BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

APPROXIMATION OF SOUND PROPAGATION BY NEURAL NETWORKS

APROXIMACE ŠÍŘENÍ ULTRAZVUKU POMOCÍ NEURONOVÝCH SÍTÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. SON HAI NGUYEN

SUPERVISOR
VEDOUCÍ PRÁCE

Prof. Ing. ADAM HEROUT, Ph.D.

BRNO 2022

Department of Computer Graphics and Multimedia (DCGM) Academic year 2021/2022

Master's Thesis Specification

24521

Student:	**Nguyen Son Hai, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Computer Vision
Title:	**Approximation of Sound Propagation by Neural Networks**
Category:	Artificial Intelligence

Assignment:

1. Study the problematic of approximation of the Helmholtz equation using neural networks.
2. Study the architectures of relevant neural networks, namely graph neural networks.
3. Experiment with available solution according to item 2. Collect and/or acquire a suitable data set for experiments.
4. Iteratively evaluate and improve the solution; collect necessary data.
5. Discuss the achieved results and propose possible future work; create a poster and a short video for presenting the project.

Recommended literature:

- Antonio Stanziola, Simon R. Arridge, Ben T. Cox, Bradley E. Treeby: A Helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound, Journal of Computational Physics, 2021
- Jeramy Watt, Reza Borhani, Aggelos K. Katsaggelos: Machine Learning Refined: Foundations, Algorithms, and Applications, Cambridge: Cambridge University Press, 2020
- Goodfellow, Bengio, Courville: Deep Learning, MIT Press, 2016

Requirements for the semestral defence:

- Items 1 and 2, considerable progress on items 3 and 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Herout Adam, prof. Ing., Ph.D.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2021
Submission deadline:	May 18, 2022
Approval date:	May 3, 2022

Abstract

Neural solvers have been increasingly explored to replace computationally expensive conventional numerical methods for solving PDEs. This work focuses on solving the time-independent Helmholtz equation for the transcranial ultrasound therapy. Using the convolutional neural networks requires the data to be sampled on a regular grid. In order to try to lift this restriction, we propose an iterative solver based on graph neural networks. Unlike Physics-informed neural networks, our model needs to be trained only once, and only a forward pass is required to obtain a new solution given input parameters. The model is trained using supervised learning, where the reference results are computed using the traditional solver k-Wave. Our results show the model's unroll stability despite being trained with only 8 unroll iterations. Despite the model being trained on the data with a single wave source, it can predict wavefields with multiple wave sources in much larger computational domains. Our model can produce a prediction for sub-pixel points with higher accuracy than linear interpolation. Additionally, our solution can predict the wavefield with downsampled Laplacian – only three samples per wavelength. We are unaware of any other existing method capable of working with such a sparse discretization.

Abstrakt

Za účelem nahrazení výpočtově náročných konvenčních numerických metod řešících diferenciální rovnice jsou neurální výpočty stále více prozkoumávány. Tato práce se zaměřuje na řešení časově nezávislé Helmholtzovi rovnice, která modeluje šíření ultrazvuku při transkraniální léčbě ultrazvukem. Při použití konvolučních neuronových sítí musí být data navzorkovaná na pravidelné mřížce, abychom odstranili dané omezení, navrhli jsme neurální výpočet založený na grafových neuronových sítích. Narozdíl od fyzikálně informovaných neuronových sítích (PINN) je potřeba náš model natrénovat pouze jednou, řešení pro množinu nových parametrů vyžaduje pouze dopředných chod. Model byl natrénovaný pomocí učení s učitelem, kde referenční data byly vypočítána pomocí konvenční metody k-Wave. Náš model má stabilní rozvinutí, přestože byl natrénovaný pouze s osmi iteracemi. Ačkoli byl model natrénovaný pouze na datech s jedním zdrojem vln, tak zvládne predikovat i vlnová pole s více zdroji i v mnohem větších výpočetních doménách. Náš model je schopen predikovat subpixelové body s větší přesností než lineární interpolace. Dále je naše řešení schopno predikovat vlnové pole i s podvzorkovaným Laplaciánem, kde jsou pouhé tři vzorky na jednu vlnovou délku. Nejsme si vědomi žádné existující metody fungující s takto řídkou diskretizací.

Keywords

neural solver, Helmholtz equation, graph neural networks, approximation of sound propagation, iterative model, PDE, partial differential equation, machine learning

Klíčová slova

neurální výpočet, Helmholtzova rovnice, grafové neuronové sítě, aproximace šíření zvuku, iterativní model, PDE, parciální diferenciální rovnice, strojové učení

Reference

NGUYEN, Son Hai. *Approximation of Sound Propagation by Neural Networks*. Brno, 2022. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Prof. Ing. Adam Herout, Ph.D.

Rozšířený abstrakt

Transkraniální léčba ultrazvukem byla schválena úřadem pro kontrolu potravin a léčiv (FDA[1]) na léčbu Parkinsonovy choroby a esenciálního tremoru [43, 16], probíhají také klinické studie využití léčby ultrazvukem za účelem na vytvoření mikrobublinek v mozku, které umožní doručení léků do specifických oblastí mozku [22]. Šíření ultrazvuku skrze lebku je popsáno Helmholtzovou rovnicí. Následující rovnice popisují obecnou diferenciální rovnici:

$$\begin{aligned} (\mathcal{L}_a u)(\boldsymbol{x}) &= g, \quad x \in \Omega \\ u(\boldsymbol{x}) &= 0, \quad x \in \partial\Omega \end{aligned} \tag{1}$$

kde a představuje parametry diferenciálního operátoru $\mathcal{L}$, u je neznámá funkce, g popisuje rozložení zdrojů v prostoru a $\boldsymbol{x}$ reprezentuje souřadnice v prostoru.

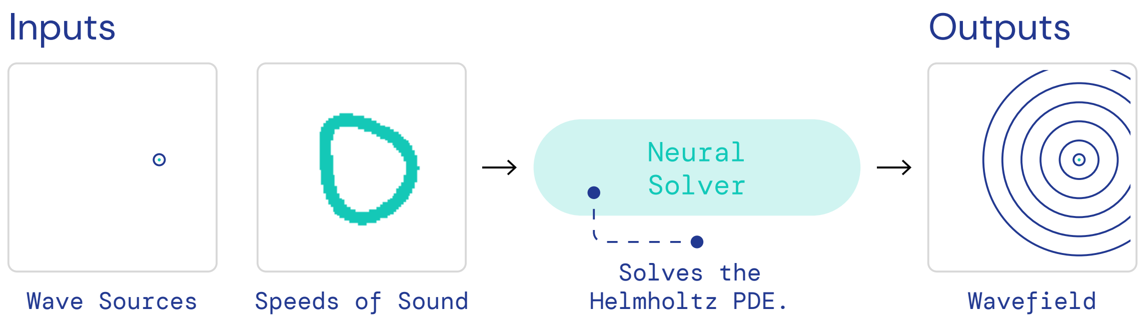


Figure 1: Navrhnutý model z mapy rychlosti zvuku a umístění zdrojů vln predikuje šíření vln modelované Helmholtzovou rovnicí.

Motivace neurálního výpočtu. Jak uvádí Treeby a Cox [51], konvenční metody pro řešení *parciálních diferenciálních rovnic* jako metoda konečných prvků [18] nebo metoda konečných diferencí [33] vyžadují velmi jemnou diskretizaci prostoru, což vede k pomalejšímu řešení dané rovnice. Metoda k-Wave pracuje s hrubší diskretizací díky využití *pseudo-spektrální metody* [6, 7, 31, 48] pro výpočet derivací. Nicméně k-Wave stále nedosahuje dostatečné rychlosti ve 3D prostoru.

Existující řešení. Neurální výpočet má za cíl nahradit konvenční metody za účelem urychlení či aproximace řešení diferenciálních rovnic. Většina metod využívá *konvoluční neuronové sítě* [47, 50] nebo *fyzikálně informované neuronové sítě* (PINN[2]) [37, 23, 9, 45, 12]. Konvoluční neuronoví sítě jsou omezeny strukturou dat, které musí být navzorkované na pravidelné mřížce, zatímco PINN metody jsou nezávislé na mřížce, na kterých byla data navzorkována. Nicméně PINN pouze reprezentuje řešení $u(\boldsymbol{x})$ diferenciální rovnice a neřeší ji přímo, proto musí být síť natrénována znova pro každou novou množinu parametrů a dané diferenciální rovnice.

Každý fyzikální systém může být popsán grafem, kde vzorek je reprezentován jako uzel v grafu. Sanchez-Gonzalez et al. [40] propojuje všechny uzly v určité vzdálenosti, kdežto Pfaff et al. [34] pracuje přímo s hranami vytvořené mesherem. Obě metody řeší dynamické

[1]FDA – Food and Drug Administration
[2]PINN – Physics-informed Neural Networks

systémy jsou založeny na *encode-process-decode* architektuře, která je definována pomocí *Graph Network* frameworku Battaglia et al. [4].

Další kategorií neurálního výpočtu jsou neurální operátory. První metodou v této kategorii je *Graph Kernel Network* (GKN) [2], která přímo kóduje Greenovu funkci dané parciální diferenciální rovnice do parametrů sítě. Jelikož tato metoda reprezentuje prostor jako graf a spojuje každý uzel se všemi ostatními uzly, počet hran roste kvadraticky s počtem uzlů. Druhou metodou patřící do neurálních operátorů je *Fourier Neural Operator* (FNO) [26], která nahrazuje kernel reprezentující Greenovu funkci za konvoluci ve Fourierovým prostoru.

Architektura modelu. V rámci této diplomové práce jsem navrhl iterativní metodu založenou na *grafových neuronových sítích* [4, 34, 40], která řeší Helmholtzovu rovnici. Daná síť je založena architektuře encode-process-decode, kde každá část je realizována pomocí plně propojených neuronových sítí. Kromě parametrů a dané diferenciální rovnice, je na vstup dále vložen residuál dané rovnice [47]. Empiricky jsme zjistili, že residuál je esenciální pro náš model.

Evaluace. Schopnost modelu generalizovat byla úspěšně otestována na datech mimo distribuci trénovacích data. Model byl schopen predikovat vlnové pole i s více zdroji, přestože byl natrénovaný na datech pouze s jedním zdrojem. Dále je model invariantní vůči velikosti výpočetní domény. Přestože byl natrénovaný pouze na datech v doméně 96×96, naše metoda je schopna provádět inferenci i v doméně o velikosti 512×512.

Přestože model primárně pracuje na datech s pravidelnou mřížkou, zvládne predikovat subpixelové vzorky s větší přesností než při použití lineární interpolace, tyto vzorky nemusí být navzorkované na pravidelné mřížce. Tudíž je náš naše metoda schopna provádět i super-resolution. Dále byla otestována schopnost našeho modelu pracovat s podvzorkovaným residuálem (tři vzorky na jednu periodu). Naše metoda jednoznačně překonala konvenční metody, které nejsou schopny pracovat v tomto nastavení.

Závěr Přestože naše metoda není rychlejší než dostupné metody, je nutno brát v potaz, že implementace naší metody není nijak optimalizována a ani zde nebyl pokus ji urychlit. Tato práce slouží především jako ověření konceptu neurálního výpočtu založeném na grafových neuronových sítích pro řešení Helmholtzovi rovnice.

Navržená metoda funguje jako iterativní solver, kde se iterativně volá grafová neuronová síť. Náš model je schopný predikovat vlnové pole ve větších doménách i s více zdroji, čímž dokazuje svoji schopnost generalizovat. Narozdíl od konvolučních neuronových sítích je naše metoda schopna provádět super-resolution pouze v oblastech zájmu. Dále je navržený model schopen pracovat i s podvzorkovaným Laplaciánem, kde má pouhé tři vzorky na jednu periodu. Nejsme si vědomi žádné konvenční ani neurální metody, která pracuje v tomto nastavení. Předběžné výsledky naší práce byly prezentovány na studentské konferenci Excel@FIT2022, kde naše práce získala všechny tři ocenění a v blízké budoucnosti plánujeme vydání vědeckého článku založeném na výsledcích popsaných v této práci. Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy České republiky prostřednictvím e-INFRA CZ (ID:90140).

Approximation of Sound Propagation by Neural Networks

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of prof. Ing. Adam Herout Ph.D.. The supplementary information was provided by doc. Ing. Jiří Jaroš Ph.D., Dr. Antonio Stanziola and MUDr. Jana Hantáková. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Son Hai Nguyen
May 18, 2022

Acknowledgements

Firstly, I am very grateful to my supervisor prof. Ing. Adam Herout Ph.D. for the patient guidance and advice he has provided. I am extremely thankful to my girlfriend MUDr. Jana Hantáková for mental support as well as neurological works analysis. I am also deeply indebted to Dr. Antonio Stanziola for the extensive guidance, his never-ending positivity and support. Additionally, I would like to express my graditude to doc. Ing. Jiří Jaroš Ph.D. for his advice and given access to the supercomputer. I would also like to thank Ing. Marta Jaroš for the access to computational resources. Last but not least, I am grateful for the mental support from my family. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90140).

Contents

Chapter 1

Introduction

The transcranial ultrasound therapy has been recently approved by Food and Drug Administration (FDA) for the treatment of essential tremor and Parkinson's disease [43]. Currently, there are experimental treatments for Alzheimer's disease and brain tumor using ultrasound to create microbubbles, which enable targeted drug delivery in brain tissue [22].

The ultrasound transmitter positioning requires a simulation of the ultrasound propagation through a skull, which depends on solving the Helmholtz equation. The waves have to be focused on the area of interest in order to make thermal ablation and other treatments [16] possible (see Figure 1.1). The simulation using traditional solvers requires a lot of computational resources. Several works tackle the problem of solving PDEs using neural networks (neural solvers), focusing on reducing the simulation time.

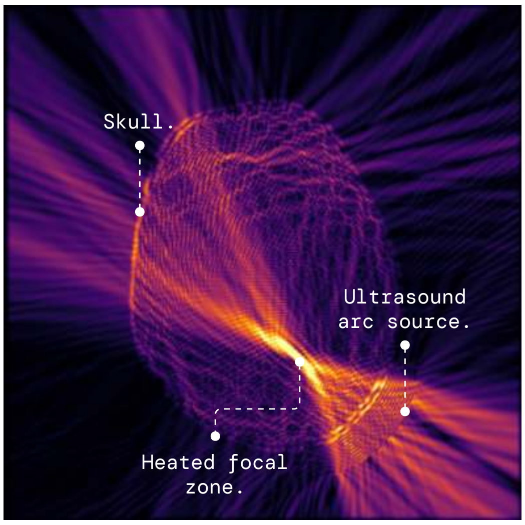


Figure 1.1: High-intensity–focused ultrasound ablation (HIFU) is one of the noninvasive treatments utilizing the focused ultrasound. In the focal zone, the acoustic energy is transformed into heat, resulting in necrosis of surrounding tissue. Figure adapted from Stanziola et al. [47].

Most neural physical systems adopt *Convolutional Neural Networks* (CNN) [47, 50] or *Physics-informed neural networks* (PINN) [37, 23, 9, 45, 12]. The first family of methods works only with data sampled on a regular grid. On the other hand, the latter group of methods is mesh-independent. However, the networks need to be retrained for every set of new PDE parameters.

Our method aims to tackle these problems by employing *Graph Neural Networks* (GNN) [42, 14], allowing more variable mesh than CNNs. We use graph edges to control the message passing between the nodes, corresponding with the samples in a space. The network is structured as an iterative solver [38]. Only a forward pass of the model is required to solve the Helmholtz equation for a new set of parameters.

In Section 2 this work describes *partial differential equation* (PDE) and its boundary conditions for the modeling of the wave propagation. Section 4 and Section 3 formulate the graph neural networks and the methods used for a physics-based deep learning. Section 6 describes our proposed method for simulating the propagation of waves. Lastly, Section 7 contains the evaluation of the method's generalization, its behavior on different sampling grids, and the effect of pruning on the accuracy and its capability to predict the wavefield with a downsampled residual.

