner and they usually have the same activation function. These units can be fully interconnected or not connected. A neural network architecture is the arrangement of these neurons in their layers, and the patterns of connection between and within layers [22]. To produce an output, the neural network takes input data and applies multiple compositions of affine and non-linear functions to it. On the figure below, an example of a five layer neural network is given Figure 2.1.

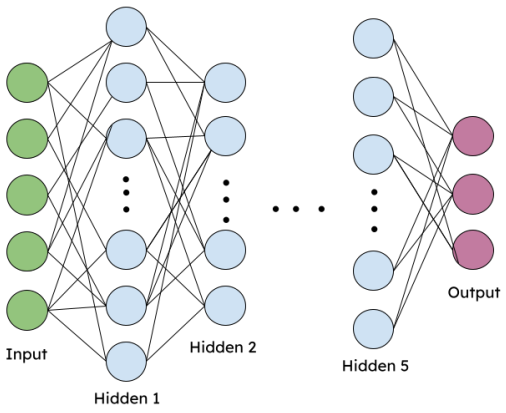


Figure 2.1: Representation of a neural network architecture with five hidden layers

To give an example, a two layer neural network can be expressed in the form:

$$N(x, \theta) = \sigma(w_2\sigma(w_1 x + b_1) + b_2) \tag{2.7}$$

where w_i are the weight matrices, b_i are the bias vectors, and σ is a nonlinear activation function.

Examples for nonlinear activation functions can be given as [16]:

- **Sigmoid Function**
  It is a nonlinear activation function mostly used in feed-forward neural networks. This function is defined for real input values and it is a bounded differentiable function. Sigmoid function's formulation can be given as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

- **Hyperbolic Tangent Function (tanh)**
  Hyperbolic tangent function is another type of activation function known as the *tanh* function. It is a smooth function which range lies between -1 to 1. For multi-layered neural networks it gives better training performance compared to the *sigmoid* function. It's formulation can be shown as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.9}$$

- **Rectified Linear Unit (ReLU) Function**
  ReLU is the most widely used activation function for deep learning applications. It is a faster activation function option compared to *sigmoid* and *tanh* and in most cases it gives a better performance. Using the gradient-descent method, it provides an easy optimization on linear models. ReLU performs a treshold operation to inputs and if the values are less than zero, it sets these values to zero. Thus, it is represented as:

$$f(x) = \max(x, 0) = \begin{cases} x_i, & x_i \geq 0 \\ 0, & x_i < 0 \end{cases} \tag{2.10}$$

- **Softmax Function**
  The Softmax function is used to compute the probability distribution of a vector of real numbers. It gives an output which is between 0 and 1 where the sum of the probabilities are equal to 1.

$$f(x) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.11}$$

For our baseline neural network we are using the *tanh* activation function for each layer, except the last layer which has a linear activation function.

2.5 Physics Informed Neural Networks

A Physics Informed Neural Network (PINN) is a deep learning framework that integrates knowledge about the underlying physics of the data, either in its architecture or loss function, see for example ([17], [19], [18]). The motivational background in this algorithm is that prior knowledge or constraints can provide accurate and physically consistent data even in the presence of data imperfections such as outliers and noisy values or absence of data.

2.6 CPU and GPU

Central Processing Units (CPUs) and Graphics Processing Units (GPUs) are essential components of modern computer systems. They are processors designed to perform computational tasks. CPUs are considered the "brain" of a computer and they coordinate and execute instructions for all other hardware components. Through a process known as multi threading, a CPU can handle multiple tasks simultaneously with its multiple cores. On the other hand, GPUs were initially developed to accelerate graphics for gaming and visual purposes. However, their capabilities with parallel processing have led to their usage in various fields such as machine learning and scientific research.

In the recent years, trends in GPU design have provided them larger memory, increased parallelism and ever-increasing degrees of programmability. With these features, GPUs have gained enough flexibility to be used in non-graphic applications [23].

The data parallel architecture of GPUs delivers notable performance gains for computationally intense applications. In performance analysis tests for deep learning applications, it was shown that GPUs complete tests faster compared to CPUs and the increase in the number of cores and the core operating frequency seems to shorten the running time [2].

In our project we were granted access to an HPC (High Performance Computing) system with multiple GPU accelerators per node and high-speed interconnect. These resources allowed us to evaluate various run time performances and provided an opportunity to explore performance enhancements through GPU accelerators.

Using application accelerators such as GPUs has been among the latest trends in the HPC community, as they are a cost-effective option. GPU accelerators significantly reduce space, power, cooling demands, and the number of operating system images that must be managed relative to CPU clusters with similar computational capabilities [7].

Chapter 3

Computational Benchmarking

This Chapter describes our methods for benchmarking and improving execution times for the "WaveSims" codebase. Such improvements allowed us to generate more data throughout the project, which sped up the model iteration process. We first considered execution times for the DG method run on finer grids; and secondly, we implemented multiple simulations in parallel to speed up data generation.

3.1 Grid Length in the DG Method

In the following, we consider the *grid length*, n, where the PDE was simulated with DG method on an $n \times n$ grid. This was of interest as we wished to obtain data at as many points as possible, which necessitates finer grids. The goal was to obtain ground truth data on as fine a grid as practically possible.

We also had to ensure that simulations remained numerically stable, a necessary condition for which is that the Courant–Friedrichs–Lewy (CFL) number remains bounded. In this context, this condition is equivalent to keeping $n\Delta t = c$ constant for every simulation. Thus, the timestep, Δt, in each simulation must be kept inversely proportional to n, as is shown in Figure 3.1.

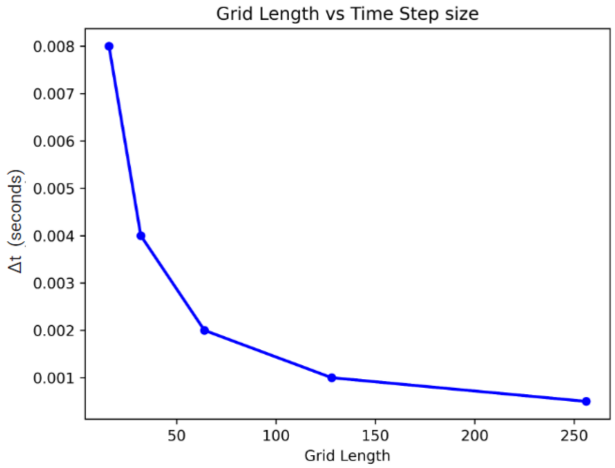


Figure 3.1: The length of grid, n, and the timestep size, Δt, required to keep the CFL number constant

Figure 3.2 shows execution time increasing with increases in grid length. Consider the

total number of integrals that need to be computed, given by $3 \times n^2 \lceil \frac{2}{\Delta t} \rceil$ - as 3 integrals need to be calculated for every grid point at each time step. Figure 3.3 shows that execution time is linear in the total number of integrals.

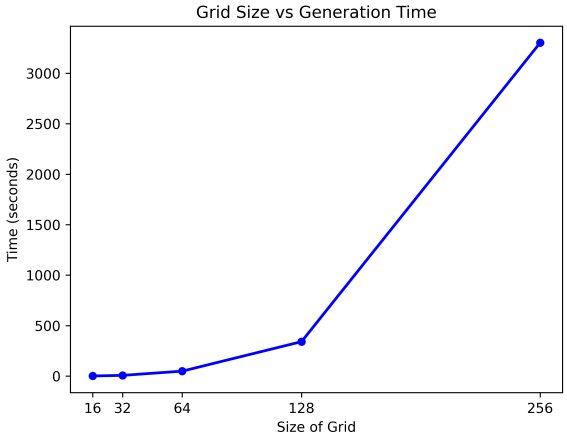


Figure 3.2: The length of grid, n, against the time taken to run one simulation

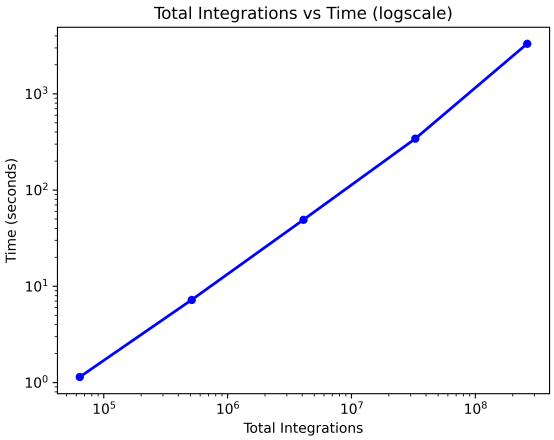


Figure 3.3: The total number of integrals against the time taken to run one simulation. Both axis are log-scale.

3.2 Parallelization

As we needed to generate data from multiple sources, with different grid lengths, we were interested in speeding up the time to simulate this data. There were three possible directions in which one could consider parallelizing the code.

As each integral is formulated as a dot product, in-built *numpy* linear algebra parallelization schemes had the potential to improve performance. However, we found that this was ineffective due to the number of overheads introduced. Forcing a simulation to run on only one thread was found to be optimal.

One other possibility was splitting the domain into smaller sub-meshes, and computing integrals in parallel. This approach, whilst interesting, was deemed outside the scope of this project. Attention was instead focused on parallelizing entire simulations separately.

The existing "WaveSims" codebase offered a functionality to run multiple simulations in one script. However, this implementation ran each simulation serially, one after the other. When working with multi-core CPU architectures, significant improvements to run time can be acheived by running simulations on parallel threads. Theoretically, the time taken to run any number of simulations can be cut in half when running on two cores, as each simulation is completely independent of any other simulation.

For a given number m of simulations, and a parallel algorithm running on k nodes, consider the speed up factor SF. This is defined as the time taken to run on one node divided by the time taken to run on k nodes. As we run each simulation independently, then, theoretically we have

$$SF_{\text{theory}}(k) = \frac{m}{\lceil k/m \rceil} = k \quad \text{if } k|m \tag{3.1}$$

We implemented parallel simulations with the python package *mpi4py*. This was benchmarked by running simulations for 32 different source points, each on a 128×128 grid for $t \in [0, 2]$. For $k \in \{1, 2, 4, 8, 16, 32\}$, we obtained the empirical speed up factor $SF_{\text{emp}}(k)$. At the fullest possible parallelization, $k = 32$, we had $\frac{SF_{\text{emp}}(32)}{SF_{\text{theory}}(32)} = 0.73$. Thus, we had achieved 73% speed up efficiency in this implementation.

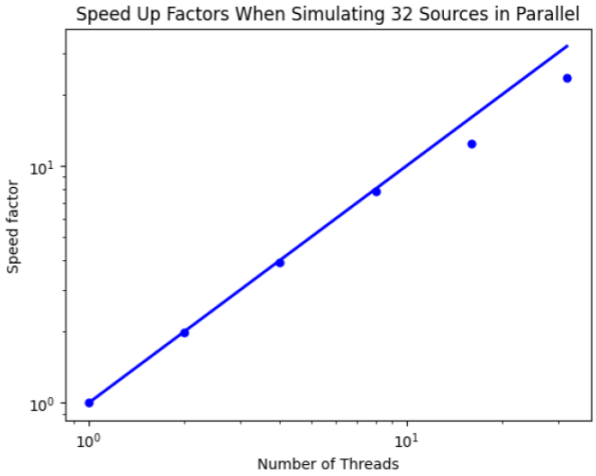


Figure 3.4: $SF_{\text{emp}}(k)$ for $k \in \{1, 2, 4, 8, 26, 32\}$.
The solid line shows $SF_{\text{theory}}(k)$. Both axis are log scale.

3.3 Central Differences and Auto-Differentiation Methods

Since we implemented our networks in Python with PyTorch, we initially used the autograd system to compute the gradients in our PDE loss terms. Autograd is a reverse automatic differentiation system. Automatic differentiation works by taking a computational graph and calculating the gradients of the inputs. In order to do reverse automatic differentiation, autograd constructs a graph of all operations that created the data as operations are executed. This produces a directed acyclic graph from the output tensors (roots) to the input tensors (leaves). In order to compute gradients using the chain rule, autograd then traverses this graph from roots to leaves. At every iteration, this graph is recreated from

scratch. While this allows for straightforward computations through traversing the graph, this method can become costly. As the complexity of operations increases, the number of steps necessary to compute gradients also increases. Storing and traversing the computational graph can become costly in both memory and time, so we looked for ways to reduce these costs.

One method we decided to try instead is the central difference method. Since we were computing the derivatives in the PDE loss function only on the grid of collocation points, we had a cartesian grid of data. This allowed us to use a finite difference method effectively. In our PDE loss functions, we needed to compute the derivative of pressure with respect to x, y and t, the derivative of x-velocity, u, with respect to x and t, and the derivative of y-velocity, v, with respect to y and t.

In order to compute the derivatives of P, u, and v with respect to time, we had to choose a Δt, and perform the following calculations where f would be our network:

$$[P_{pdt}, u_{pdt}, v_{pdt}] = f([t + 0.5\Delta t, x, y])$$
$$[P_{mdt}, u_{mdt}, v_{mdt}] = f([t - 0.5\Delta t, x, y])$$

Then we would be able to calculate $\frac{dP}{dt} = \frac{P_{pdt} - P_{mdt}}{\Delta t}$. We performed similar calculations for the derivatives of u and v with respect to time, as well as P, u, and v with respect to x and y using Δx and Δy.

These calculations introduced new parameters into our networks, $\Delta x, \Delta y$, and Δt. Ultimately we settled on 0.00001 for all three. To compare the speed of using each method, we trained a network for 100 epochs using autograd and another using central difference. The autograd network took 1.73 seconds per epoch, while the central difference network took 0.24 seconds per epoch.

Chapter 4

Data Only Neural Networks

4.1 Architecture

For our Data Only Neural Network (Data NN), we used a residual network architecture that consists of five hidden layers, each with 100 nodes. There is a residual connection between the first and forth hidden layers, and between the second and fifth hidden layers. Extensive numerical experimentation suggested that this network would be expressive enough to fit the PDE. Figure 4.1 shows graphically our network architecture.

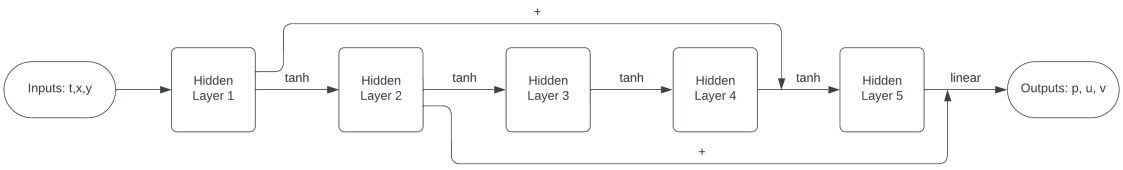


Figure 4.1: A graphical representation of the residual network architecture

Data NNs were trained only on mean-squared error as the loss function over data points. This training data consists of three inputs, t, x, y, and three outputs, p, u, v, which were obtained via the DG based simulation as described in Section 2.3.

We trained three different Data NNs for our purpose – one using data obtained on a 16×16 grid, the second on a 32×32 grid and lastly on a 64×64 grid. For interpolation models, we used 80% of the data points available which were approximately 6240 and 44430 points for the respective grid sizes. For extrapolation models, we used the first 50% of the time points on the same grids, resulting in about 3750 and 27770 points. We used a learning rate scheduler for both models starting with a learning rate of 10^{-3}, decreasing the learning rate by a multiplicative factor of 0.1 every 3000 epochs. The networks trained on the above sparse grids were tested on a fine 128×128 grid. It should also be noted that the labelled training data for this method was obtained from the Discontinuous Galerkin method, as outlined in Chapter 2 which is computationally inefficient. Ideally, when we added the PDE loss to our neural network, we would have liked to keep the amount of labeled data to a minimum.

4.2 Data Only Neural Network Results

The Data NN was trained on data points taken from 16×16, 32×32 and 64×64 grids

on the entire space and time domain. We used the DG method as outlined in Chapter 2 to generate the labelled training data for this network. After training the model with data created on 16×16 grid for 168500 epochs, we obtained the plots comparing the ground truth and prediction of the network for pressure, x and y velocity which can be seen on Figure 4.2.

For the network trained on data with 32×32 grid for 106000 epochs we have the following results on Figure 4.3.

On Figure 4.4 we have the results for the network trained on data with 64×64 grid for 58500 epochs.

As expected, the network performs a lot better in the first half of the time domain since we only provided it training data there. The hope is that the network should extrapolate to the rest of the time domain once we add the PDE loss but ultimately when we incorporate the PDE loss into the network we would keep this data to a minimum. This is because it is computationally expensive to generate this data and one of the goals of this project is to find less expensive methods for solving the wave equation.

We also created heat map animations for our trained models as well as the ground truth to have a better visual understanding in our results. On Figure 4.5, a comparison between the prediction from trained model on 16×16 and the ground truth can be seen. Figures represents the state of pressure on a defined time, t=0.5. The norm of differences between the ground truth and network prediction is also shown as heat map. The network tries to predict the pressure but both the state of the pressure and the values are not so accurate.

Figure 4.6 shows the state when t=1.5 where the network failed to capture the reflective movement which starts on the top of the shown area. As the network was trained with data created on a courser grid like 16×16, high error in the prediction was expected. However, failure to capture the reflection had to be investigated in order to be associated with this fact. To investigate, heat map animations for trained model on data with 32×32 grid were compared.

The comparison between the ground truth and the prediction from the trained model on 32×32 at t=0.5 can be seen on Figure 4.7. It is obvious that this model provided more accurate results than the model on 16×16. To see whether this model captured the reflection or not, the heat map representation at t=1.5 was generated.

Figure 4.8 a comparison on the ground truth and trained model for 32×32 grid at t=1.5 can be seen. It is clear that this model was able to reflect the movement. Hence, having a denser grid size for data to obtain the reflection is the right argument.

On Figure 4.10 comparisons on the ground truth and trained model for 64×64 grid at t=0.5 and t=1.5 is given for further investigation. The expectation was that the model trained on 64×64 should be able to capture the reflection at t=1.5. However, it failed to do so. We suspect the model needs more training even though it provides better numerical results compared to our other models.

To check whether our arguments applies for the times after t=1.5 or not, heat maps for all trained models are created at t=1.92. On Figure 4.11 the heat map comparison between the ground truth and the model trained on 16×16 for the defined time is given. While the model tries to provide nonzero predictions it still can not capture the reflective movement.

Considering Figure 4.12, we can see that this is not the case for the trained model on 32×32 as expected. The model is able to reflect the movement and these representations aligns with the results given on Figure 4.6 and Figure 4.8.

On Figure 4.13, we can see that model trained on 64×64 was still unable to capture the reflection since it needs more training.

The trade off between accuracy and time for making predictions using different grid

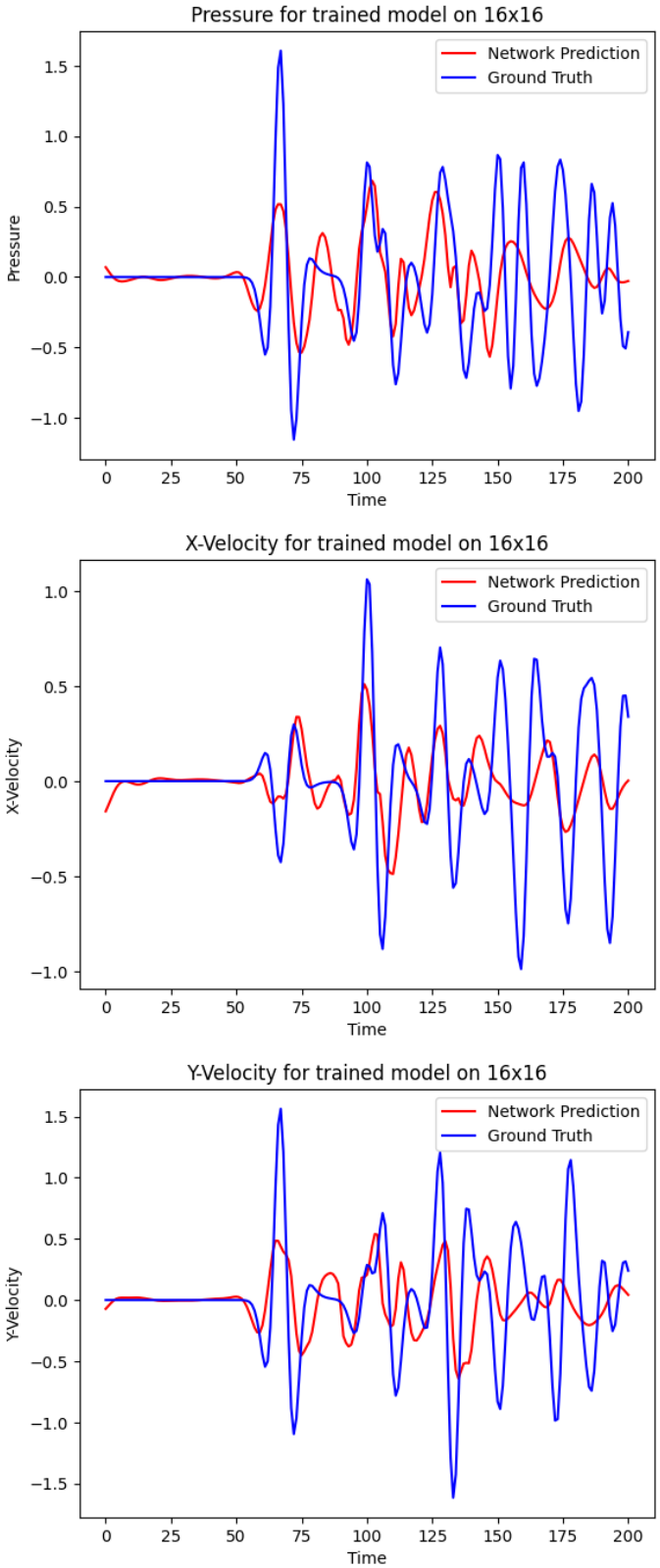


Figure 4.2: Results for pressure x and y-velocity for trained model on 16×16 grid

sizes for training can be seen in table 4.2. Model execution times do not change significantly