Chapter 2

Equations Governing Propagation of Waves

This section outlines equations used for wave propagation modeling – such as sound waves, water waves, etc. Nevertheless, this work is based on equations used in sound-waves modeling. The wave equation describes the propagation of waves through space:

$$\frac{\partial^2 u(\boldsymbol{x})}{\partial t^2} = c(\boldsymbol{x})^2 \nabla^2 u(\boldsymbol{x}), \tag{2.1}$$

where $u : \mathbb{R}^D \to \mathbb{C}$ is a wavefield, $D \in \mathbb{Z}^+$ denotes the dimensionality, $\boldsymbol{x} \in R^D$ denotes a spatial coordinate, t stands for time variable and $c : \mathbb{R}^D \to \mathbb{R}^+$ is a speed of sound.

However, in transcranial focused ultrasound therapy [22, 16, 43], the ultrasound beam is applied for a period of time exceeding the time required to reach a steady state. Consequently, in this work, the time-independent Helmholtz equation is used to model the propagation of waves (Figure 2.1).

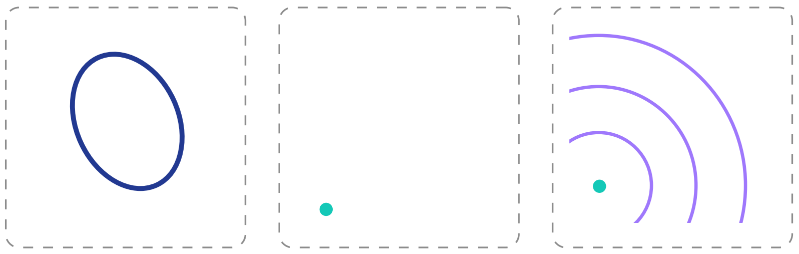

(a) Speeds of sound c. (b) Source distributions ρ. (c) Wave field u.

Figure 2.1: Given the inhomogeneous *speeds of sound* c and source distributions ρ, the objective is to retrieve steady-state wave field u.

2.1 Helmholtz Equation

The Helmholtz equation is a *linear* partial differential equation, and it represents a time-independent form of the Wave equation (Equation (2.1)). The time-independent form is achieved by the separation of the variables in the wave equation. Although the simulation of an ultrasound propagating through the skull involves nonlinear, shear, and other effects, Stanziola et al. [47] present that these effects can be applied additionally after calculating the wavefield modeled by the Helmholtz equation. Thus, we focus on modeling a simplified model of wave propagation described by the Helmholtz equation subjected to the Sommerfield radiation condition at infinity [46]:

$$\left[\nabla^2 + \left(\frac{\omega}{c(x)}\right)^2\right] u(x) = \rho(x), \tag{2.2}$$

$$\text{s.t.} \lim_{|x|\to\infty} |x|^{\frac{n-1}{2}} \left(\frac{\partial}{\partial |x|} - i\frac{\omega}{c_0}\right) u(x) = 0, \tag{2.3}$$

where $D \in \mathbb{Z}^+$ is the dimensionality, $\omega \in \mathbb{R}$ stands for the angular velocity of the source, $\boldsymbol{x} \in \mathbb{R}^D$ is a spatial coordinate, $c : \mathbb{R}^D \to \mathbb{R}^+$ is the inhomogeneous speed of sound (SOS) at the certain point $\boldsymbol{x}$, $\rho : \mathbb{R}^D \to \mathbb{C}$ denotes the forcing term, and $u : \mathbb{R}^D \to \mathbb{C}$ is the **unknown** acoustic wavefield. The inhomogeneous SOS is considered only within a domain of interest Ω, for the rest of the domain $\partial\Omega$ the speed of sound is homogeneous with a value equal to c_0. Graphical example of the components of the Helmholtz equation is shown in Figure 2.2.

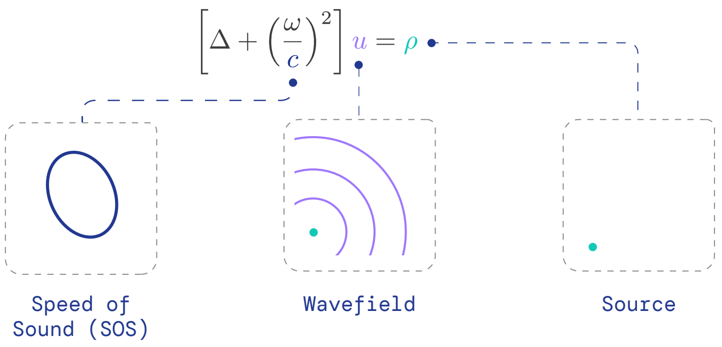

$\left[\Delta + \left(\frac{\omega}{c}\right)^2\right] u = \rho$
Speed of Sound (SOS)
Wavefield
Source

Figure 2.2: In this work, speeds of sound c contains a model of a skull, which has a different speed of sound compared to the surrounding medium. Source values ρ are zero, except at the location of a source, where the value is set to the source amplitude. *Inverse problem* is not contemplated in this work. Thus the wavefield u is the only unknown in the equation.

2.2 Perfectly Matched Layer

In order to satisfy the Sommerfield radiation condition in a finite domain Ω, a boundary condition such as *Perfectly Matched Layer* (PML) [5] can be applied. PML is an artificial layer surrounding the domain Ω attenuating all incoming waves to prevent the reflections from the domain's border, as depicted in Figure 2.4. Consequently, PML simulates an infinite domain, as shown in Figure 2.3.

By employing the PML, an absorption term is introduced into the derivative operators:

$$\frac{\partial}{\partial \eta} \to \frac{1}{\gamma_\eta}\frac{\partial}{\partial \eta}, \tag{2.4}$$

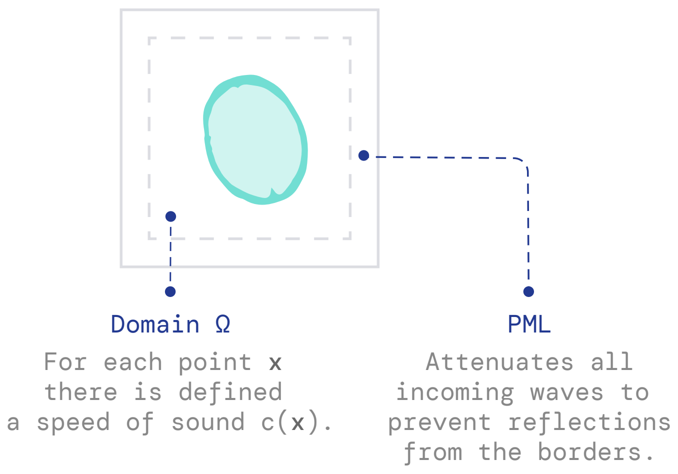

Domain Ω
For each point x there is defined a speed of sound c(x).
PML
Attenuates all incoming waves to prevent reflections from the borders.

Figure 2.3: To satisfy Sommerfield condition (Equation (2.3)), an artificial layer (PML) is wrapped around the domain Ω to attenuate all incoming waves. Adapted from Stanziola et al. [47].

where $\eta = x_1, x_2, \ldots, x_n$, x_j corresponds to the j-th spatial dimension, and where

$$\gamma_\eta = \begin{cases} 1, & \eta \in \Omega \\ 1 + \dfrac{i}{\omega}\sigma(\eta), & \text{otherwise} \end{cases} \tag{2.5}$$

where ω is the angular velocity from Equation (2.2), Ω is the domain of interest, and the absorption profile σ is defined by the following equation:

$$\sigma(\eta) = \sigma_{\max}\left(1 - \frac{\hat{\eta}}{\Delta L}\right)^2, \tag{2.6}$$

where $\sigma_{\max}$ is maximum PML absorption parameter, ΔL defines the width of a PML and $\hat{\eta}$ is the spatial distance from the domain border in a given dimension.

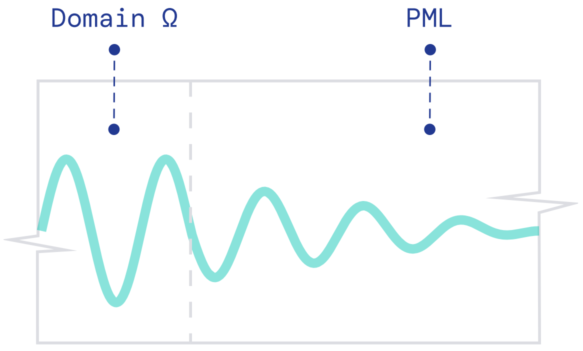


Figure 2.4: The PML layer attenuates all incoming waves quadratically (Equation (2.5)) in order to satisfy the Sommerfield radiation condition at infinity.

When the PML is used as a boundary condition, the Laplace operator is transformed in the following way:

$$\nabla^2 = \sum_\eta \frac{\partial^2}{\partial \eta^2} \to \hat{\nabla}^2 = \sum_\eta \frac{1}{\gamma_\eta}\frac{\partial}{\partial \eta}\left(\frac{1}{\gamma_\eta}\frac{\partial}{\partial \eta}\right) \tag{2.7}$$

Chapter 3

Physics-Based Deep Learning

Watt et al. [56] defines neural networks as computational models that describe empirical data and the underlying phenomena f^*, with little or no human involvement. We denote *idealized* parameters of functions with a superscript $*$. Goodfellow et al. [13] view a feed-forward network as a mapping $\boldsymbol{y} = f(\boldsymbol{v}, \boldsymbol{\theta})$ from an input $\boldsymbol{v}$ to an output $\boldsymbol{y}$ (e.g. category, heatmaps, etc.). An example of the mapping is depicted in Figure 3.1.

But the one question that arises is if the neural network is able to solve a PDE? According to *Universal Approximation Theorem*, [17] any single-layer network with a nonlinear activation function can approximate **any** continuous function to a reasonable accuracy. Therefore, the network should be able to represent a PDE solution at least.

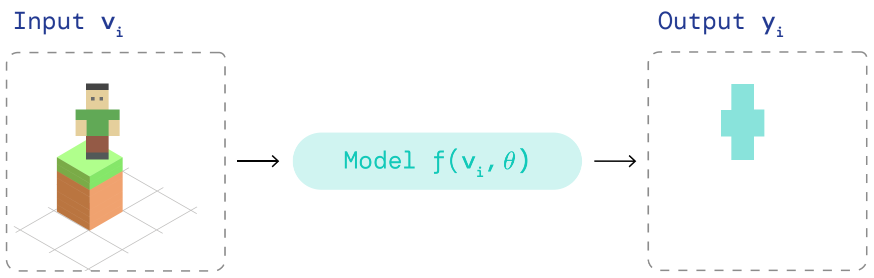


Figure 3.1: A feedforward neural network can be defined as a mapping from an input $\boldsymbol{x}$ to an output $\boldsymbol{y}$. In this particular example, an input image is mapped to a segmentation mask (the person is segmented).

According to Goodfellow et al. [13], most of the learning algorithms can be divided into two categories – **supervised learning** and **unsupervised learning**. The main difference between the two approaches is the presence of a *label* – representing reference or ground truth "solution".

Supervised learning. Given a dataset $\mathcal{D}$ containing features $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n$, each feature $\boldsymbol{v}_i$ is associated with a label $\boldsymbol{y}_i$ – the true label is known. Formally, finding the optimal model weights $\boldsymbol{\theta}^*$ is defined as:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{v}_i, \boldsymbol{y}_i) \in \mathcal{D}} L(f(\boldsymbol{v}_i, \boldsymbol{\theta}), \boldsymbol{y}_i), \tag{3.1}$$

where $L(\cdot,\cdot)$ symbolizes a loss function.

Unsupervised learning. Unlike supervised learning, the unsupervised approach omits the ground truth labels $\boldsymbol{y}_i$. This approach is particularly useful when obtaining the ground truth is not viable – computationally infeasible, or requires an immoderate amount of time. Formally, acquiring an optimal set of model weights is described by the following equation:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{v}_i \in \mathcal{D}} L(f(\boldsymbol{x}_i, \boldsymbol{\theta})) \tag{3.2}$$

3.1 Neural Network Optimization

Definition of the optimal model weights θ^* is described by Equation (3.1) and Equation (3.2). But how would one find the optimal model weights θ^*? In order to find the optimal weights $\boldsymbol{\theta}^*$, optimization algorithms are used. Methods such as *Adam* [20] or *SGD* (with momentum) [39] belong to the *first-order* optimization methods. These methods are still the most popular ones, due to their low memory consumption, computational requirements, and robustness. As an example, the Vanilla Gradient Descend algorithm is described by Algorithm 3.1.

Listing 3.1: Pseudo-code for Gradient Descend algorithm, where α, θ, $\mathcal{D}$ denote learning rate, model parameters and dataset, respectively. $J(\cdot,\cdot)$ is an objective function, and $L(\cdot,\cdot)$ is a loss function. Each iteration contains following steps: computing a loss value, computing derivative of the loss value w.r.t. parameters θ, and updating parameters using computed gradients, where α denotes a learning rate of an optimizer.

```
1. Compute prediction $\hat{\boldsymbol{y}} = f(\boldsymbol{x}, \boldsymbol{\theta}_i)$
2. Compute loss value $L(\hat{\boldsymbol{y}}, \boldsymbol{y})$ .
3. Compute gradients w.r.t parameters $\frac{\partial}{\partial \boldsymbol{\theta}_i} J(\boldsymbol{\theta}_i, \mathcal{D})$.
4. Update parameters $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \frac{\partial}{\partial \boldsymbol{\theta}_i} J(\boldsymbol{\theta}_i, \mathcal{D})$
5. Go to step 1, until $|L(\hat{\boldsymbol{y}}, \boldsymbol{y})|$ is sufficiently small or other condition is met.
```

Although the second-order methods converge in fewer optimization iterations, they are less popular due to their heavy memory, and computation consumption [49]. Even though methods such as *BFGS* [24] or *Levenberg-Marquardt* [32] tackle the problem with high memory footprint, first-order methods are still the preferred option due to their scalability for larger neural networks as can be seen in Figure 3.2.

3.2 Categorization

Physics-Based Deep Learning (PBDL) can be categorized in two ways. The first is based on the unknown variable. The other type of categorization distinguishes the integration depth of a physics model into a model.

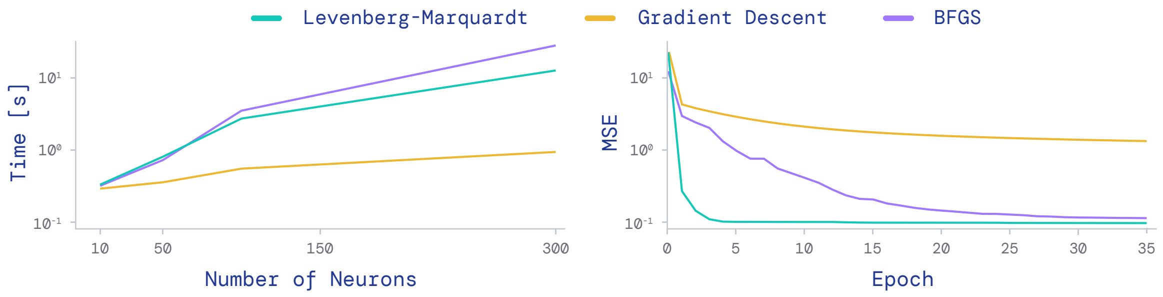


Figure 3.2: Although second-order methods (BFGS, Levenberg-Marquardt) converge in significantly less epochs than SGD, the required computational time increases exponentially with the number of neurons.

3.2.1 Based on the Unknown

The first concept of categorization of Physics-Based Deep Learning (PBDL) is based on the unknown variable. If the method's objective is to find the parameters of a physical system $a(\boldsymbol{x})$ based on the observations $u(\boldsymbol{x})$, then this method belongs to the *inverse problems* category. Otherwise, it is a part of *forward simulations* group, where the task is to predict a state or temporal evolution (see Figure 3.3). As an example, given the seismic waves, the goal of an inverse problem would be to locate the source of the earthquakes. In the forward simulation, the source of the waves is known, and the goal is to solve which locations are affected and how they are affected by the seismic waves.