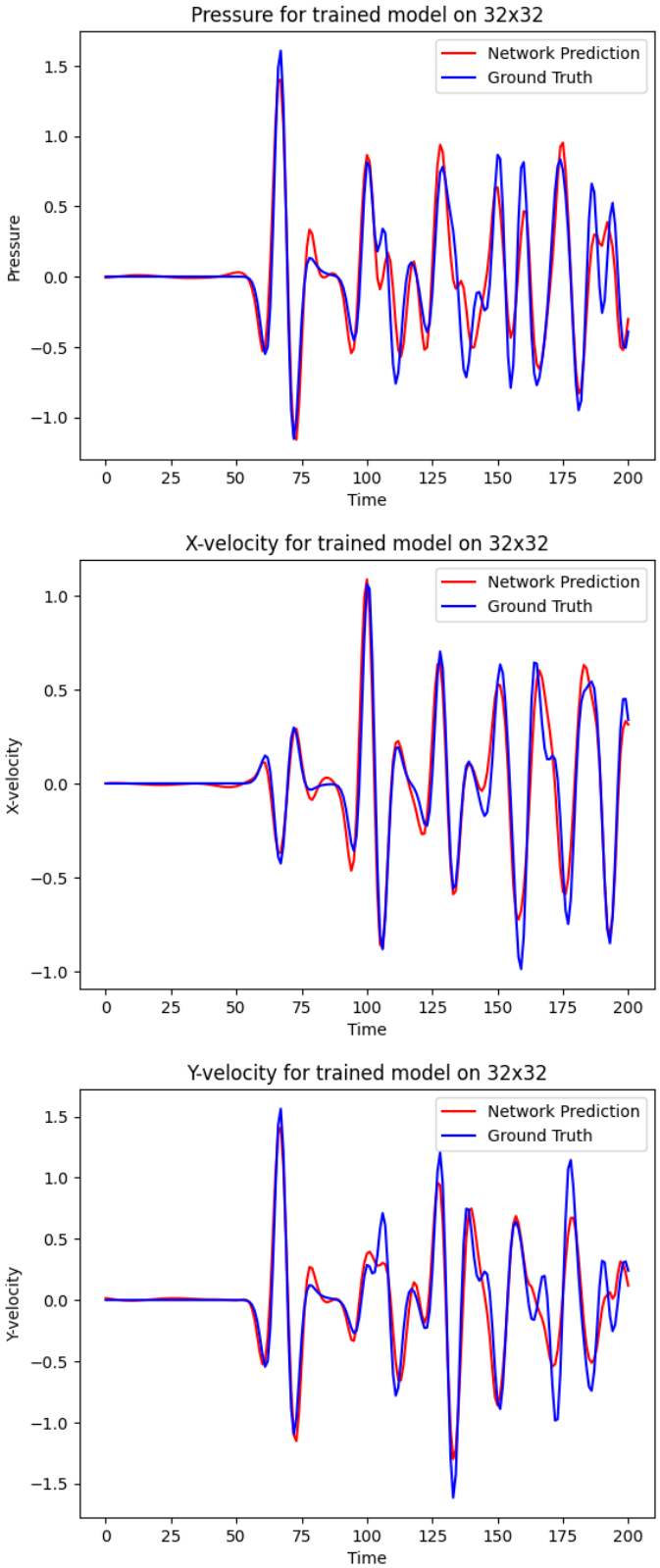


Figure 4.3: Results for pressure x and y-velocity for trained model on 32×32 grid

between the trained models, and it should be noted that these times represent orders of

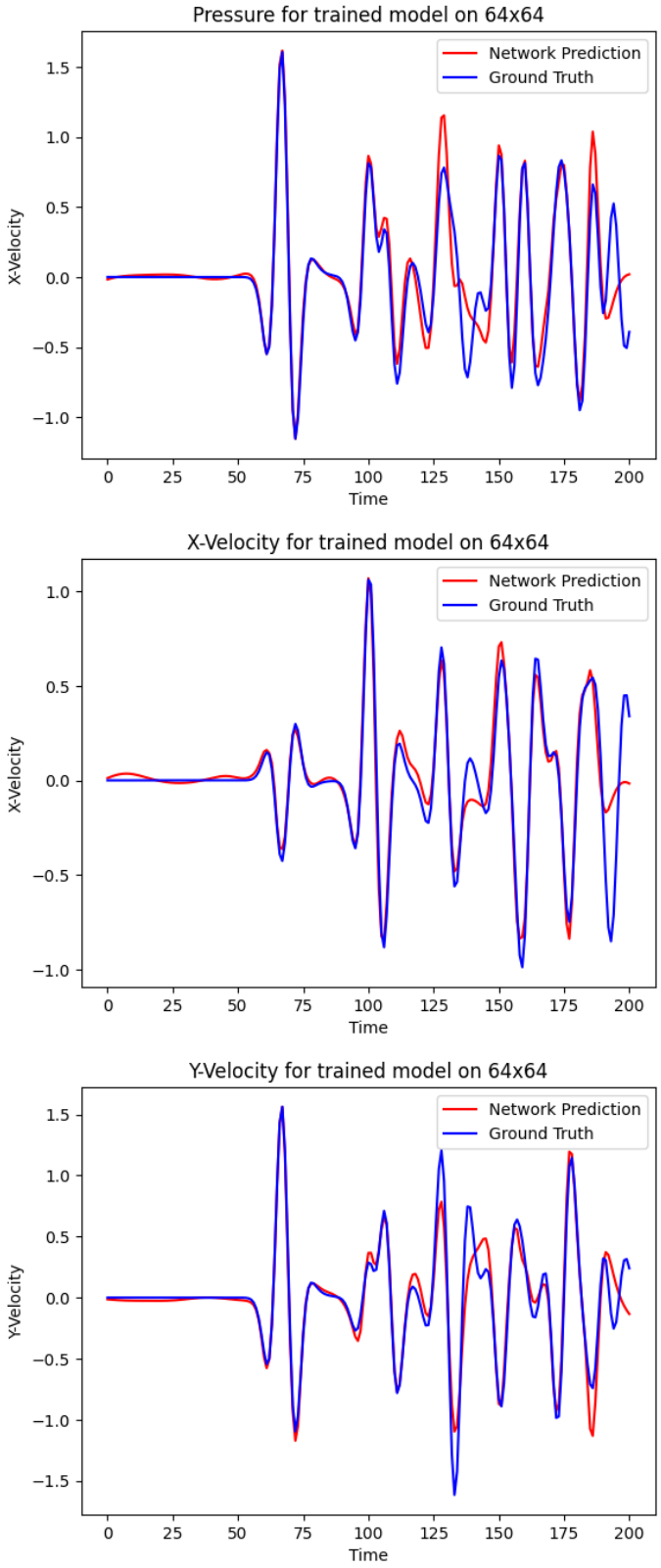


Figure 4.4: Results for pressure x and y-velocity for trained model on 64×64 grid

magnitude improvement compared to the time taken to generate data on a 128×128 grid

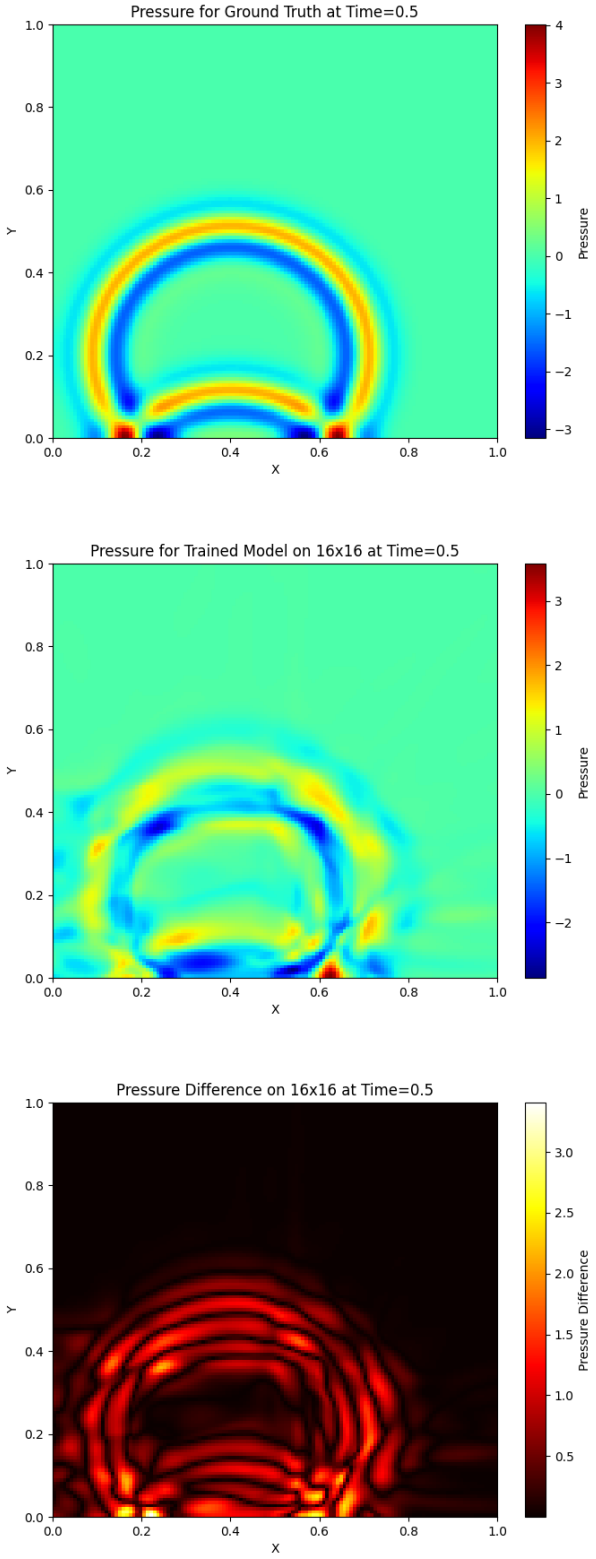


Figure 4.5: Heat map representations of the pressure for ground truth, trained model on 16×16 grid and the difference at t=0.5

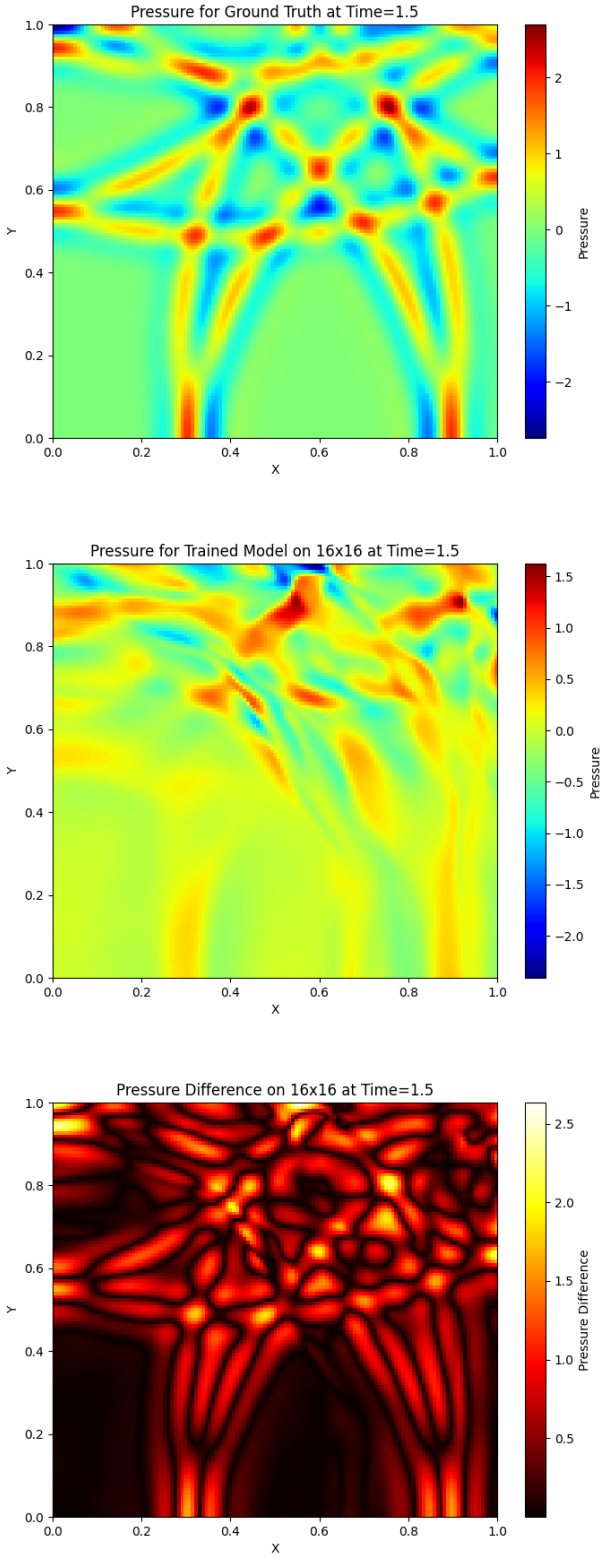


Figure 4.6: Heat map representations of the pressure for ground truth, trained model on 16×16 grid, and the difference at t=1.5

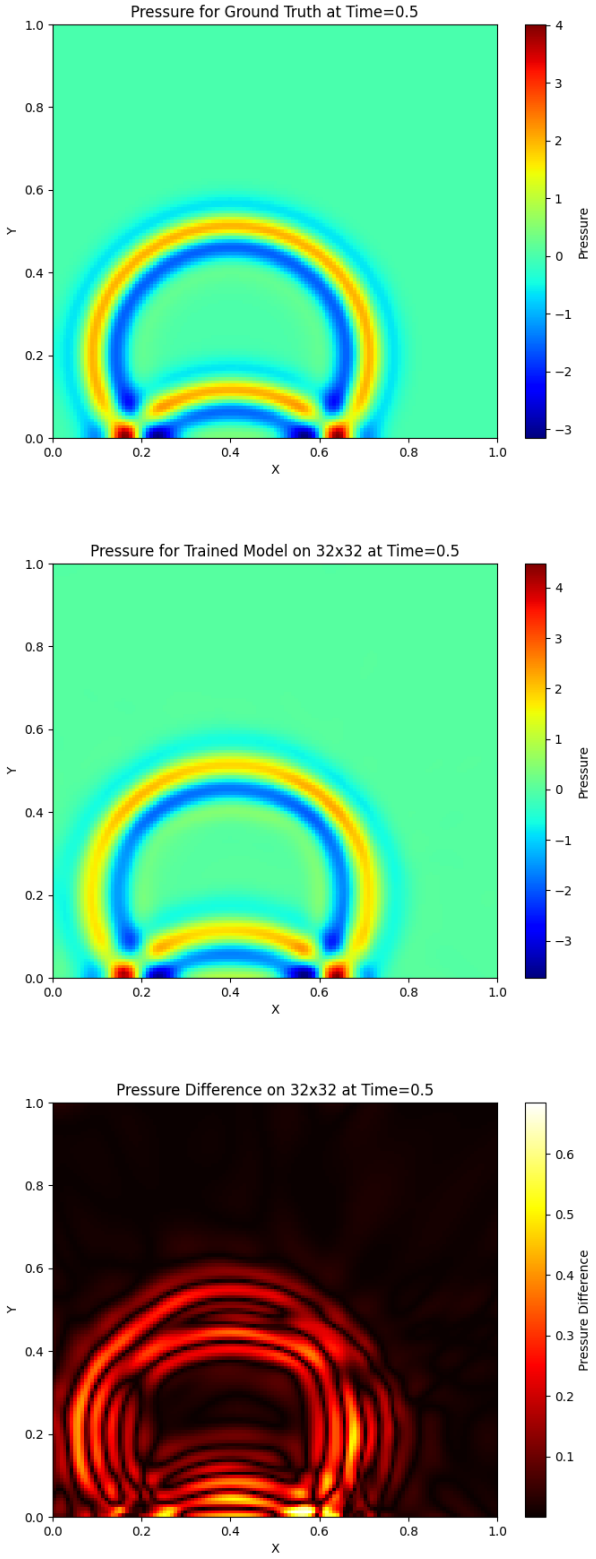


Figure 4.7: Heat map representations of the pressure for ground truth, trained model on 32×32 grid and the difference at t=0.5

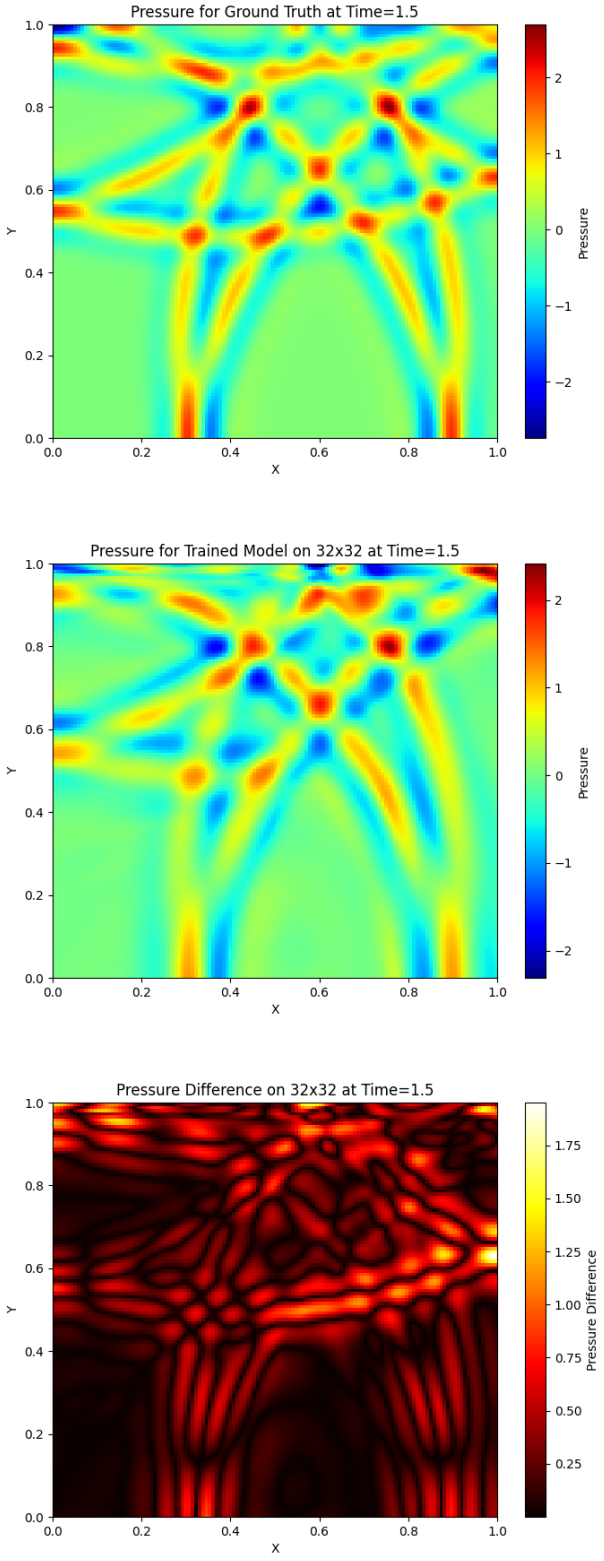


Figure 4.8: Heat map representations of the pressure for ground truth, trained model on 32×32 grid and the difference at t=1.5