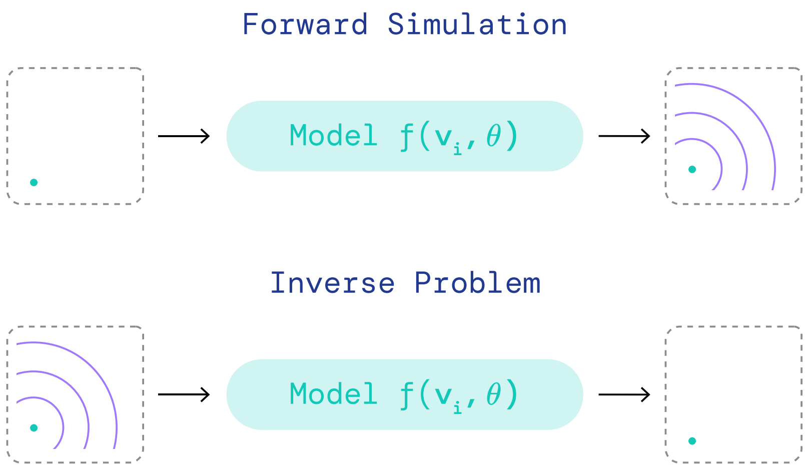


Figure 3.3: The PBDL can be categorized based on the unknown. If the parameters of a physics model are unknown, it belongs to the category of inverse problems. Otherwise, it belongs to the forward simulation category, where the model parameters $a(\boldsymbol{x})$ are known, and the goal is to solve the unknown field $u(\boldsymbol{x})$.

3.2.2 Based on the Physics Integration

The physics model can be integrated into the deep learning (DL) model in multiple ways. Thuerey et al. [50] distinguishes three main categories based on the integration depth of a physics model:

Supervised. With the *supervised* approach, a physical model is only used to generate the data. There is no other interaction with the physical model.

Loss-terms. A PDE describing the physical model is encoded in the loss function. More detailed description of *physics loss* is in Section 3.3.

Interleaved. Interleaved category represents the tightest coupling between deep learning and a physical model. The physical simulation (solver) is intertwined directly with the learning process. Naturally, a differentiable solver is expected.

3.3 Physics-Informed Loss Function

The ground-truth data for physical systems are usually generated in the supervised settings using the numerical simulation of a physical model. Thuerey et al. [50] claim that with the knowledge of a target physical model, specific inductive biases can be made toward improving the training process.

In this work, we only consider time-independent *partial differential equation* (PDEs). Using the notation framed by Thuerey et al. [50], a PDE can be formulated as a function $\mathcal{F}$ of the derivatives of an unknown $u(\boldsymbol{x})$:

$$\mathcal{F}(u, u_x, u_{xx}, \dots, u_{xx\dots x})(\boldsymbol{x}) = g(\boldsymbol{x}), \tag{3.3}$$

where the x subscript denotes a spatial derivative w.r.t to x, and g stands for a forcing term. The notation is slightly abused by letting x be any dimension in an attempt to keep the equation short and clear.

By defining a residual R using Equation (3.4), R should be closer to zero the closer the approximation $\boldsymbol{u}$ is to the true solution $\boldsymbol{u}^*$. Therefore the residual acts as a loss function that should be minimized.

$$R = \mathcal{F}(u, u_x, u_{xx}, \dots, u_{xx\dots x})(\boldsymbol{x}) - g(\boldsymbol{x}) \tag{3.4}$$

While Thuerey et al. [50] states that using only the residual term R as a loss function $L_{physics}$ would result in a random offset or scale, we assume a forcing term $g(\boldsymbol{x})$ to be nonzero because we use the Helmholtz equation with constant sources, so the scaling of the unknown $u(\boldsymbol{x})$ is constrained. Therefore, $L_{physics}$ loss defined by Equation (3.5) can be used as a loss function without causing random scaling issues [45, 47]. Subsequently, the training objective is defined by Equation (3.6).

$$L_{physics} = ||(R(u))(\boldsymbol{x})||_2^2 \tag{3.5}$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{v}_i \in \mathcal{D}} L_{physics}(f(\boldsymbol{v}_i, \boldsymbol{\theta})) \tag{3.6}$$

However, the combined loss can help to pin down the prediction in more points not just in $g(\boldsymbol{x})$ [50]. The resulted objective function is described as:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{\boldsymbol{v}_i \in \mathcal{D}} \lambda_1 ||\boldsymbol{y}_i - f(\boldsymbol{v}_i, \boldsymbol{\theta})||_2^2 + \lambda_2 L_{physics}(f(\boldsymbol{v}_i, \boldsymbol{\theta})), \tag{3.7}$$

where λ_1, λ_2 indicate how the supervised and the physics term contribute to the objective function, respectively.

Example 1: Let Equation (3.8) be the governing equation describing the behavior of a target system, where $u(\boldsymbol{x})$ denote the unknown. With unsupervised settings, a physics loss $L_{physics}$ can be leveraged. By reordering terms of Equation (3.8), the residual R is obtained as follows:

$$\left[\nabla^2 + \left(\frac{\omega}{c}\right)^2\right] u(\boldsymbol{x}) = \rho(\boldsymbol{x}), \tag{3.8}$$

$$R(\boldsymbol{x}) = \left[\nabla^2 + \left(\frac{\omega}{c}\right)^2\right] u(\boldsymbol{x}) - \rho(\boldsymbol{x}). \tag{3.9}$$

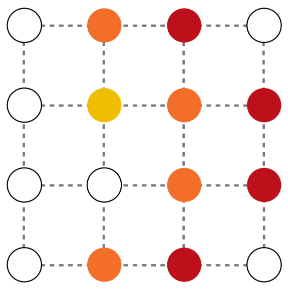
With the defined residual R, we can utilize the physics loss function (Equation (3.5)), which results in:

$$L_{physics} = \left\| \left[\nabla^2 + \left(\frac{\omega}{c}\right)^2\right] u(\boldsymbol{x})\rho(\boldsymbol{x}) \right\|_2^2. \tag{3.10}$$

Chapter 4

Different Viewpoints on Graph Neural Networks

Although *convolutional neural networks* [47, 50] have been a preferred option for physics-based problems, it restricts the form of discretization of the target domain – domain can be only discretized using a regular grid (Figure 4.1).


(a) Regular grid.

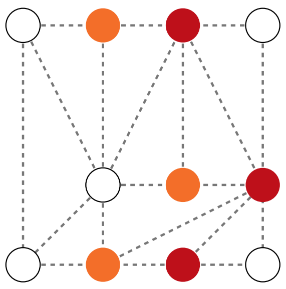

(b) Arbitrary graph structure.

Figure 4.1: Figure (a) depicts an image, which can be represented as a fixed-size grid graph. However, as Figure (b) shows, a graph can hold more complex shapes – in this case the edges were generated using triangulation of nodes.

On the other hand, the discretization restrictions are no longer present with more general structures such as graphs, which allow various discretization. Every physics system can be modeled using a graph, as samples produced by spatial sampling can be represented as nodes in a graph. Sanchez-Gonzalez et al. [40] connect nodes in a certain radius, whereas Pfaff et al. [34] operate directly with the edges produced by a mesher. These two method networks [34, 40] are based on the *encode-process-decode* architecture defined under the *Graph Network* framework [4]. However, the mentioned methods predict only the time evolution of time-dependent PDEs.

Graph Neural Networks (GNN) were introduced by Scarselli et al. [42] and Gori et al. [14]. Li et al. [25] added a new network architecture to the "GNN" family later. Since then there have been numerous works regarding neural networks for graphs [21, 54, 52].

4.1 Message Passing Network

Gilmer et al. [12] formed the Message passing neural networks (MPNN) framework, which unifies a portion of the models working with graphs. Let a tuple $G = (V, E)$ denote a graph with a set of edges E and a set of nodes V. Connection between the nodes is denoted by the edge $\boldsymbol{e}_{ij}$, which connects nodes $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$. The edge can also contain features alongside the representation of the connection between the nodes.

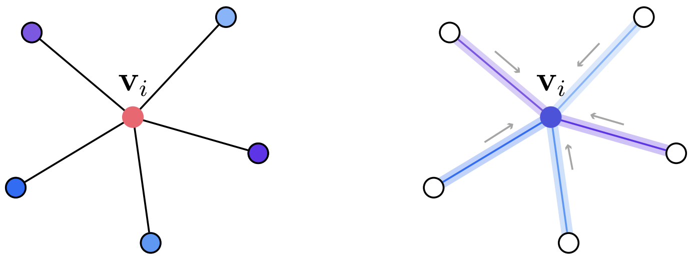


Figure 4.2: In a message passing step, the updated node's state is computed from its current state and aggregated messages from its neighbors. Adapted from Towards Data Science blog[1].

The MPNN framework can be split into three steps: *message pass*, *update* and *decode* (see Algorithm 4.1). Message pass step propagates the information between the neighbours K-times, where each additional message pass step increases the node's *receptive field*.

Listing 4.1: An algorithm of a MPNN framework consists of three steps: message, update and decode.

```
1. Message.
   A message $m_i^{k+1} = \rho^v_{j\in N(i)}\left(\phi^v\left(v_i^k, v_j^k, e_{ij}\right)\right)$ is computed for each node. $\rho^v$ represents
   an aggregator, which aggregates messages from the neighborhood $\{v_j\}_{j\in N(i)}$
   of the node $v_i$. $\phi^k$ is a differentiable function. $k \in \mathcal{Z}^+$ denotes a message pass
   iteration step.

2. Update.
   Each node state $v_i$ is updated using following equation: $v_i^{k+1} = \gamma\left(v_i^k, m_i^{k+1}\right)$, where $\gamma$
   is a differentiable function. For more message pass steps, go to step 1.

3. Decode.
   After $K$ steps, create a final feature vector representing the whole
   graph $\hat{y} = \rho^u\left(\{v_i^K\}_{j\in G}\right)$. $\rho^u$ denotes a differentiable function.
```

A *message* m_i^{k+1} of i-th node at $k+1$ message pass iteration is computed using a differentiable function ϕ^v – such as multi layer perceptron (MLP). Function ϕ^v takes the edge e_{ij} and nodes it connects as an input(see Equation (4.1)). Messages of neighbouring nodes are aggregated using a permutation invariant function ρ^v such as sum, mean, etc.

[1]Figure adapted from https://towardsdatascience.com/graph-neural-networks-as-neural-diffusion-pdes-8571b8c0c774.

$$m_i^{k+1} = \rho^v_{j\in N(i)}\left(\phi^v\left(v_i^k, v_j^k, e_{ij}\right)\right), \tag{4.1}$$

where $N(i)$ is a set of indices of nodes neighboring with a node v_i. e_{ij} denotes an edge connecting nodes v_i and v_j.

Following the message step is the *update* step, where the nodes' features v^{k+1} are updated using the previous state v^k and aggregated message m^{k+1}. The update step is defined for i-th node by Equation (4.2). The *message pass step* refers to the execution of the update and the message step (Figure 4.2). One message pass step can be view as one iteration of the node update:

$$v_i^{k+1} = \gamma\left(v_i^k, m_i^{k+1}\right), \tag{4.2}$$

where γ is a differentiable function such as MLP. After K message passings, the nodes' features are decoded to a single vector graph representation $\hat{y}$ using a differentiable function ρ^u:

$$\hat{y} = \rho^u\left(\{v_i^K\}_{j\in G}\right) \tag{4.3}$$

4.2 Unified Framework

Gilmer et al. [12] introduced the message-passing neural network (MPNN) and Wang et al. [53] presented the non-local neural network (NLNN), unifying various GNN and GCN architectures, and a portion of self-attention methods, respectively. Battaglia et al. [4] later presented the *graph networks* (GN) framework, which generalizes MPNN as well as NLNN methods.

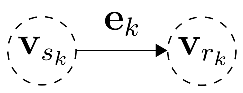


Figure 4.3: Example of the notation used within the GN framework. $\boldsymbol{v}_i$ corresponds to the attributes of i-th node. s_k and r_k are indices of the sender and receiver nodes, respectively. Lastly, $\boldsymbol{e}_k$ denotes attributes of k-th edge.

The GN framework [4] defines graph as a tuple $G = (\boldsymbol{u}, V, E)$. The $\boldsymbol{u}$ denotes a global attribute of a graph. A set of nodes attributes of the cardinality N^v and N^e is defined as $V = \{\boldsymbol{v}_i\}_{i=1:N^v}$. The $E = \{(\boldsymbol{e}_k, r_k, s_k)\}_{k=1:N^e}$ is a set of edges' attributes, where $\boldsymbol{e}_k, r_k \in \{1,\ldots,N^v\}, s_k \in \{1,\ldots,N^v\}$ denote the attributes of an edge, an index of an receiver node and an sender node, respectively (see Figure 4.3).

$$\begin{aligned} \boldsymbol{e}'_k &= \phi^e(\boldsymbol{e}_k, \boldsymbol{v}_{r_k}, \boldsymbol{v}_{s_k}, \boldsymbol{u}) & \bar{\boldsymbol{e}}'_i &= \rho^{e\to v}\left(E'_i\right) \\ \boldsymbol{v}'_i &= \phi^v(\bar{\boldsymbol{e}}'_i, \boldsymbol{v}_i, \boldsymbol{u}) & \bar{\boldsymbol{e}}' &= \rho^{e\to u}\left(E'\right) \\ \boldsymbol{u}' &= \phi^u(\bar{\boldsymbol{e}}', \bar{\boldsymbol{v}}', \boldsymbol{u}) & \bar{\boldsymbol{v}}' &= \rho^{v\to u}\left(V'\right) \end{aligned} \tag{4.4}$$

where $E'_i = \{(\boldsymbol{e}'_k, r_k, s_k)\}_{r_k=i,k=1:N^e}$, $V' = \{\boldsymbol{v}'_i\}_{i=:N^v}$ and $E' = \bigcup_i E'_i$. The ϕ^e and ϕ^n functions compute per-edge and per-node updates, respectively. ϕ^u is used to compute a global update. The ρ corresponds to a function mapping a set of elements to a single element. ρ

function must be invariant to permutations and take variable number of arguments (e.g. summation, product, mean, maximum, etc.) [4]. The number of *message passings* corresponds to the number of update steps. The more message passings there are, the larger neighbourhood is aggregated.

4.2.1 Update steps of a GN block

Given a graph $G = (\boldsymbol{u}, V, E)$, where $\boldsymbol{u}, V$ and E denote global attributes, edges and nodes, respectively, the update steps are defined follows:

Listing 4.2: Description of the graph network's update step. For more details see Section 4.2.

```
1. Edge update $\phi^e$.
   The function $\phi^e$ is applied for each edge $(\boldsymbol{e}_k, r_k, s_k)$, resulting in updated
   attributes $\boldsymbol{e}'_k$. The function $\phi^e$ takes edge's attributes $\boldsymbol{e}_k$,
   receiver's attributes $V_{r_k}$, sender attributes $V_{s_k}$ and the global attributes $\boldsymbol{u}$.

2. Edge aggregation $\rho^{e \to v}$.
   After updating the edges' attributes, edges are aggregated on "node-level".
   All edges which have $i$-th node as a receiver will be aggregated/reduced into a single
   element $\bar{e}'_i$. Selection of the edges is defined as $E'_i = \{(\boldsymbol{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$.

3. Node update $\phi^v$.
   Similarly to the edge update step, all nodes $\boldsymbol{v}_i$ attributes are updated using $\phi^v$,
   resulting in $\boldsymbol{v}'_i$. The arguments of the update function $\phi^v$ are aggregated edges $\bar{e}'_i$,
   node's old attributes $\boldsymbol{v}_i$ and global attributes $\boldsymbol{u}$

4. Node aggregation $\rho^{v \to u}$.
   Given set of all the nodes $V' = \{\boldsymbol{v}'_i\}_{i=1:N^v}$, the aggregation $\rho^{v \to u}$ takes $V'$ and results
   in the aggregated element $\bar{\boldsymbol{v}}'$.

5. Edge aggregation $\rho^{e \to u}$.
   The set of all edges $E' = \bigcup_i E'_i$ is aggregated using $\rho^{e \to u}$ into $\bar{\boldsymbol{e}}'$.

6. Global update $\phi^u$.
   Finally, aggregated edges $\bar{\boldsymbol{e}}'$, aggregated nodes $\bar{\boldsymbol{v}}'$ and old global attributes $\boldsymbol{u}$ are
   transformed using $\phi^u$ into updated global attributes $\boldsymbol{u}'$.
```

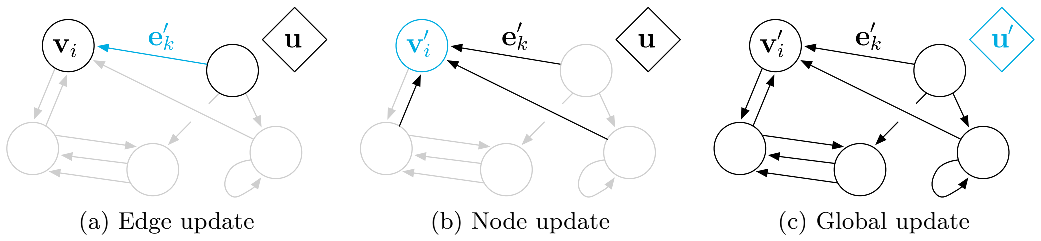


Figure 4.4: Diagram of the updates in a GN block. Black color denotes data, which are used in the update, blue color corresponds to the updated element. See Section 4.2.1 and Equation (4.4) for more details. Adapted from Battaglia et al. [4].