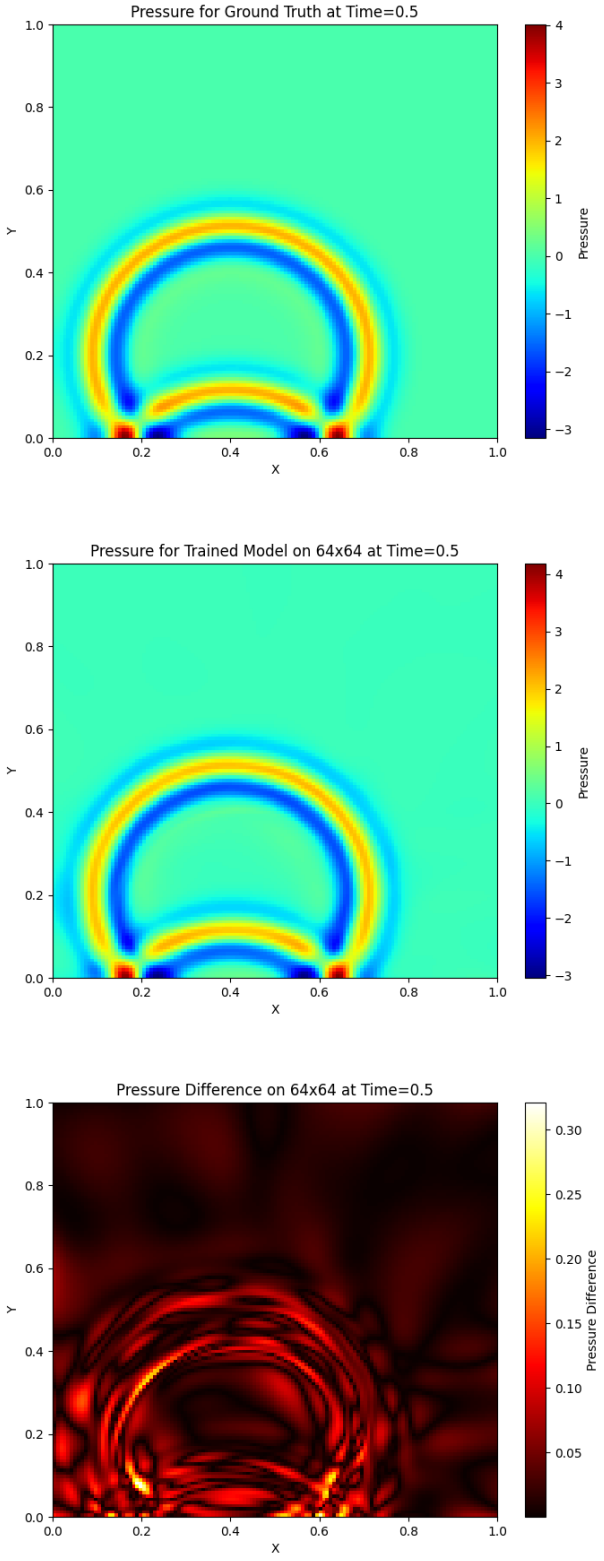


Figure 4.9: Heat map representations of the pressure for ground truth, trained model on 64×64 grid and the difference at t=0.5

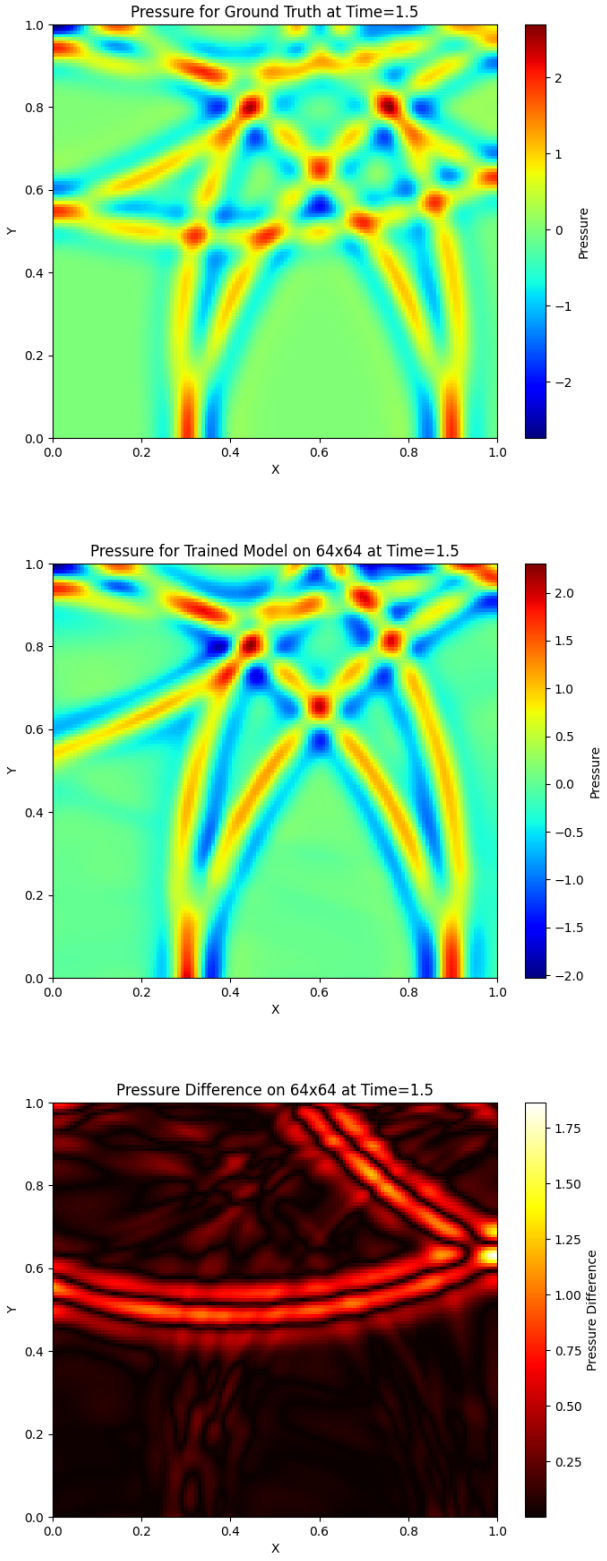


Figure 4.10: Heat map representations of the pressure for ground truth, trained model on 64×64 grid and the difference at t=1.5

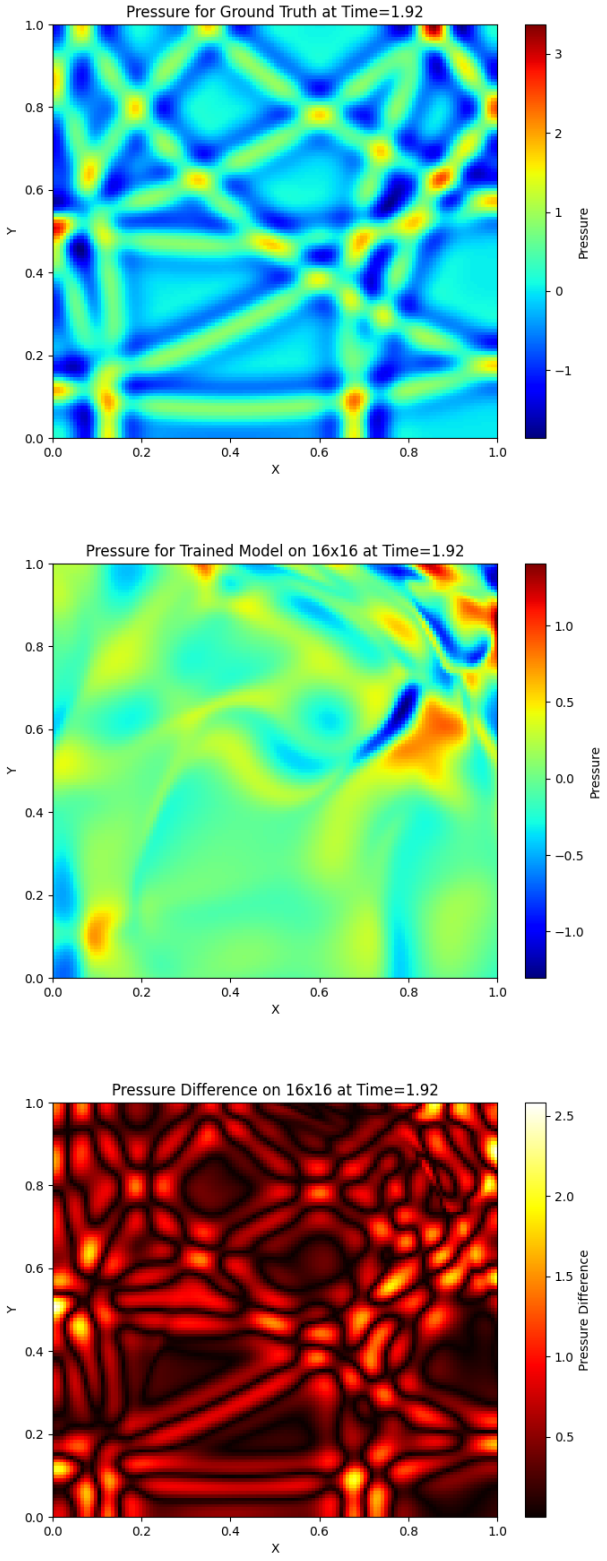


Figure 4.11: Heat map representations of the pressure for ground truth, trained model on 16×16 grid and the difference at t=1.92

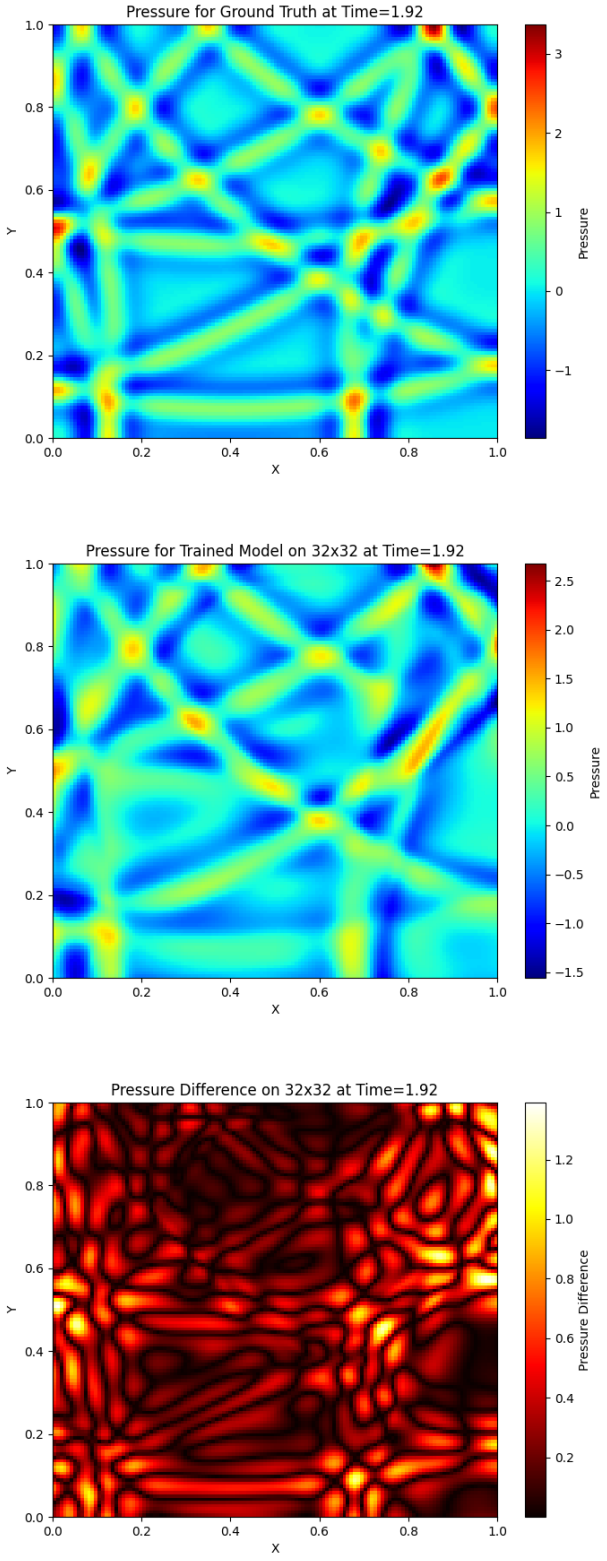


Figure 4.12: Heat map representations of the pressure for ground truth, trained model on 32×32 grid and the difference at t=1.92