Chapter 5

Exiting Solutions for Solving PDEs Using Neural Networks

Although the section name points out the PDE neural solvers in general, a strong emphasis is put on solving the Helmholtz equation. The Helmholtz equation can be solved using "classical" methods such as finite-difference (FDM) [33] and finite element methods (FEM) [18]. However, Treeby and Cox [51] state that using these methods is computationally expensive. This is caused mainly by a fine resolution requirement, resulting in a too-large grid. To reduce memory as well as computational consumption, the *k-Wave* toolbox [51] uses *k-space* pseudospectral method [6, 7, 31, 48].

k-Wave reduces the required computational time compared to FEM and FDM. However, with every dimension, the required time of a simulation rises by an order of magnitude, thus making the conventional solvers still computationally expensive [47]. Subsequently, the neural solvers have been recently extensively researched, with the premise of decreasing computational demands over the conventional numerical methods. We consider a general PDE with parameters a of the following form:

$$\begin{aligned} (\mathcal{L}_a u)(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad x \in \Omega \\ u(\boldsymbol{x}) &= 0, \quad x \in \partial\Omega, \end{aligned} \tag{5.1}$$

where $\mathcal{L}$ is a differential operator, and u, g are functions on the spatial domain. g represents the forcing term, whereas u is the unknown in the forward simulation. Spatial coordinate is denoted by $\boldsymbol{x}$.

5.1 Physics-informed Neural Networks

Recently, there has been a growing interest in mesh-independent methods. Various *Physics-informed neural network* (PINN) methods [12, 45, 9, 37, 23] use *multilayer perceptron* (MLP) to map a spatial coordinate $\boldsymbol{x}$ directly to a solution $u(\boldsymbol{x})$. Thus, PINN methods learn to represent the solution instead of computing one. An optimization algorithm is used to find the closest representation to the true solution $u^*(\boldsymbol{x})$. However, these networks need to be retrained for each set of new PDE parameters a. Nonetheless, querying a new position $\boldsymbol{x}$ only requires a forward pass of the model, making it mesh-independent.

PINN [36] utilizes an underlying PDE by encoding it into the physics loss (see Section 3.3). The neural network takes the spatial coordinates $\boldsymbol{x_i}$ as an input and outputs a physical quantity $u_i(\boldsymbol{x_i})$. Auto-differentiation is utilized to compute the physics loss containing gradients w.r.t. input. Given that the model directly represents the unknown function and uses the auto-differentiation to compute its derivatives, the PINN methods enforce a strong-form solution. An example of the pipeline of PINN methods can be seen in Figure 5.1.

Despite that Chamberlain et al. [8], shows the network architecture and topology can be designed to correspond with underlying PDE. PINN methods generally use a fully connected shallow neural network [45, 36, 55, 9]. The second-order methods are employed as a result of using the shallow neural networks with few parameters, to assure quicker convergence to the PDE solution.

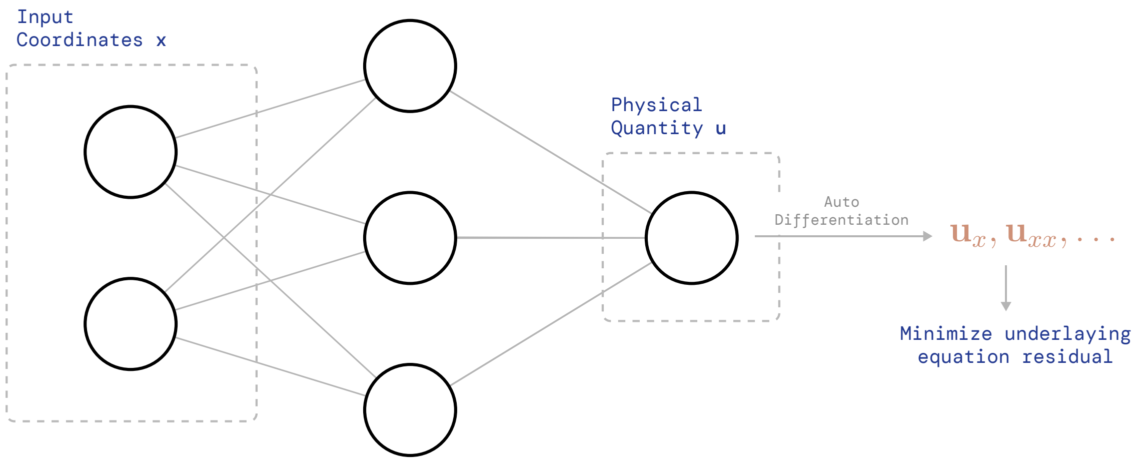


Figure 5.1: In PINN methods, the spatial coordinates are used as an input of a neural network. The network outputs the prediction of a physical quantity. The gradients of the output w.r.t. input are computed effortlessly by using the auto-differentiation. PINN approaches usually use a fully connected shallow neural network [36, 45, 9]. The figure is based on Ben Moseley blog[1].

5.2 Neural Operators

Another branch of neural solvers called *Neural operators* [2, 26] aims to learn a direct mapping between the infinite-dimensional function spaces from forcing term $g(\boldsymbol{x})$ to an unknown $u(\boldsymbol{x})$ (see Equation (5.1)), thus making them discretization-invariant.

5.2.1 Graph Kernel Network

Anandkumar et al. [2] architecture design is motivated by reformulating the linear PDEs using the Green's function G. The Green's function $G : \Omega \times \Omega \to \mathbb{R}$ is a solution to the following problem:

$$(\mathcal{L}_a G(\boldsymbol{x}, \boldsymbol{s}) = \delta_x(\boldsymbol{s}), \tag{5.2}$$

[1]Figure based on https://benmoseley.blog/my-research/so-what-is-a-physics-informed-neural-network

where $\boldsymbol{x} \in \Omega$, $\boldsymbol{s} \in \Omega$ and δ_x is a Dirac delta function centered at the point $\boldsymbol{x}$. $\mathcal{L}$ is a linear differential operator with parameters a. Then the solution $u(\boldsymbol{x})$ can be presented as:

$$u(\boldsymbol{x}) = \int_\Omega G(\boldsymbol{x}, \boldsymbol{s}) f(s)\, ds. \tag{5.3}$$

Note that Green's function depends on parameters a, and is different for each set of parameters a.

Graph Kernel Networks (GKN) directly approximates the Green's function by using a graph neural network. Additionally, the integral is viewed as an averaging aggregation of the messages in the graph neural network (see Section 4.1). The resulting approximation is described by the following equation:

$$u^{t+1}(\boldsymbol{x}) = \sigma\left(W u^t(\boldsymbol{x}) + \left(\mathcal{K}(a;\theta)u^t\right)(\boldsymbol{x})\right), \tag{5.4}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function, $W \in R^{n \times n}$ is parametrized matrix, θ are the parameters of the kernel κ and the kernel integral $\mathcal{K}$ is defined as follows:

$$\left(\mathcal{K}(a,\theta)u_t\right)(\boldsymbol{x}) = \int_\Omega \kappa\left(\boldsymbol{x}, \boldsymbol{s}, a(\boldsymbol{x}), a(\boldsymbol{s}); \theta\right) u_t(\boldsymbol{x})\, ds, \tag{5.5}$$

The main disadvantage of GKN is connecting each node with all the other nodes, making it computationally and memory expensive in a larger domain. As the authors state, decreasing the resolution causes errors, making the method less competitive. Other methods, such as *HelmNet* [47], usually cast the PDE in a finite-dimensional Euclidian space. Therefore, these methods are discretization-dependent.

5.2.2 Fourier Neural Operator

Li et al. [26] replace the kernel operator defined by Equation (5.5) with the *Fourier integral operator* $\mathcal{K}$:

$$\left(\mathcal{K}(\theta)u_t\right)(\boldsymbol{x}) = \mathcal{F}^{-1}\left(P_\theta \cdot \mathcal{F}(u_t)\right)(\boldsymbol{x}), \tag{5.6}$$

where $P_\theta \in R^{n \times n}$ is a convolution matrix in Fourier space, $\mathcal{F}$ and $\mathcal{F}^{-\infty}$ denote Fourier transform and its inverse, respectively.

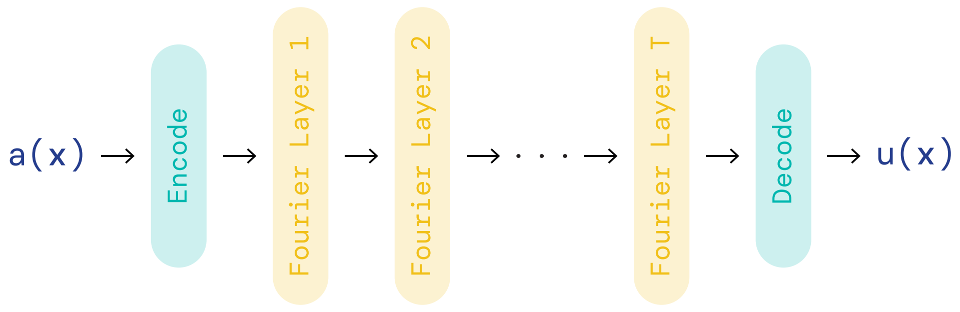


Figure 5.2: The Fourier Neural Operator adopts the encode-process-decode. Encode and decode blocks are implemented using fully connected layers. The process part is realized using the *Fourier layers*. Adapted from Li et al. [26].

The Fourier neural operator has an iterative structure. It is composed of encode, process, and decode blocks. The encode and decode parts are implemented using a fully connected layer, as depicted in Figure 5.2. The process part is realized by using the Fourier layer $\mathcal{F}$, which transforms the input into Fourier space, performs a linear transform P, and filters out higher frequencies. Then the inverse Fourier transform $\mathcal{F}^{-1}$ is applied. Additionally, the Fourier layer input is linearly transformed in its original space by tensor W. This result is then added to the output of the inverse Fourier transform (see Figure 5.3).

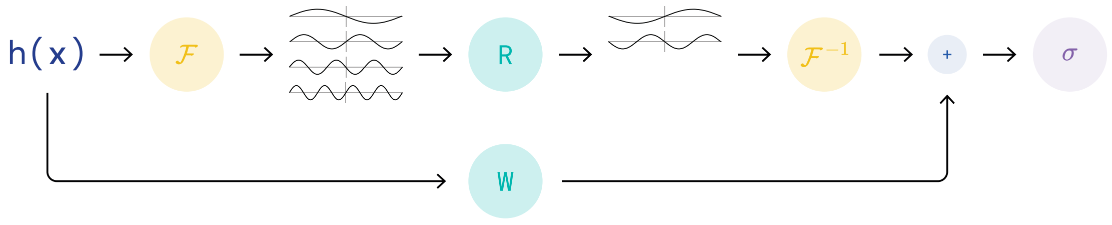


Figure 5.3: The Fourier layer transforms the input h into Fourier space, performs a linear transformation P for the lower frequencies, filters out higher frequencies and applies inverse Fourier transform. Adapted from Li et al. [26].

Fourier Neural Operator (FNO) achieves lower error than Graph Kernel Method, at the cost of requiring uniformly sampled data. Additionally, it has a significantly higher speed than the traditional solvers. In the original papers, both neural operators are evaluated on Burgers' equation, Darcy flow, and Navier-Stokes equation.

5.3 HelmNet

In contrast with Neural Operator [2], *HelmNet* [47] learns the mapping only between the finite-dimensional Euclidian spaces by using Convolutional Neural Networks (CNN). Authors of *Helmnet* [47] pursue the identical objective as our work – solving the Helmholtz equation (Equation (2.2)). Helmnet is based on the U-net architecture [38], and it is structured as an iterative solver described by the following equation:

$$\begin{aligned} (\Delta u^{k+1}, h^{k+1}) &= f_\theta(u^k, e^k, h^k) \\ u^{k+1} &= u^k + \Delta u^{k+1}, \end{aligned} \tag{5.7}$$

where u denotes an unknown wave-field from Helmholtz equation (Equation (2.2)), h is a recurrent *belief state*, e_k represents the PDE residual (see Section 3.3). f_θ is a learnable differentiable function, implemented using a modified U-net [38]. The authors discovered that instead of passing the PDE parameters directly, incorporating these parameters into the residual term R (Equation (3.4)) eases the network's learning of the mapping from the parameters of a PDE a to a solution u. We omitted the spatial coordinate $\boldsymbol{x}$ in order to improve the clarity of the equation.

The training is guided using only a physics loss formed by the residual of the Helmholtz equation. The method applies gradient clipping at a value of 1 to avoid exploding gradients. The other distinctive feature of HelmNet is utilization of a replay buffer, which indirectly enables the model to be trained for unrolling for a large number of iterations.

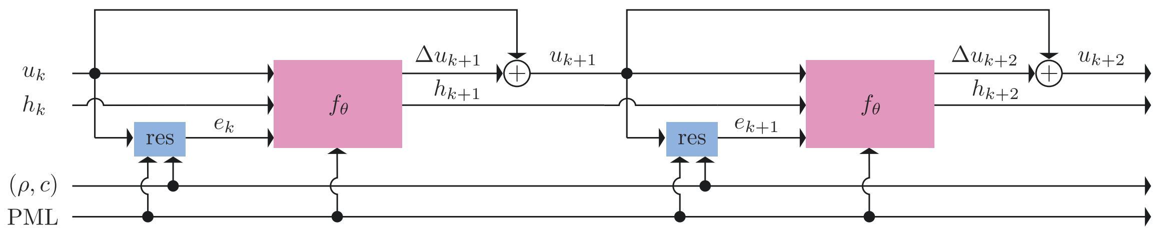


Figure 5.4: Scheme of an iterative solver proposed by Stanziola et al. [47]. A belief state h, residual e and wave-field prediction from previous iteration are passed as inputs to a function f_θ in each iteration. The function f_θ is realized using a modified version of U-Net, which is depicted in Figure 5.5. Adapted from Stanziola et al. [47].

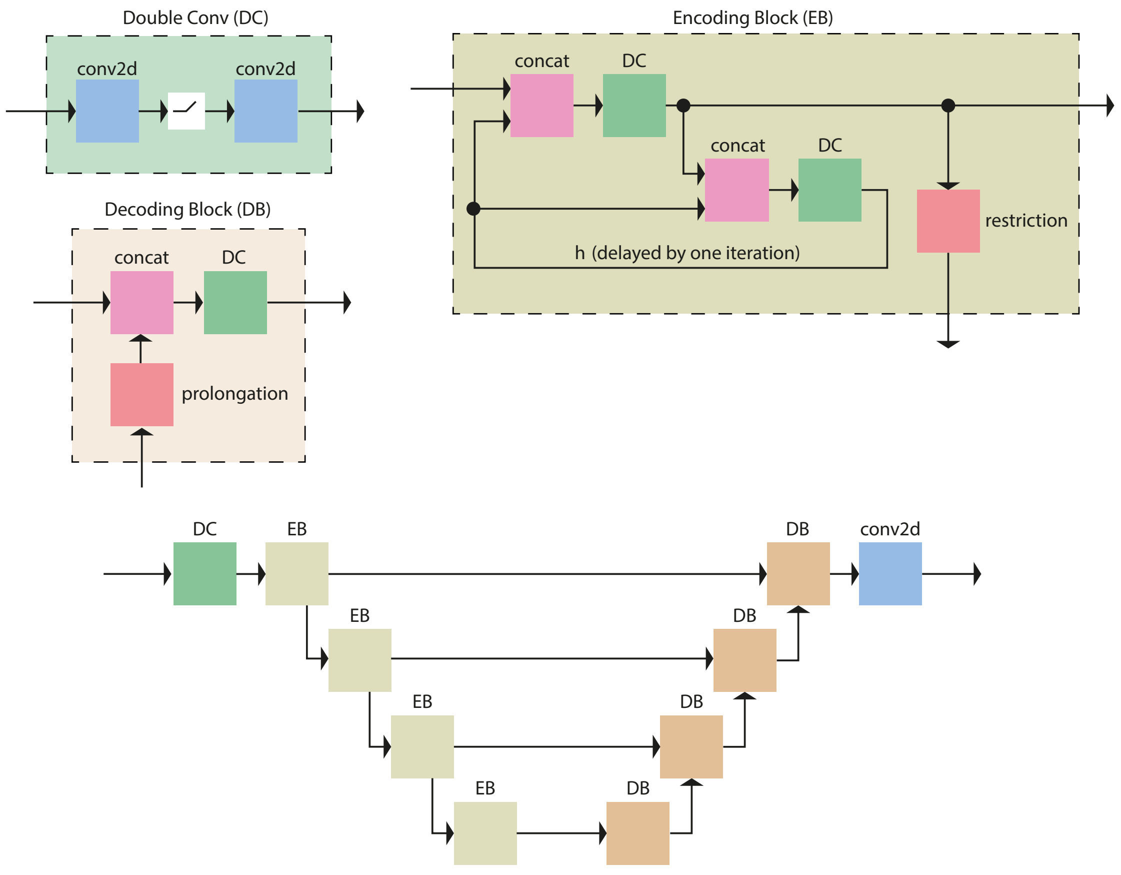


Figure 5.5: Scheme of a modified version of U-Net. Each decoding block consists of two 3×3 convolutional kernels and PReLu activation function [15]. Each convolution layer contains only 8 channels, resulting in a lightweight network with 47k parameters. Adapted from Stanziola et al. [47].

Chapter 6

Proposed Solution for Sound Propagation Approximation

To solve the Helmholtz equation, we propose an iterative solver based on the graph neural networks. The neural network architecture and the training procedure are described more extensively in Section 6.2 and Section 6.4. Unlike Helmnet [47], our solution employs supervised learning, thus a reference solution $u^*(\boldsymbol{x})$ is required. The following section provides a more detailed description of the used dataset.

6.1 Creating a Synthetic Dataset

We use the method proposed by Stanziola et al. [47] to generate synthetic skulls as sound speed maps. The shape of the idealized skull is created by summing circular harmonics with random amplitude and shape. The whole work considers normalized units $\omega = 1\,\mathrm{rad/s}$ and background speed of sound of $1\,\mathrm{m/s}$. As depicted in Figure 6.1, the skull thickeness ranges from 2 to 10 m with sound speed varying from 1.5 to 2 m/s. Stanziola et al. [47] generated coarser samples with 96×96 grid size with 1 m grid spacing. We opted for a higher resolution, which provides an option for later experiments with different resolutions by downsampling the samples. Hence we set the grid size to 384×384 with 0.25 m grid spacing. The source location is generated randomly for each sample, but each coordinate value is a multiple of four due to subsequent downsampling. By using these settings the samples can be downsampled by a factor of 2 and 4 effortlessly.

Unlike Helmnet [47], our model is trained using supervised learning. For that reason, each data sample requires a *ground truth* wavefield. Obtaining a closed-form solution for each data sample would be infeasible, thus, we used k-Wave [51] to generate numerical solutions for each sample as the ground truth. For each skull, three random wave source positions are generated. Due to the linearity of the Helmholtz equation, solutions can be *added* together to create a new, more complex solution with multiple sources, as shown in Figure 6.2. Training, validation, and test sets contain 24 000, 3 000, and 3 000 wavefields samples, respectively. The dataset is available in a public repository[1].

[1] https://sc-nas.fit.vutbr.cz:10443/xnguye16/ssw-dataset

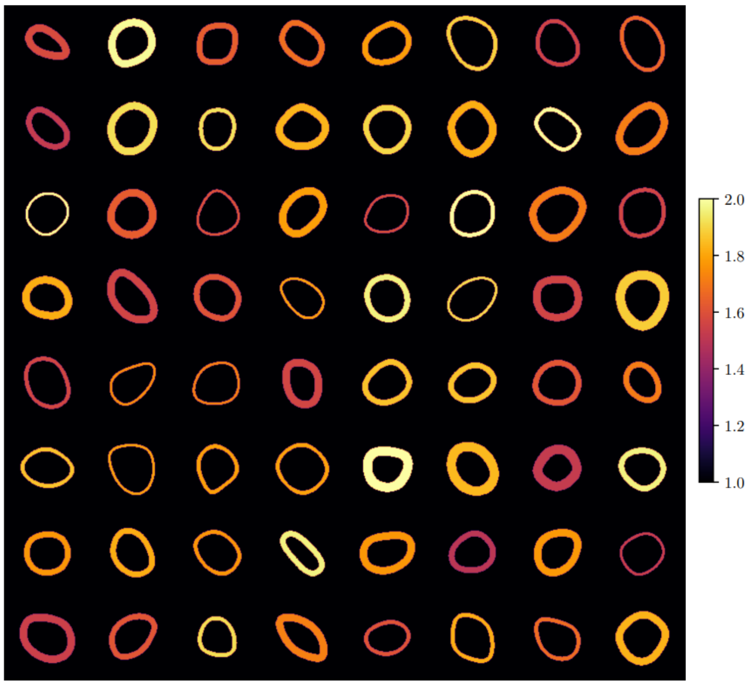


Figure 6.1: Synthetic skulls were generated using the method proposed by Stanziola et al. [47]. Summing several elliptical harmonics results in a shape similar to a skull. Each skull has a random sound speed between 1.5 and 2 m/s. Downloaded from Stanziola et al. [47].

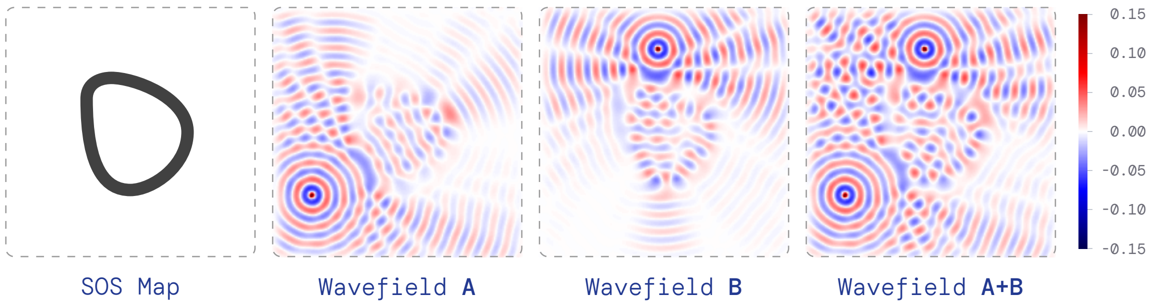


Figure 6.2: Synthetic skulls were generated by summing several elliptical harmonics [47]. Reference wavefields were computed using k-Wave [51] solver. Three wavefields were computed for each SOS map, each with a different source location. Wavefields can be added together to produce a new and more complex wavefield as a consequence of the Helmholtz equation linearity.

6.2 Network Architecture Solving the Helmholtz Equation

The proposed solution's network architecture is depicted in Figure 6.3. The network is based on the graph neural network formalized using the Graph Network framework [4] with the encode-process-decode structure [34, 40], since it should handle data sampled on an irregular grid to a certain degree.

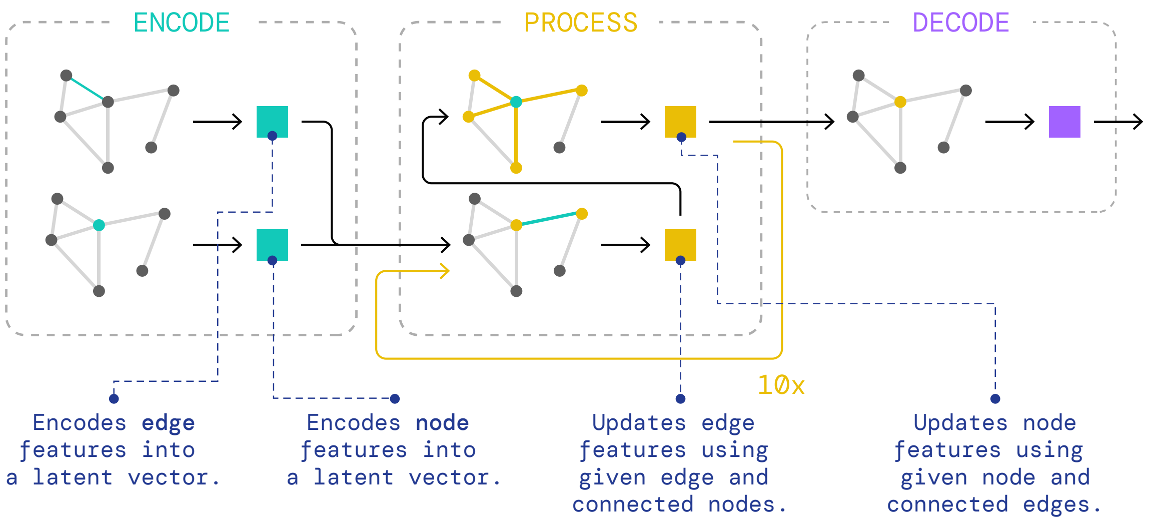

ENCODE
PROCESS
DECODE
10x
Encodes edge features into a latent vector.
Encodes node features into a latent vector.
Updates edge features using given edge and connected nodes.
Updates node features using given node and connected edges.

Figure 6.3: The network architecture consists of three parts – encode, process, and decode. In the encoder stage, all features are respectively transformed into latent vectors. A message-passing mechanism is executed in the process stage, where information from neighboring nodes is aggregated to produce updated features for the target node and edges. Lastly, the decoder transforms data from latent space to output space.

All parts of the process block ϕ^e, ϕ^v, both encoders ϵ^v, ϵ^e and decoder δ^v are implemented using a two-layer MLP with ReLU activation function [1] and a residual connection. We omit the global features, therefore ϕ^u function is not used. The latent and output size is 64, except δ^v, where the output size equals to the model's output size. The MLP is depicted in Figure 6.4. Due to residual connections, our method defined by Equation (6.2) can be viewed as:

$$u^{k+1} = u^k + g_\theta(v, e, u^k, R^k), \tag{6.1}$$

where g_θ would be the MLPs without the residual connection. After this modification it resembles to the Euler forward method. Furthermore, Lu et al. [28] present that each residual block can be perceived as one step of the Euler method:

$$u^{k+1} = hf(u^k), \tag{6.2}$$

where h represents the step size. Additionally, for the same reasons described in Section 5.2.1, we opted for the averaging aggregation function.

All parameters from the Helmholtz equation, such as SOS map, source map, or spatial coordinate, are used as an input alongside with the prediction and residual from a previous iteration. All nodes are sampled on a uniformly spaced grid, where all nodes within a radius $r = 0.02$ are connected.

Encode Encoder consists of two separate MLPs ϵ^v, ϵ^e for nodes and edges, respectively. These encoders transform input features (processess each node and each edge separately) into latent vector of size 64. Each node feature v_i with a spatial position $\boldsymbol{x}_i$ is composed of SOS map$c(\boldsymbol{x}_i)$, wave source distribution $\rho(\boldsymbol{x}_i)$ and PML absorption term $\sigma(\boldsymbol{x}_i)$. The distance information $|x_ij|$ and $x_i - x_j$ between connected nodes is encoded in the edges.

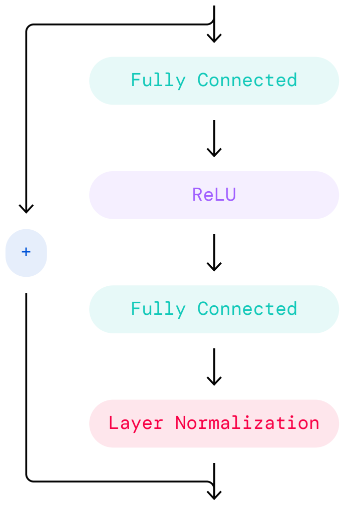


Figure 6.4: The main building block for the whole network is based on the two-layer MLP with a ReLU activation function and a residual connection.

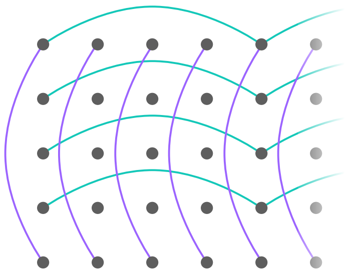


Figure 6.5: Hop connections connect every n-th node. In this particular case $n = 4$. Hop connections $n \in \{3, 5, 10\}$ are used in the proposed solution.

The SOS map is sampled using 96×96 regular grid. Thus the data are downsampled using a factor of 4. Connections between the nodes are created using radius $r = 0.02$, node coordinates are normalized to range $[0, 1]$.

Additionally, we use connections between every n-th node (hop connection), increasing the model's receptive field (Figure 6.5). The hop connections can be seen as a multi-resolution graph [27, 35] and they should enable the network to predict larger area in less unroll iterations. Our model contains hop connections between every 3-th, 5-th and 10-th node.

Process Processor unit is a derivation of a processor defined by Pfaff et al. [34]. It consists of P identical blocks – Graph Block [4]. Each block contains a separate set of weights.

As depicted in Figure 6.6, the Graph Block consists of three parts: ϕ^e, ϕ^v and $\rho^{e \to v}$. Function ϕ^e is implemented using MLP. It encodes data of an edge and nodes connected to it into a new edge feature. Then edges connected to a node are aggregated using function $\rho^{e \to v}$. In our work, we use the averaging function as an aggregation function. Aggregated features are transformed into an updated version of node features using a function ϕ^v. The previously mentioned function ϕ^v is also implemented using MLP. The described functionality is also referred to as *message passing* [12]. One Graph Block corresponds to one message passing. More Graph Blocks result in a larger receptive field of the network.

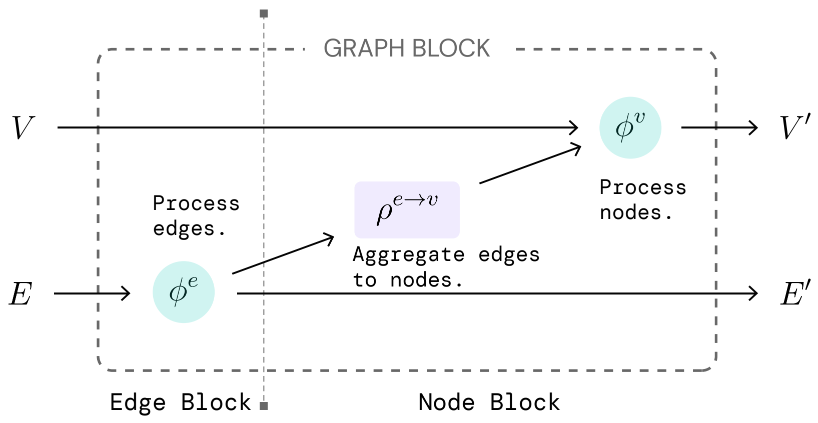


Figure 6.6: Graph block consists of two parts: edge block and node block. Edge block uses features of a given edge and its nodes to update edge's features. Node block updates a node features based on all the nodes connected to the target node. Functions ϕ^e, ϕ^v are implemented using MLPs, aggregation function $\rho^{e\to v}$ is using the *averaging* function.

Decode Last stage of the network δ^v decodes node features from latent space into the scaled wavefield prediction βu_{k+1}, where $\beta = 500$ is a scaling term (see Section 6.4).

6.3 Approximating the Equation Residual

Due to the explicit residual calculation, an approximation of the Laplacian has to be computed. Equation (2.2) is transformed into the residual in the following way:

$$R = \left[\nabla^2 + \left(\frac{\omega}{c(\boldsymbol{x})}\right)^2\right] u(\boldsymbol{x}) - \rho(\boldsymbol{x}), \tag{6.3}$$

Based on the conducted experiments (Section 6.3), we use a pseudo-spectral method, which is restricted to the regular grid, to approximate the derivatives.