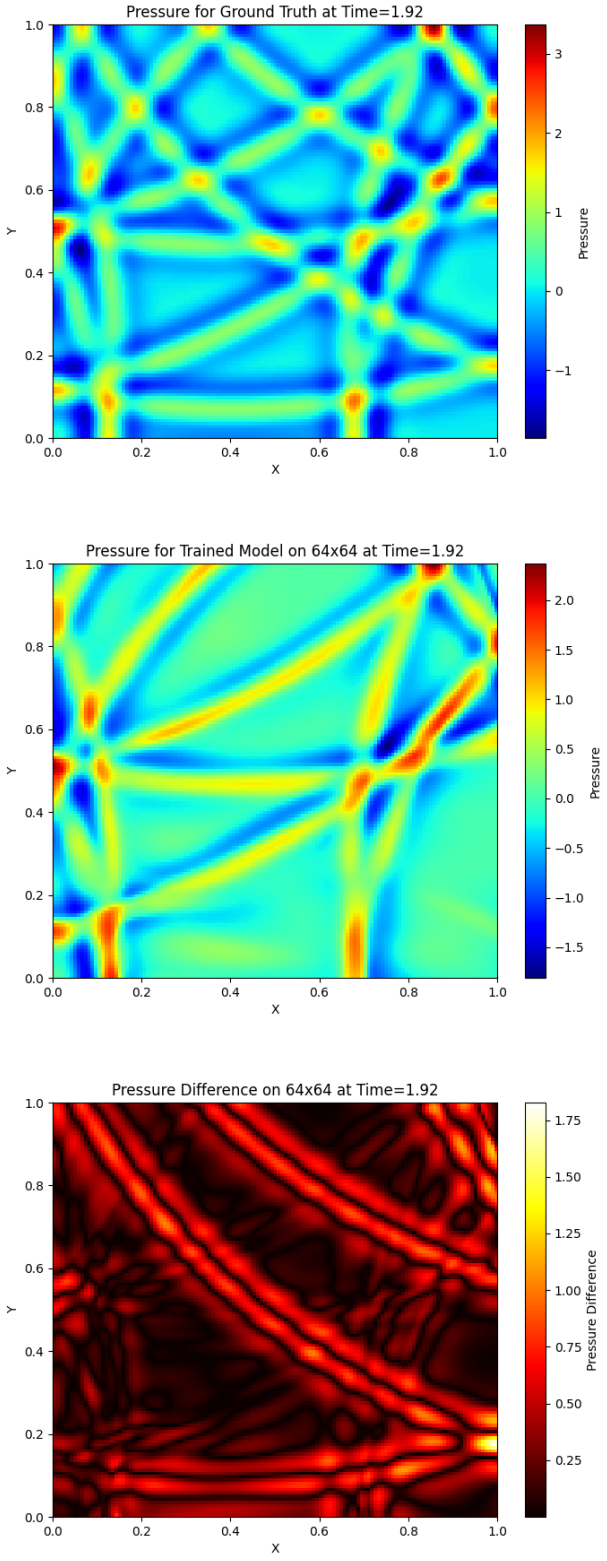


Figure 4.13: Heat map representations of the pressure for ground truth, trained model on 64×64 grid and the difference at t=1.92

	Model on 16×16 grid	Model on 32×32 grid	Model on 64×64 grid
Difference at t=0.5	56.5	13.9	5.1
Difference at t=1.5	73.5	41.9	35.8
Difference at t=1.92	75.0	43.4	45.05

Table 4.1: A table comparison of the Frobenius norm of differences between model predictions and ground truth for respective times.

using DG methods ($\simeq 350s$). On the other hand, relative errors improve notably when moving from 16×16 to 32×32 and to 64×64 training data.

The time spent training, while large, is not of concern because it is a one-time expense and once the model is trained it can be evaluated on any grid very fast. Thus, this cost can amortised effectively.

	16x16 grid on 128x128	32x32 grid on 128x128	64x64 grid on 128x128
Model Execution Time	<1 s	<1 s	<1 s
Relative Error	82.9%	27.4%	15.3%
Training Time	6 hours	6 hours	15 hours

Table 4.2: A table comparison of accuracy and time trade off between trained models

After creating the Data NN and obtaining results from our trained models, we started to work on PINNs in order to predict faster and more accurately compared to our data only network.

Chapter 5

Physics Informed Neural Networks

5.1 Overview

Physics Informed Neural Networks (PINNs) have the ability to incorporate physical constraints into vanilla Neural Networks, greatly enhancing their ability to model PDEs. The main idea behind PINNs is to train the network on a set of data points while also constraining the network to obey the PDEs they are supposed to model. Since the network is already supposed to be modelling a continuous function that fits the data points, the PDE constraints can easily be enforced by computing the necessary derivatives at the required points, which are referred to as collocation points. As a result, this helps the network reduce its dependency on labelled data, which is computationally expensive to generate. The easiest way to make this modification is by adding an extra term to the loss function, as described in the following section.

In order to investigate the performance of PINNs on modelling the 2D acoustic wave equation, we broadly trained two different models – extrapolation models and interpolation models. For the first category, we provided the network with data over the entire space and time domain. For the second, we provided the network with data over the first half of the time domain and the entire space domain. We found that for our problem, PINNs were not effective at improving upon the data only models.

5.2 Physics Loss and Collocation Points

For computing the data loss, we used a standard mean squared error, as shown in Equation (5.1), to compare the predicted pressure and predicted velocity to their respective ground truths.

$$\frac{1}{n}\sum_{i=1}^{n} ||\tilde{p} - p||_2^2 + ||\tilde{u} - u||_2^2 + ||\tilde{v} - v||_2^2 \tag{5.1}$$

When implementing the physics loss function, we considered the first order form of the wave equation as described in Chapter 1, while also accounting for the reflective boundary conditions and zero initial conditions. These conditions together essentially informed our network to obey some physical laws (in our case the PDE) while predicting the outputs. For the pressure and velocity comparisons, we used the L_1 error norm because that has shown faster convergence in the case of PINNs [13]. Together, the overall loss function is summarized below:

$$L_{overall} = w_1 \cdot L_{data} + w_2 \cdot L_{physics} \tag{5.2}$$

$$L_{physics} = L_{DO} + L_{BC} + L_{IC} \tag{5.3}$$

$$L_{DO} = ||\partial_t \tilde{p} + \partial_x \tilde{u} + \partial_t \tilde{v} - f_s(\boldsymbol{x}, t)||_1 + ||\partial_x \tilde{p} + \partial_t \tilde{u}||_1 + ||\partial_y \tilde{p} + \partial_t \tilde{v}||_1 \tag{5.4}$$

$$L_{BC} = ||\partial_x \tilde{u}||_1 + ||\partial_y \tilde{v}||_1 \tag{5.5}$$

$$L_{IC} = ||\tilde{p}(x, y, 0)||_1 + ||\tilde{u}(x, y, 0)||_1 + ||\tilde{v}(x, y, 0)||_1 \tag{5.6}$$

where L_{DO} is the differential operator loss coming from the 2D Acoustic Wave Equation as described in Chapter 2, L_{IC} is the initial condition loss, and L_{BC} is the boundary condition loss.

The main advantage for the $L_{physics}$ component of the loss is that it should provide our network more information without the availability of labelled data. Thus, these points can be provided to the network on a much finer grid over space and time. In theory, they should help the network improve the results in regions where we already have provided it labelled data, and also approximate solutions in regions where we do not provide the network with labelled data. The figures below demonstrate the way we would theoretically sample points for computing L_{data} and $L_{physics}$.

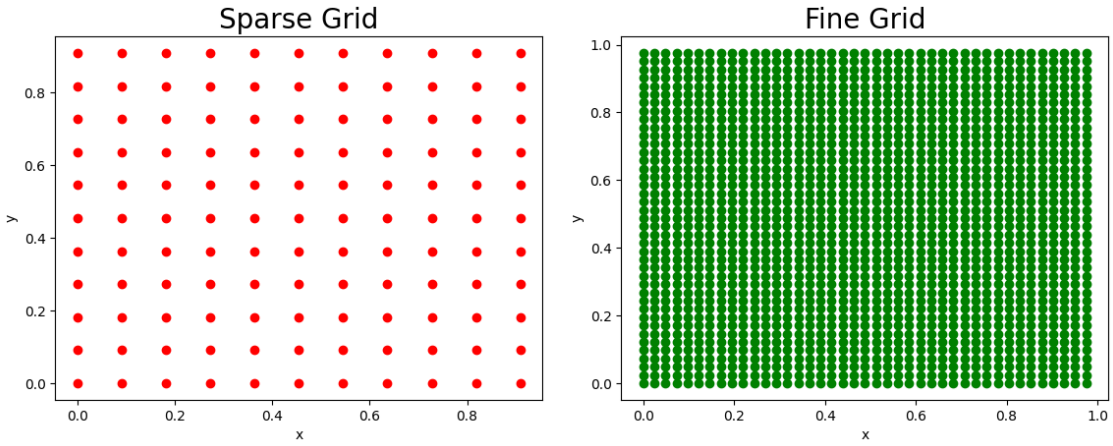


Figure 5.1: Sampling from a finer grid for $L_{physics}$ and a coarse grid for L_{data}

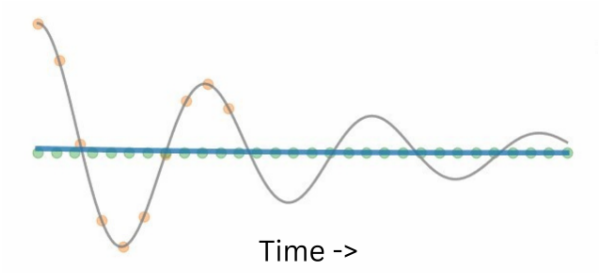


Figure 5.2: Extrapolation models taking labelled data only in the first half of the time domain

5.3 Learning Schemes and Results

5.3.1 Scheme 1 : Data + Physics Loss over the entire domain

To train models that used both physics and data loss, we trained on labelled data points first and incorporated the physics loss in the training process after sufficient training loss convergence. The data points were provided on a coarse 32×32 grid with 51 time steps in our domain, while the collocation points for the physics loss were sampled from a fine 100×100 grid with 1671 time steps in the entire domain. When implementing interpolation, the data points were sampled over the $[0, 2]$ time region. When implementing extrapolation, the data points were sampled over the $[0, 1]$ time region.