6.3.1 Pseudo-spectral Method

Stanziola et al. [47] approximate first-order derivatives using FFT-based method (see Equation (6.4)). They are then composed into the Laplacian. The Laplacian can not be computed directly as a result of using PML as the boundary condition (Equation (2.4)).

$$f_\eta(\eta) \approx \mathcal{F}^{-1}\left(\mathcal{F}\left(ik_\eta f(\eta)\right)\right), \tag{6.4}$$

where $\eta = x_1, x_2, \ldots, x_n$ denotes a spatial dimension, f_η stands for a gradient of the function f w.r.t. η, k_η represents wavenumbers in a given direction, $\mathcal{F}$, $\mathcal{F}^{-1}$ are Fourier transform and its inverse, respectively.

6.3.2 Average Gradient on Star

The *Average Gradient on Star* (AGS) and *Per-Cell linear Estimation* (PCE) methods unlike the FFT-base method can be utilized on an irregular grid. In a 2D mesh, for a triangle t

with vertices v_i, v_j, v_k, PCE is defined as follows [30]:

$$\nabla f_t \approx (f_j - f_i)\frac{(v_i - v_k)^{\perp}}{2A_t} + (f_k - f_i)\frac{(v_j - v_i)^{\perp}}{2A_t}, \tag{6.5}$$

where A_t is an area of the triangle t, f_i is a value of a vertex v_i and $(e)^{\perp}$ denotes a perpendicular vector to the vector e.

Average Gradient on Star (AGS) averages all gradients from neighboring vertices to compute per-vertex gradients:

$$\nabla f(v) \approx \frac{1}{\sum_{i \in \mathcal{N}(v)} A_i} \sum_{i \in \mathcal{N}(v)} A_i \nabla_{PCE} f_i \tag{6.6}$$

where $\nabla_{PCE} f_i$ is defined by Equation (6.5) and $\mathcal{N}(v)$ is set of vertices connected to the vertex v.

6.4 Training the Iterative Model

As mentioned before, the model is trained using **supervised** learning. Any addition of physics terms resulted in an unstable training and constrained the model's ability to learn more than a small neighborhood around the source.

6.4.1 Loss Function

Naturally, mean-squared error is applied as a loss function

$$L = \frac{1}{N} \sum ||u^T - \beta u^*||_2^2, \tag{6.7}$$

where $\beta = 500$ is a scaling term, N is the number of graph samples. All samples are downscaled to 96×96. Symbol u^T is the predicted solution after T unroll iterations. In other words, loss function only takes into account a prediction from the last unroll iteration.

The value of scaling term β was selected empirically. It can be seen as scaling the source amplitude by β. Usage of the scaling term changes the magnitude as well as the direction of loss function gradients. We hypothesize that the scaling term is essential mainly due to the numerical stability, since the wavefield values are too small when the wavefield is not scaled.

6.4.2 Training phases

Although the model can be trained end-to-end, the two-phased approach requires only half of the computational time (Table 6.1). The first phase involves training the network with 3 unroll iterations for ≈ 70 k optimization iterations. As illustrated in Figure 6.7, the network is then fine-tuned for ≈ 10 k optimization steps with 8 unroll iterations. Both phases are trained using the Adam optimizer [20] with a learning rate $\alpha = 3e - 5$. The network is trained on 8 A100 40GB GPUs using Pytorch Lightning[2] distributed data-parallel (DDP) accelerator.

[2]https://www.pytorchlightning.ai/

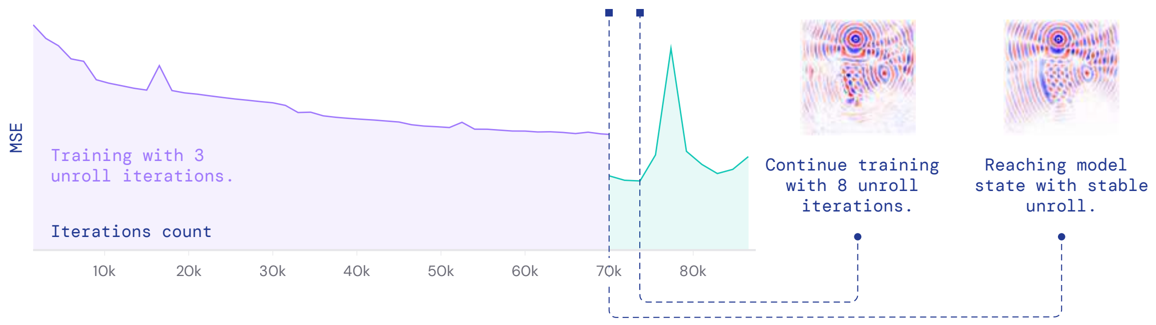


Figure 6.7: Even though end-to-end training requires the same number of optimization steps as the two-phased approach, it requires more time, since the duration of one optimization step is longer. Thus we opted for training with two phases. The first part of training comprises 70k optimization iterations with three unroll iterations. Afterward, the network is fine-tuned for less than 10k optimization iterations with 8 unroll iterations. Here the network learns to stabilize unrolling. The displayed error curve was calculated on the validation set. Thus, the peak in the second phase of the training signalizes overfitting.

Table 6.1: Two phased training significantly reduces the computational resources compared to the end-to-end approach. The training time was measured on a computer with 8 A100 40GB.

	End-to-end	**2 phases**
MSE	14.301	14.577
Duration [h]	21	10
Optim. steps	$\approx$ 80k	

6.4.3 Mini-batching

Instead of computing the gradients based on the whole dataset, the gradients are approximated from a mini-batch. Based on the assumption that the data distribution of the mini-batch is close to the data distribution of the whole dataset, the gradients should be similar as well.

Mini-batch from a set of images is typically formed by concatenating the same-sized images along a new dimension. However, if the graphs are mini-batched the same way as the images, it would result in high memory consumption. This is due to the padding, since all graphs can have a different shape, nodes count, etc. For that reason, we use mini-batching implemented by PyG[3].

Formation of the mini-batch graph is described by Equation (6.8). All adjacency matrices $\boldsymbol{A}_i$ are stacked diagonally resulting in the one large graph containing multiple isolated graphs. Node features $\boldsymbol{v}_i$ are concatenated along the node dimension (see Figure 6.8).

[3]Pytorch Geometric - https://pytorch-geometric.readthedocs.io/en/latest/index.html

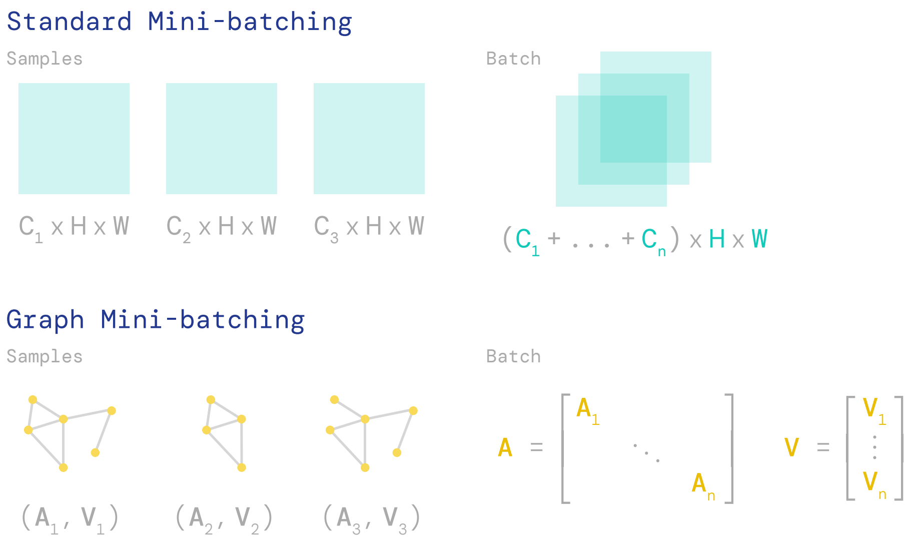


Figure 6.8: Concatenating graphs along a new dimension would require padding – setting all graphs to the same size. Nonetheless, memory overhead caused by padding can be avoided if all graphs are formed into one large graph containing multiple isolated graphs.

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_1 & & \\ & \ddots & \\ & & \boldsymbol{A}_n \end{bmatrix}, \qquad \boldsymbol{V} = \begin{bmatrix} \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{v}_n \end{bmatrix} \tag{6.8}$$

Since all the graph samples are isolated graphs, the message can not be exchanged between two nodes that do not belong to the same graph after applying the message-passing operators.

6.4.4 Implementation of the Multi-GPU Training

Since a graph is one of the most general data structures, it requires more data to describe it. For that reason, we struggled with memory issues, allowing us to set the batch size only to 2 on a 40 GB GPU.

The problem of having a small batch size can be avoided by employing multi-GPU training. The multi-GPU training is achieved using Pytorch Lightning implementation of *Distributed Data-parallel* training (DDP). DDP feeds a mini-batch to each GPU, and then each GPU computes the gradients w.r.t. the mini-batch. Afterwards, gradients are averaged across all the GPUs, resulting in a effective batch size equal to $N \times$ batch size, where N corresponds to the number of GPUs (see Figure 6.9). Since our model was trained on a node with 8 A100 40 GB, the effective batch size was 16.

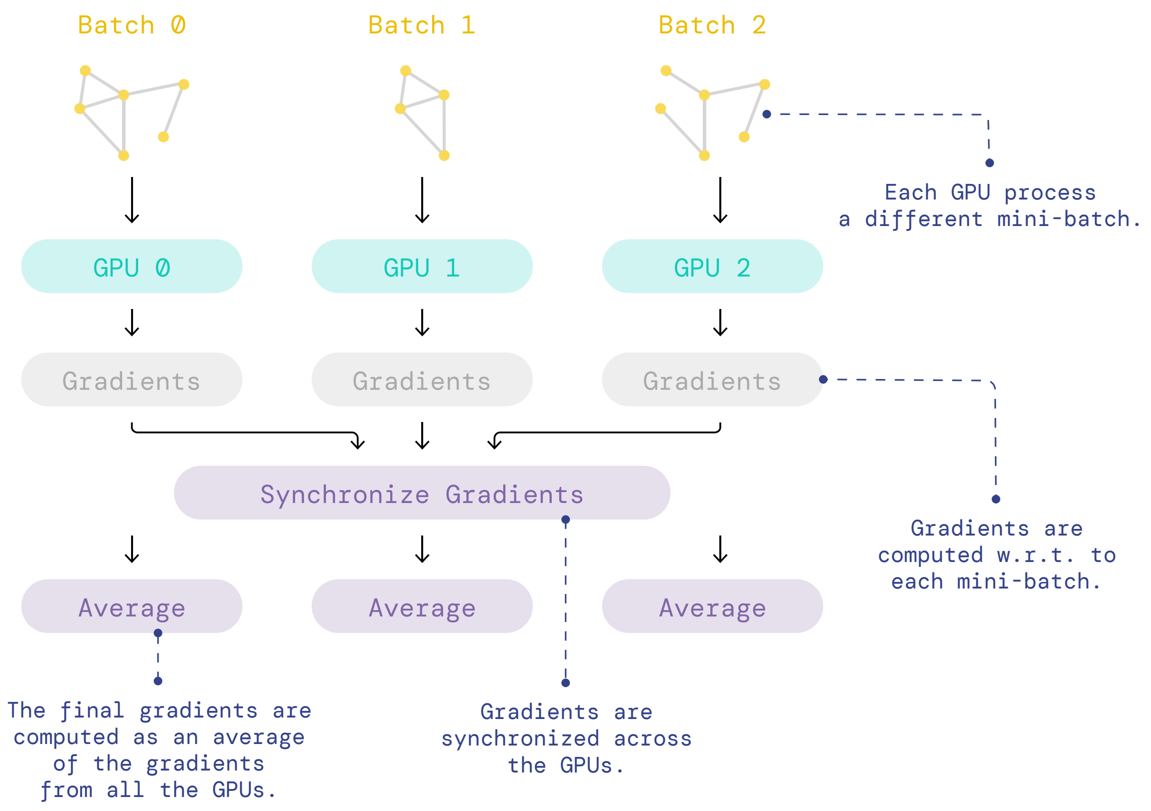


Figure 6.9: In the distributed data-parallel training (DDP), each GPU receives a mini-batch and computes gradients w.r.t. the given mini-batch. Afterwards, gradients from all the GPUs are synchronized and averaged to produce the final gradients.

Chapter 7

Evaluation

To evaluate the performance of the trained model, we put our solution under multiple tests. Firstly, the model's generalization ability is demonstrated using samples outside of the training and validation distribution, including more than 5-times larger domain. The reference solutions are computed using k-Wave Toolbox [51]. Additionally, our model is compared to Helmnet [47], where the MSE is used as an evaluation metric:

$$\text{MSE} = \frac{1}{N}||u - \beta u^*||_2^2, \tag{7.1}$$

where N is the nodes count, $\beta = 500$ is the scaling term (see Section 6.4.1), u and u^* are predicted and reference wavefield, respectively. We scale the predicted wavefield u by β if the tested system produces an unscaled wavefield u^t – wavefield with a source magnitude set to one.

7.1 Testing the Model's Generalization in Various Domains

Our model has only been trained on the synthetic skulls, therefore a square SOS map is out of the training distribution. Nonetheless, our model is able to predict SOS maps, which it has never seen, indicating that the model learned to solve the Helmholtz equation, as illustrated in Figure 7.1. Furthermore, Figure 7.2 depicts that the model learned the interaction between waves from multiple wave sources, although it was trained on samples with a single wave source.

7.1.1 Unroll Stability

For most samples with 96×96 grid size, 8 unroll iterations are sufficient to predict the wavefield for the whole computational domain. Nonetheless, more complex wavefields as well as larger domains require more than 8 unroll iterations. This happens due to more wave reflections that need to be simulated. To test whether our model is capable of predicting wavefields even in larger domains, we measured unroll stability for 128 iterations in a 96×96 domain. As depicted in Figure 7.3, the error does not rapidly diverge until approximately 70-th iteration.

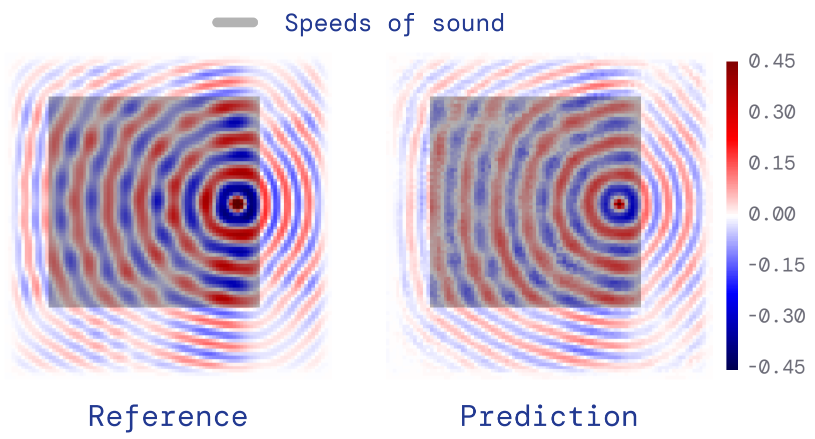


Figure 7.1: The model is able to predict wavefield with the square heterogenity in the middle, although it was only trained on synthetic skulls.

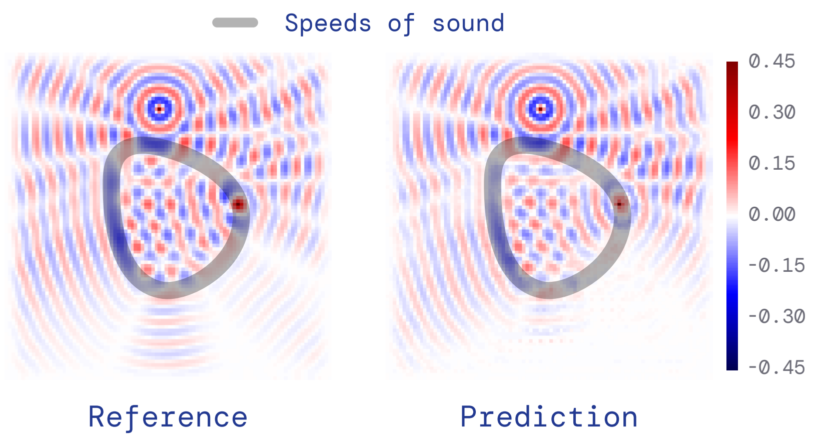


Figure 7.2: The model is able to predict wavefields with multiple wave sources, despite the fact it was trained with single-source samples.

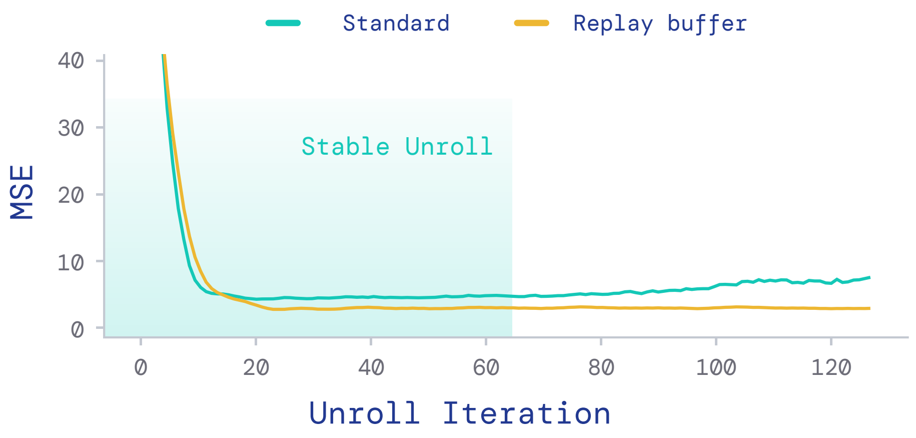


Figure 7.3: Although, the model is only trained with 8 unroll iterations, the error does not greatly diverge until the 70-th iteration. We use a replay buffer to mitigate the unroll divergence. The unroll stability was tested in a 96×96 domain.

We utilized a replay buffer [19] to reduce the unroll divergence. The replay buffer contains 400 quaternions (u_k, R_k, e, v). The iteration index k is randomly initialized to an integer from 0 to the half of the maximum iteration count Q – we set $Q = 256$. After training the model with the replay buffer, the error does not diverge (Figure 7.3). Thus, the model learned to stop, when the wavefield is solved.

7.1.2 Larger domains

The model is not invariant to the domain size, as a result of its dependence on the nodes coordinates $x_i \in [0, 1]^D$ (see Section 6.2). We firstly train the model in the first phase to create a domain-invariant model as described in Section 6.4. The input features corresponding to the nodes coordinates $\boldsymbol{x}$ are set to zero during the second phase. With this modification, we are able to train a size-invariant model, where the model predicts the wavefield in a 512×512 domain, as shown in Figure 7.4.

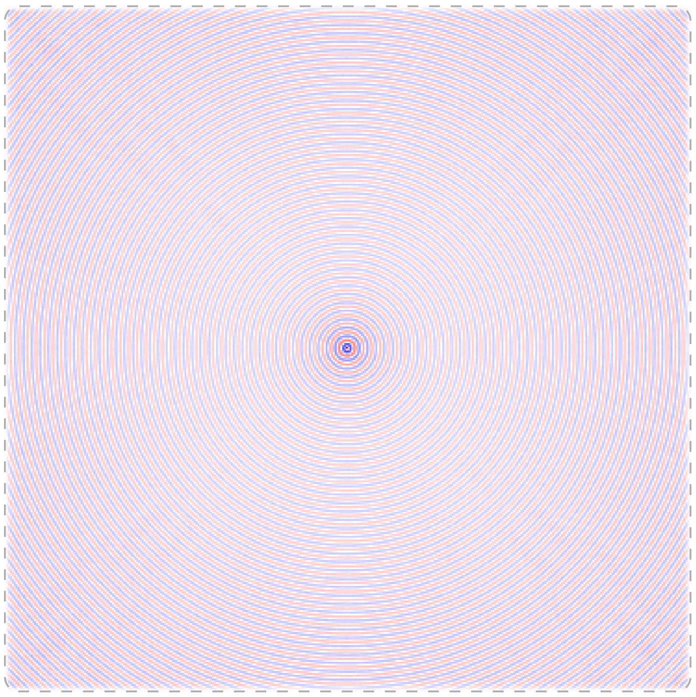


Figure 7.4: Our model is able to perform inference even on larger domains 512×512. However, during fine-tuning, the feature of each node corresponding to its position is set to zero. If the position is set to zero before the first phase of training, the model is not able to converge to a state of stable unrolling.

The replay buffer fails to produce stable unrolling in the larger domain as depicted in Figure 7.5, despite the fact it significantly stabilizes the unroll iterations (Figure 7.3) in a 96×96 domain. Small errors start to appear at the corners of the domain and are later amplified and propagated by the next unroll iterations.

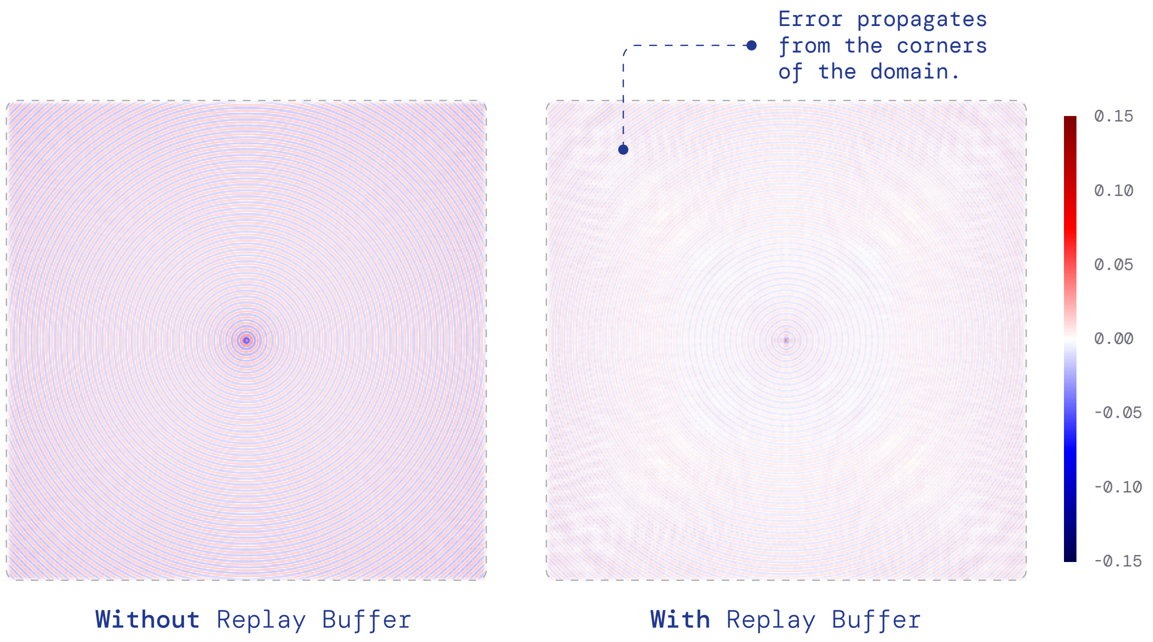
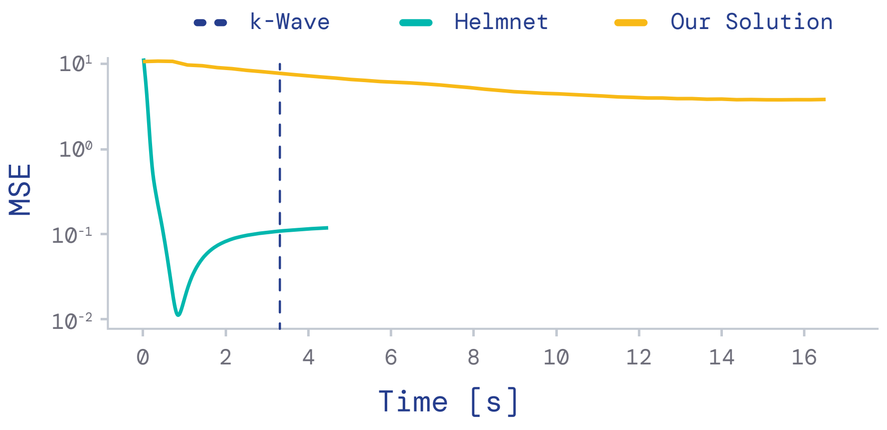


Figure 7.5: Although, replay buffer stabilizes the unroll in the 96×96, the network trained without the replay buffer produces more stable unroll. The model trained with the replay buffer starts to produce errors at the corners of the domain, which are amplified by the next unroll iterations.



p

Figure 7.6: Our model is notably slower than Helmnet and k-Wave and is not able to reach error as low as Helmnet. We assume that the speed is slower because the graph contains more data to be processed, despite containing the same amount of information as the images. Additionally, we did not optimize the residual calculation in any way. The experiment was conducted on a machine with the Nvidia A100 GPU.

7.1.3 Comparison with Helmnet

The accuracy of Helmnet [47] is remarkably higher than the accuracy of our solution, as depicted in Figure 7.7. We believe that the higher error of our model is caused by fine-tuning the training with the replay buffer. Thus, our model is not trained with enough unroll iterations to predict the wavefield with such low error. Our assumption is supported by Figure 7.6, where our model reaches a certain level of error, and stops to improve the predicted wavefield. Because the training with the replay buffer was done in later stages of our work, there was not enough time to tune the hyperparameters of the replay buffer.

Due to a larger amount of data to be processed (nodes and edges), our model is significantly slower than Helmnet and k-Wave. Thus, failing at one of the main aspects of the neural solvers. Nevertheless, our work mainly served to prove a concept, and its implementation is not optimized in any way.

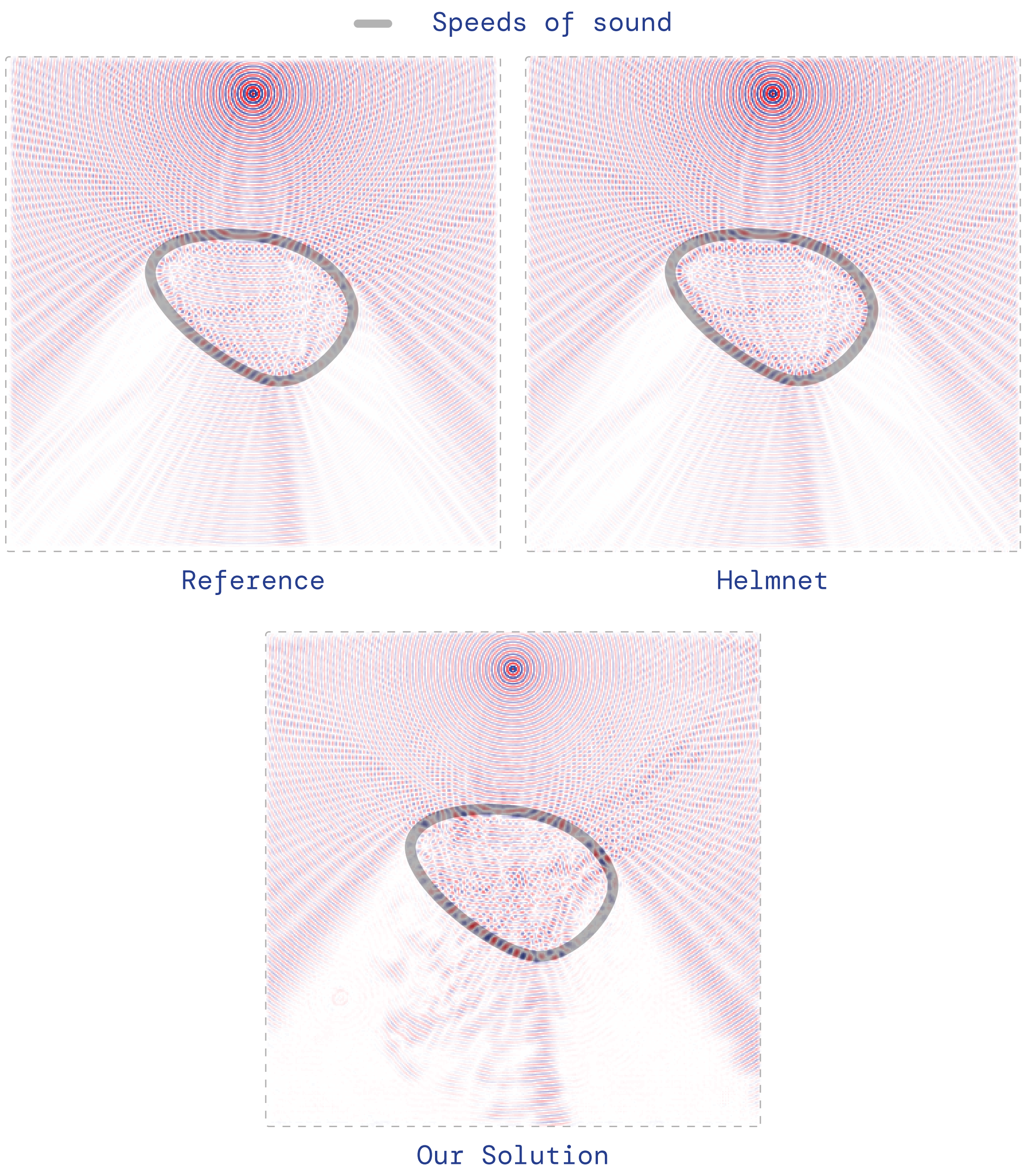


Figure 7.7: Helmnet produces a significantly more accurate wavefield than our solution. Nevertheless, our method still produces produces an adequate prediction.

7.2 Gradient Approximators for Residual Calculation

The FFT-based approximation of the residual is more accurate than the AGS approximation. Although the AGS is able to approximate the shape of the gradient, its magnitude error is significantly larger. The proposed model is sensitive to directional and magnitude error in the Laplacian due to explicitly calculating the residual. As illustrated in Figure 7.8, with more significant residual error, the model can only predict the wavefields to a certain distance from the source.

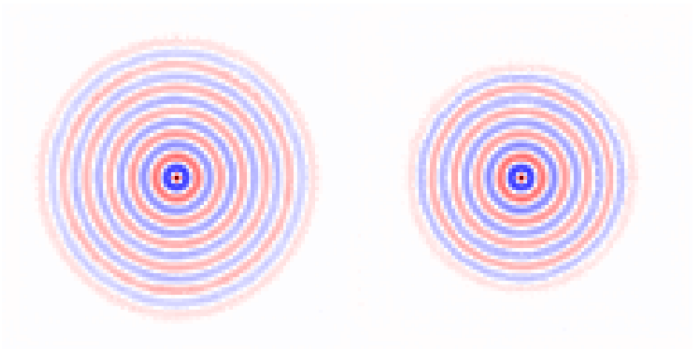


Figure 7.8: Despite having only three unroll iterations, the model with the FFT-based derivative approximation is able to predict significantly larger area than the model with the AGS approximation. When using AGS, the predicted area does not noticeably grow with more unroll iterations.

7.3 Predicting Irregularly Sampled Data

To alleviate the constraint of using data sampled on a regular grid, we experimented with data sampled on different grids, as depicted in Figure 7.9. Using other than the FFT-based derivative approximators is not feasible, as shown in Section 6.3 experiments. Thus, we opt for computing the Laplacian utilizing FFT. Although the Fourier transform is defined on graphs [41], it is more computationally expensive than the regular FFT, which makes them impractical.

In order to use FFT with irregular grids, we utilize a linear interpolator. Laplacian of the irregularly sampled data is retrieved using interpolation of regularly sampled data. Residual then can be computed the same way as with a regular grid.

Regular Grid Samples are sampled on a regular grid – they are evenly spaced.

Random Grid Samples coordinates are produced using a uniform random generator. As illustrated in Figure 7.9, sampling with a random generator fails to produce evenly spaced samples – producing samples too close to each other resulting in numerical instabilities.