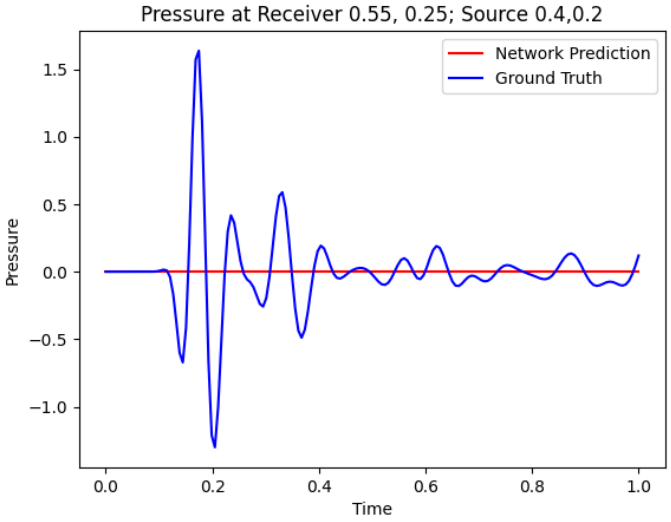


Figure 5.3: Results from both Interpolation and Extrapolation Models

We trained networks with added physics loss for both extrapolation and interpolation and noticed that the network converged to a constant zero prediction over all of space and time, indicating a strong convergence towards this solution. Based on our observations from these results, we investigated the behavior of the physics loss function. Some of the potential issues we identified were – i) traversing the highly irregular loss landscape, ii) a potential numerical discontinuity in the initial time steps due to the forcing term – PINNs tend to struggle to capture such sharp changes [11] iii) a highly localized Ricker wavelet near the source in the forcing term, and iv) complex parameters for modelling the wave (steepness 5000; frequency 5). With these issues in mind, we moved forward to train the network with other schemes in attempts to handle these challenges.

Complex Loss Landscape

The work done in [8] indicates that training PINNs for modelling PDEs with complex parameters can be a challenging task. They show that vanilla PINNs only manage to train well for easy parameter regimes and fail to learn the relevant physics for more challenging and realistic regimes. The paper specifies that the added regularization term involves a differential operator which could be ill-conditioned and non-convex (unlike L_1 or L_2 regularization). They claim the added PDE-based regularization term can make the loss landscape significantly more complex, making optimization a much more difficult task. It should also

be noted that some of the failure modes discussed in [8], such as the 1D diffusion reaction equation, vary over one spatial direction and time. The PDE we are trying to model has two spatial directions, along with time, and we naturally expected it to have a much more complex loss landscape. The authors of the same paper proposed two learning schemes to handle these complex loss regimes, both of which are described below.

5.3.2 Scheme 2 : Curriculum Learning

Curriculum learning as described in [8] has proven to be useful in training PINNs with complex loss landscapes. The idea behind this approach is to start training the network with some set of parameters for the PDE that the network can easily capture, and then iteratively work towards the more complex parameters that we treat as our target constraints.

In our case, those parameters would be the frequency (ω) and steepness (τ) terms in the Ricker wavelet ($f_s(x,t)$) as shown below. A high value of τ makes the wave increasingly localized, and a high value of ω makes it highly oscillatory.

$$f_s(\boldsymbol{x},t) = \frac{\tau}{\pi}\left(1-(2\pi\omega)^2(t-t_0)^2\right)\exp\left(-\frac{1}{2}\left[(2\pi\omega)^2(t-t_0)^2+2\tau||\boldsymbol{x}-\boldsymbol{x}_s||^2\right]\right) \tag{5.7}$$

While implementing curriculum learning for our network, we started by iteratively training the network in the following order – i) Steepness – 5; frequency – 2 ii) Steepness – 100; frequency – 2 iii) Steepness – 1000; frequency – 3 iv) Steepness – 2000; frequency – 4 v) Steepness – 5000; frequency – 5. However, this scheme did not yield promising results for the wave under any of the parameters as seen in Figure 5.4. It should be noted that this scheme was implemented under the constraints of no labelled data.

5.3.3 Scheme 3 : Sequence to Sequence Learning

Sequence to Sequence Learning as described in [8] also seeks to deal with the complexity of the PDE parameters. In sequence to sequence learning, the network is trained on the time steps in an iterative process, as opposed to training on the whole time domain at once. In other words, we would train the network at the first time step Δt before moving on to train on to the second time step 2Δt while using Δt as the initial conditions. When we implemented the above scheme for our model, the results were less than ideal as can be seen in Figure 5.6. The figure below describes the sequence to sequence learning scheme for a problem varying in one spatial dimension over time.

5.3.4 Scheme 4 : Data Loss + Physics Loss over a specific region

Another major issue that we found with the 2D acoustic wave equation is the discontinuity introduced by the forcing term during the initial time steps. PINNs might have difficulties capturing such sharp changes, and such discontinuities might make the loss function ill-conditioned. Thus, we decided to remove collocation points from the first 0.8 seconds of the time domain where the network would already be receiving data points for training. We provided it labelled data during first 1 second of the time domain, and collocation points during the last 1.2 seconds of the time domain. The overlap of 0.2 seconds between both physics and data loss will be used by the physics loss as the initial condition during training.

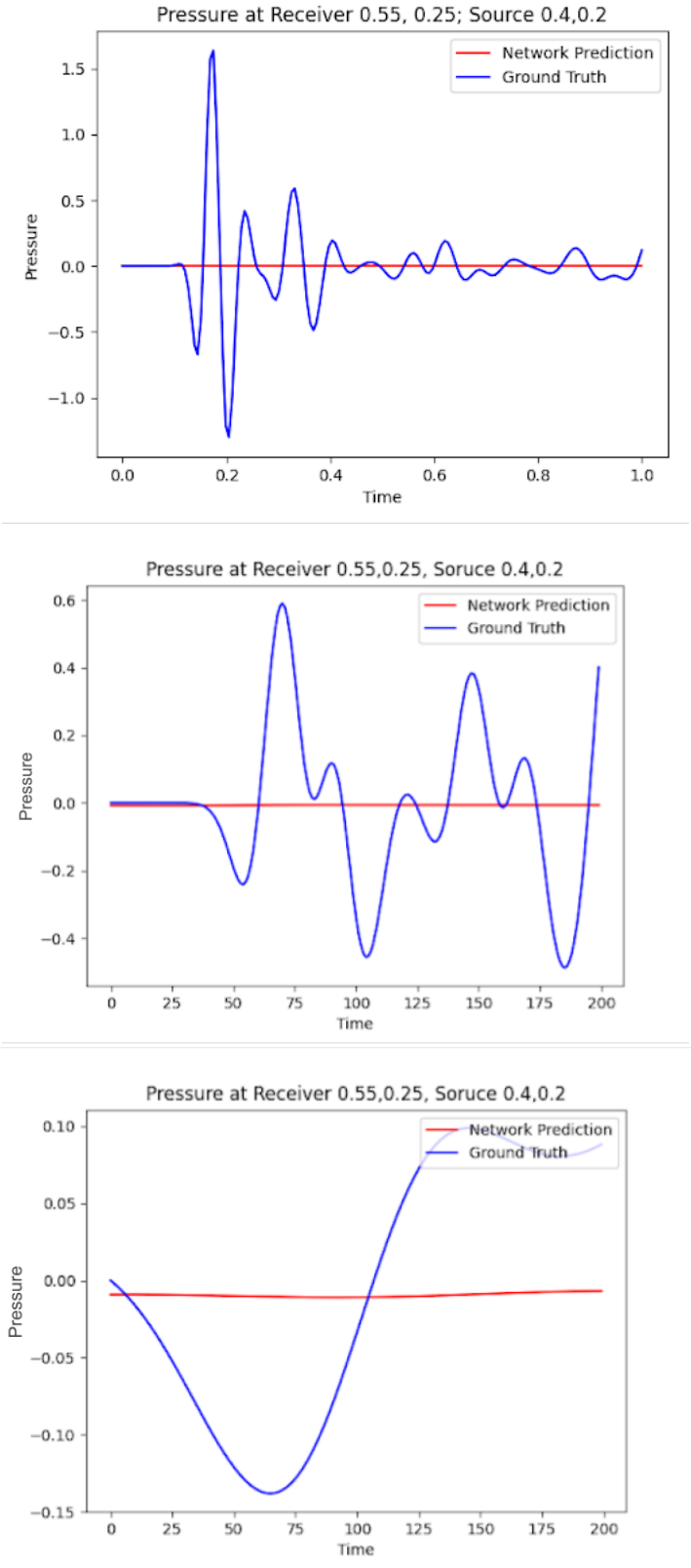


Figure 5.4: Results for Pressure under different parameters – i) Steepness – 5000; frequency – 5; ii) Steepness – 100; frequency – 2; iii) Steepness – 5; frequency – 2

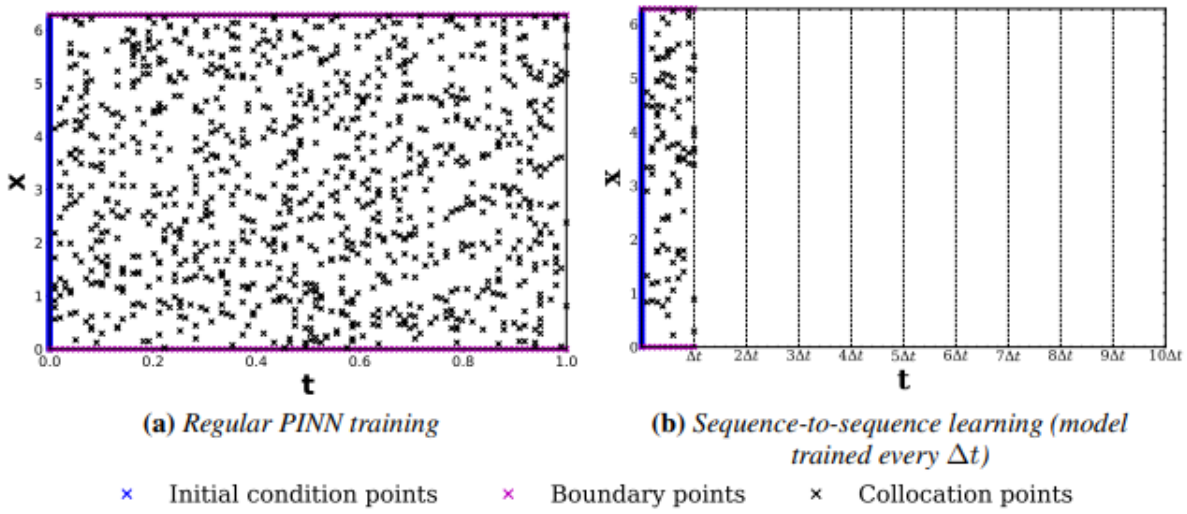


Figure 5.5: Representation of Sequence to Sequence learning from [8]

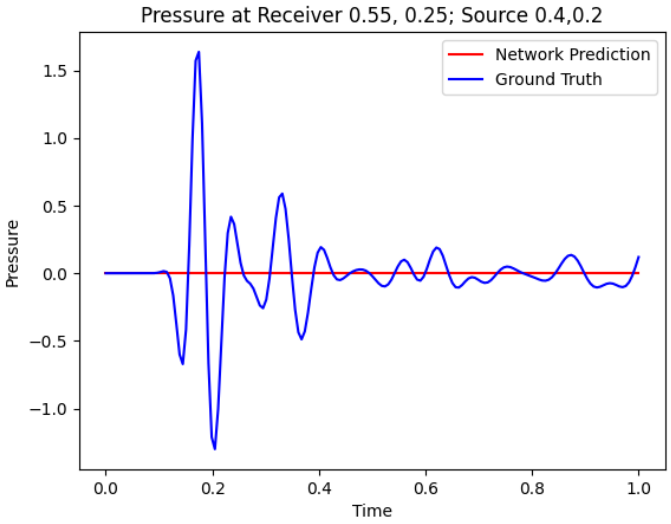


Figure 5.6: Results from Sequence to Sequence Learning

We again notice in Figure 5.8 that the addition of physics loss gives us subpar results in the region of extrapolation. Even the region of overlap seems to be inaccurate upon the addition of the physics loss.

5.4 Conclusion

The schemes we have implemented so far have led us to the conclusion that PINNs might not be appropriate for modelling the 2D acoustic wave equation when considering forcing terms. This is especially the case in the absence of labelled data from the Galerkin scheme. While PINNs have shown promising results for PDEs with simple parameters, the physics loss landscape may be too complex for our PINN to regularize using any of the existing schemes. The experimental results we have gotten so far demonstrate the same.

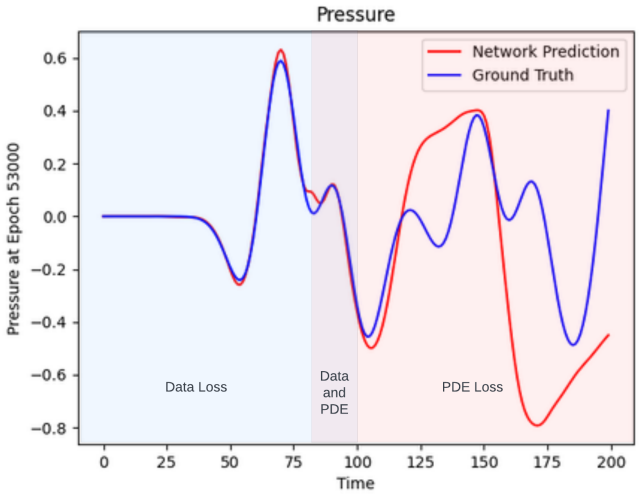
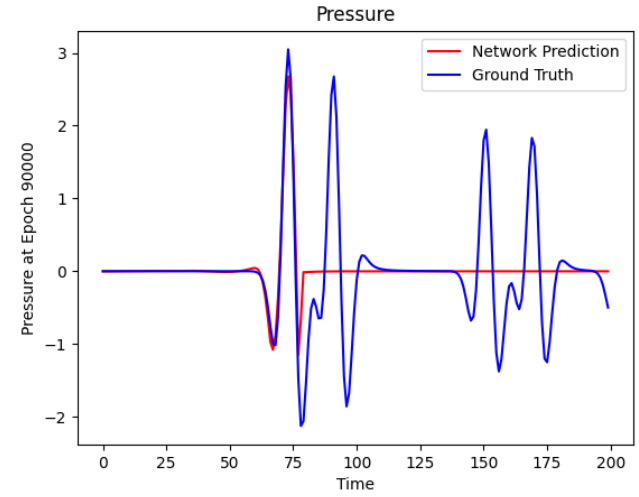


Figure 5.7: Representation of the learning scheme



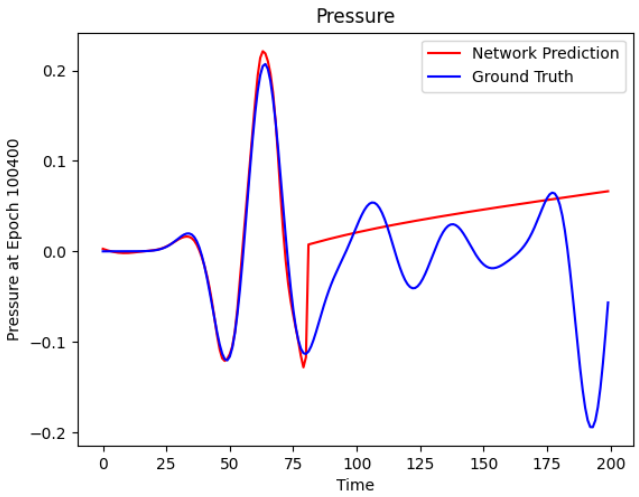


Figure 5.8: Results from this scheme on two sets of parameters

Appendix A

Abbreviations

1D. One Dimensional

2D. Two Dimensional

AI. Artificial Intelligence

AMD. Advanced Micro Devices

CPU. Central Processing Unit

DG. Discontinuous Galerkin

GPU. Graphics Processing Unit

HPC. High Performance Computing

IPAM. Institute for Pure and Applied Mathematics. An institute of the National Science Foundation, located at UCLA.

MCMC. Markov Chain Monte Carlo

ML. Machine Learning

NN. Neural Networks

PDE. Partial Differential Equation

PINN. Physics Informed Neural Network

ReLU. Rectified Linear Unit Function

RIPS. Research in Industrial Projects for Students. A regular summer program at IPAM, in which teams of undergraduate (or fresh graduate) students participate in sponsored team research projects.

tanh. Hyperbolic Tangent Function

UCLA. The University of California at Los Angeles.

Selected Bibliography Including Cited Works

[1] J. Blechschmidt and O. G. Ernst, *Three ways to solve partial differential equations with neural networks — a review*, GAMM-Mitteilungen, 44 (2021).

[2] E. BUBER and B. DIRI, *Performance analysis and cpu vs gpu comparison for deep learning*, in 2018 6th International Conference on Control Engineering Information Technology (CEIT), 2018, pp. 1–6.

[3] B. Cockburn and C.-W. Shu, *Tvb runge-kutta local projection discontinuous galerkin finite element method for conservation laws ii: General framework*, Mathematics of Computation, 52 (1989), pp. 411–435.

[4] F. B. W. S. M. Fitch and W. Pitts., *A logical calculus of the ideas immanent in nervous activity*, Bulletin of mathematical biophysics, 5 (1943), p. 115–133.

[5] K. Gilbert and H. Bass, *Acoustic waves*, in Encyclopedia of Atmospheric Sciences, J. R. Holton, ed., Academic Press, Oxford, 2003, pp. 1–12.

[6] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, *Physics-informed machine learning*, Nature Reviews Physics, 3 (2021), pp. 422 – 440.

[7] V. V. Kindratenko, J. J. Enos, G. Shi, M. T. Showerman, G. W. Arnold, J. E. Stone, J. C. Phillips, and W.-m. Hwu, *Gpu clusters for high-performance computing*, in 2009 IEEE International Conference on Cluster Computing and Workshops, 2009, pp. 1–8.

[8] A. S. Krishnapriyan, A. Gholami, S. Zhe, R. M. Kirby, and M. W. Mahoney, *Characterizing possible failure modes in physics-informed neural networks*, in Neural Information Processing Systems, 2021.

[9] C. L. Wight and J. Zhao, *Solving allen-cahn and cahn-hilliard equations using the adaptive physics informed neural networks*, Communications in Computational Physics, 29 (2021), pp. 930–954.

[10] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature News, (2015).

[11] L. Liu, *Discontinuity computing with physics-informed neural network*. Available at 10.36227/techrxiv.19391279, jun 2022.

[12] M. A. McDonald, M. P. Lamoureux, and G. F. Margrave, *Galerkin methods for numerical solutions of acoustic, elastic and viscoelastic wave equations*, CREWES Research Report, (2012).

[13] B. Moseley, A. Markham, and T. Nissen-Meyer, *Solving the wave equation with physics-informed deep learning*, 2020.

[14] ——, *Solving the wave equation with physics-informed deep learning*. Available at https://doi.org/10.48550/arXiv.2006.11894, 2023.

[15] M. A. Nabian, R. J. Gladstone, and H. Meidani, *Efficient training of physics-informed neural networks via importance sampling*, Computer-Aided Civil and Infrastructure Engineering, 36 (2021), pp. 962–977.

[16] C. Nwankpa, W. L. Ijomah, A. Gachagan, and S. Marshall, *Activation functions: Comparison of trends in practice and research for deep learning*, ArXiv, abs/1811.03378 (2018).

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

[18] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations*, ArXiv, abs/1711.10566 (2017).

[19] ——, *Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations*, Journal of Computational Physics, 378 (2019), pp. 686–707.

[20] F. Renac, M. de la Llave Plata, E. Martin, J. B. Chapelier, and V. Couaillier, *Aghora: A High-Order DG Solver for Turbulent Flow Simulations*, Springer International Publishing, Cham, 2015, pp. 315–335.

[21] N. Ricker, *The form and laws of propagation of seismic wavelets*, Geophysics, 18 (1953), pp. 10–40.

[22] S. Sivanandam and M. Paulraj, *Introduction to artificial neural networks*, Vikas Publishing House, 2009.

[23] D. Steinkraus, I. Buck, and P. Simard, *Using gpus for machine learning algorithms*, in Eighth International Conference on Document Analysis and Recognition (ICDAR'05), 2005, pp. 1115–1120 Vol. 2.

[24] S. Wang, X. Yu, and P. Perdikaris, *When and why pinns fail to train: A neural tangent kernel perspective*, CoRR, abs/2007.14527 (2020).

[25] A. Zayegh and N. A. Bassam, *Neural network principles and applications*, in Digital Systems, V. Asadpour, ed., IntechOpen, Rijeka, 2018, ch. 7.