Offset Regular Grid To create data sampled on a non-regular grid with evenly spaced samples, we perturb data sampled on a regular grid. Given a grid (96×96), the sample

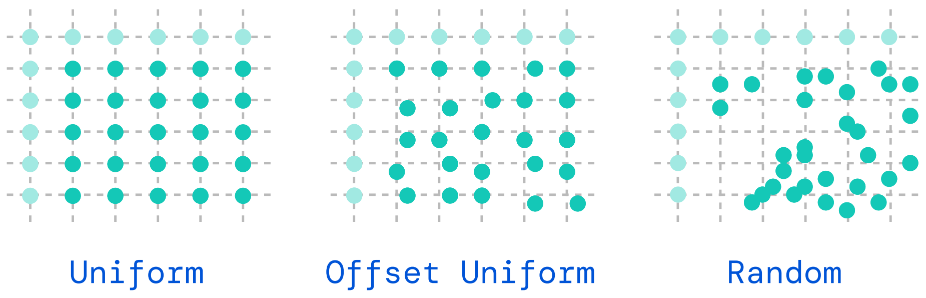


Figure 7.9: We utilize FFT-based derivative approximation to compute the Laplacian. Thus in the proposed solution, we use a regular grid. We attempted to lift the restriction of using the regular grid by several experiments described in Section 7.3.

coordinates are perturbed using the following method:

$$\hat{\boldsymbol{x}} = \boldsymbol{x} + \mathcal{U}(-\epsilon, \epsilon), \tag{7.2}$$

where $\boldsymbol{x}$ is the sample coordinates and $\mathcal{U}$ stands for uniform distribution. The perturbation $\epsilon = 0.0026$ is calculated as a 25% perturbation in a 96×96 grid ($\epsilon = 0.25 * 1/96$).

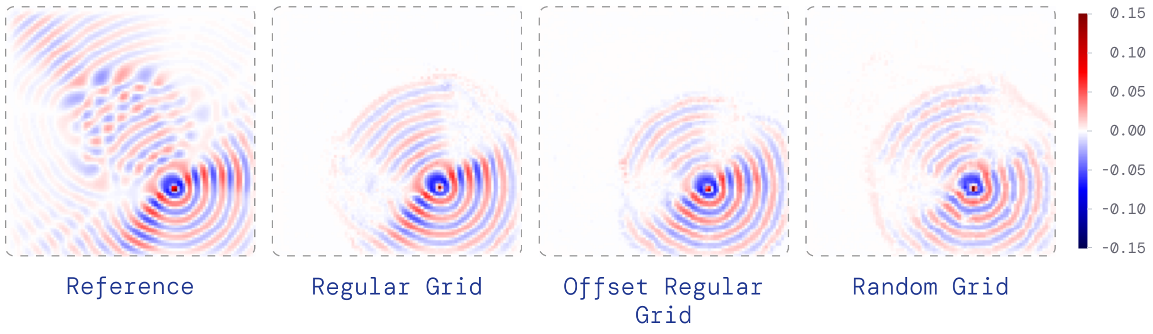


Figure 7.10: Proposed model produces the best results with data sampled on the regular grid. However, data sampled on the offset regular grid still produces usable wavefields. However, randomly sampled data suffer from an interpolation error, due to unevenly sampled domain. All models are evaluated with only 3 unroll iterations due to limited access to the supercomputer.

Results Any irregularity in the data sampling grid results in model's incapability of stable unrolling. Figure 7.10 does not sufficiently depict the difference between the wavefields predicted on different grids. In Table 7.1 the difference between the wavefields predicted on the irregular grid and regular grid is more apparent.

Table 7.1: The model is not capable of stable unrolling on the irregular grid. The irregular grid increases the model's error substantially. Models are evaluated only with 3 unroll iterations due to the limited access to the computational resources.

	Uniform	**Offset Uni.**	**Random**
MSE	**24.8**	27.1	27.9
Optim. steps	$\approx 63k$	$\approx 70k$	$\approx 120k$

7.4 Achieving Super-resolution

To test the model's ability to predict wavefield for the upsampled points, we insert data sampled on the irregular grid into the uniformly sampled data. This is done in order to increase resolution in certain areas of the computational domain (see Figure 7.11).

During the training, we simulate super-resolution by generating 500 upsampled points using a uniform random generator. The upsampled data are then concatenated with the uniformly sampled data. Since the FFT is well defined only on the regular grid, the residual can only be computed for uniformly sampled data. Thus, we obtain the residual of random points by interpolating the residual from uniformly sampled neighbours.

Results To evaluate the upsampled predictions produced by our model, the MSE is computed only from the upsampled points. As shown in Table 7.2, our model offers lower error than the linear interpolation method. Despite that, the MSE of the upsampled points (MSE = 20.88) is substantially higher in comparison to MSE of uniformly sampled points (MSE = 13.65). We hypothesize that the higher error of the upsampled points is caused by their irregularity, as shown in experiments described in Section 7.3.

Table 7.2: As a baseline solution, we used a linear interpolation of the neighbouring nodes. Compared with our model, the MSE error of the upsampled points is notably lower. Nonetheless, the MSE of the predicted uniform samples is 13.65, which is significantly lower.

	Interpolated	**Predicted**
MSE	22.51	**20.88**

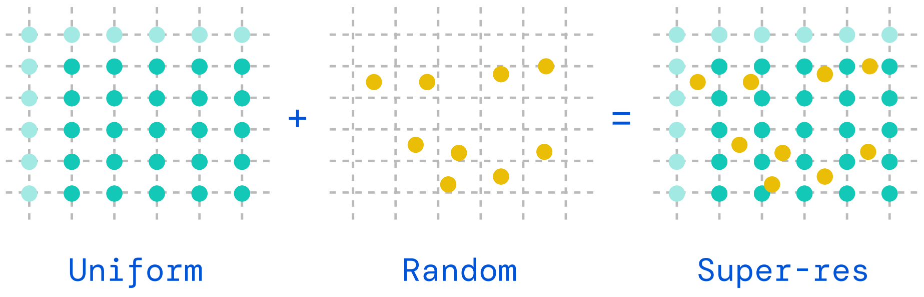


Figure 7.11: 500 random points are concatenated to the uniformly sampled data to train the network to perform prediction for upsampled data. The residual for the upsampled data is computed by interpolating residual from the uniformly sampled neighbours.

7.5 Operating the Model with the Downsampled Laplacian

As Ayala et al. [3] state, the state-of-the-art parallel FFT implementations are still bounded by the communication bottleneck. For that reason, the scalability of the FFT is rather limited. Instead of using a more efficient implementation of FFT, we opted to avoid the problem altogether.

Our solution, as well as Helmnet [47], uses the FFT multiple times in the residual approximation. In a larger domain or a 3D space, the scalability issues of the FFT might bottleneck the neural solver. For that reason, we test whether the neural solver is capable of solving the wavefield with *downsampled* Laplacian.

Given the wavefield's angular velocity$\omega = 1$ and grid spacing $\mathrm{dx} = 1\,\mathrm{m}$ described in Section 6.1, the number of *points per wavelength* (PPW) is 6.283. Therefore, a wavefield with $\mathrm{dx} = 2\,\mathrm{m}$ will statisfy the Nyquist-Shannon sampling theorem [44], because the points per wavelength would be 3.141. Purely based on the sampling theorem, the neural solver should work with a downsampled Laplacian. Nonetheless, **none** of the traditional solvers work reasonably with so few PPW, and we are not aware of any neural solver capable of working with these extreme settings.

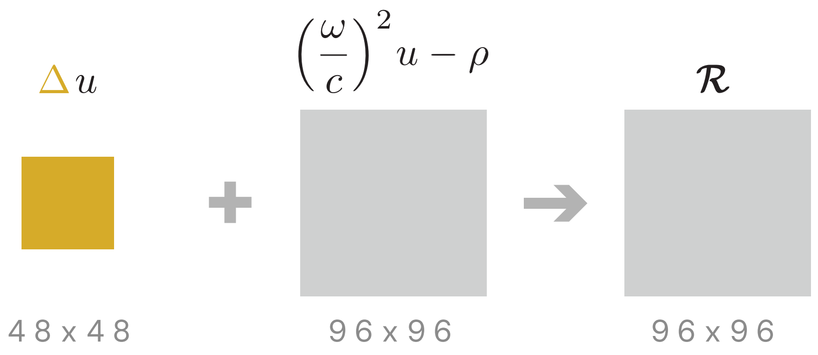


Figure 7.12: To overcome the scalibility issues of FFT in the residual calculation, we propose to downsample the input of the pseudo-spectral method. The output of the pseudo-spectral method is upsampled using bicubic interpolation to match the resolution of other terms of the residual.

Downsampling the Laplacian. Downsampling the Laplace operator (Equation (2.7)) is described by the following equation:

$$\frac{1}{\gamma_\eta}\frac{\partial}{\partial\eta} \to \frac{1}{\gamma_\eta}\mathrm{Up}\left(\frac{\partial}{\partial\eta}\mathrm{Down}\left(\cdot\right)\right), \tag{7.3}$$

where Up$(\cdot)$ and Down$(\cdot)$ stand for upsample and downsample using bicubic interpolation, respectively. The sample size is 96×96, but the gradient is computed in 48×48 resolution.

Results. Our method can correctly predict the wavefield even with the downsampled Laplacian as illustrated by Figure 7.3. PPW $= 3.14$ satisfies the sampling theorem, however **none** of the traditional solvers produce acceptable results. Furthermore, we are unaware of any neural solver working with these settings.

As expected, the FFT execution time is significantly lower if the input of the FFT is downsampled. The input and the output have to be resampled, yet the total time spent computing the gradient remains lower than the computation without downsampled Laplacian (see Table 7.3).

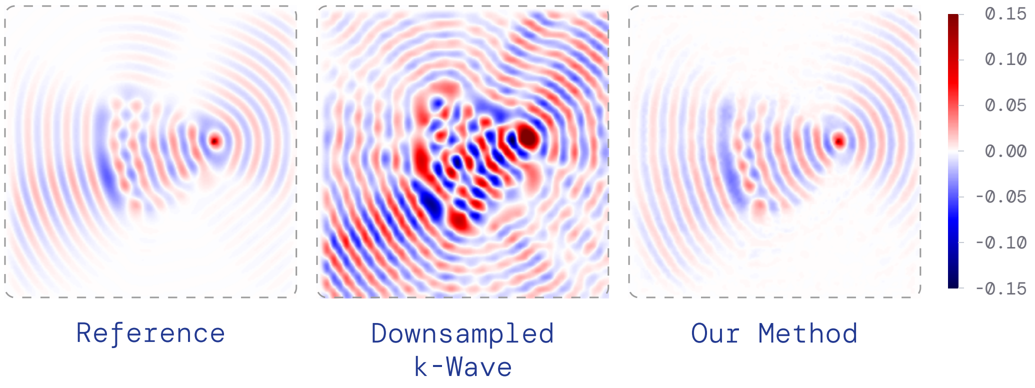


Figure 7.13: Unlike k-Wave, our model can operate with the downsampled Laplacian, where PPW = 3.

Table 7.3: Downsampling the Laplacian resulted a minor increase in error. Nonetheless, the large MSE of the downsampled k-Wave proves the traditional methods' incapability to work with sparser discretization. The execution times were measured on the NVIDIA Quadro RTX 6000 GPU.

	Baseline	**Downsampled Residual**	**Downsampled k-Wave**
MSE	**14.58**	14.77	560.89
FFT [ms]	0.704	**0.252**	—
Spectral Derivative [ms]	2.762	**0.803**	—

7.6 Effect of Pruning on the Model's Performance

One of the primary motivations behind the neural solvers is to reduce the time required to obtain a PDE solution. Due to this reason, we analyze pruning the network's weights, which might improve the network's speed. However, Frankle and Carbin [10] formulate the *Lottery Ticket Hypothesis* (LTH), stating that it can increase the model's accuracy as well. Frankle and Carbin [10] use the *Iterative Magnitude Pruning* (IMP) to find Winning Tickets – pruned models performing better than the unpruned ones. To formulate the sparsity of the network, P_m denotes the percentage of unpruned weights – $P_m = 75\,\%$, when $25\,\%$ of weights are pruned.

Our proposed method differs by employing the two-phased training. Pruning can be applied at the beginning of the first or the second phase. We followed the LTH methodology so we reset the model weights after each pruning. Three different scenarios were tested:

Training The pruning is applied before the whole training starts, which is the beginning of the first training phase – training with 3 unroll iterations.

Fine-tuning As the name suggests, the model is pruned at the beginning of the second training phase (fine-tunning) – training with 8 unroll iterations. The model weights are reset to the state at the beginning of the second training phase, meaning the weights are not set to initialization.

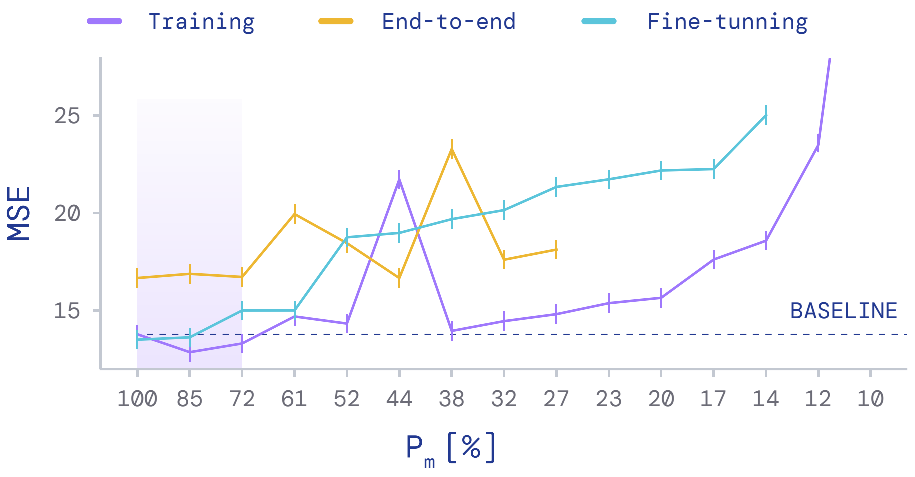


Figure 7.14: Any pruned network with better performance than the best-unpruned network (baseline), is referred to as *winning ticket*. Winning tickets only occur when the network is pruned before the first training phase, where any network with $P_m \geq 72\,\%$ is a winning ticket (purple area). P_m denotes the percentage of unpruned weights displayed on the logarithmic scale.

End-to-end End-to-end training with 8 unroll iterations was tested. The network is pruned at the beginning of the training – same as Frankle and Carbin [10].

Results As Figure 7.14 shows, the winning ticket can only be obtained with the two-phased training. We believe that the *fine-tuning* approach failed to produce any winning tickets due to not resetting the weights to the ones from the initialization. *End-to-end* approach failed because it requires significantly more optimization steps than the two-phased approach (see Section 6.4). This is leaving the two-phased training approach most likely to produce any winning tickets. The winning tickets were only obtained with low levels of pruning $P_m \geq 72\,\%$, meaning the network might not be over-parametrized. Iterative pruning can be used to increase the model's accuracy, even though the winning tickets are not sufficiently sparse to utilize sparse multiplication or improve inference speed. Despite that, there are multiple methods for stabilizing the LTH [29, 11]. We did not conduct any related experiments, due to the expensive nature of the LTH experiments.

7.7 Ablation Study of the Proposed Model

This section covers additional tests and reasonings behind the proposed solution's design choices. Firstly, the significance of the hop connections is tested. Then we test the effect of a training noise on the model performance. Lastly, we assess the trade-off between the unroll iterations and graph block count.

7.7.1 Hop Connections

The decision to use hop connections was based on the results from the first phase of the training. The error decreased significantly with hop connections in the first phase. However, as can be seen in Table 7.4, after fine-tuning, where the model truly learns the propagation of sound waves, hop connections only result in an insignificant error decrease.

Table 7.4: Reason behind the usage of multiple hop connections in the proposed solution was based on the results from the first stage of the training. Nonetheless, MSE from the fine-tuning stage prove that the hop connections are unnecessary.

	Hops $\{3, 5, 10\}$	**No Hops**
First Phase	24.78	26.12
Fine-tuning	**14.58**	14.62

We hypothesize that the hop connections redundancy in the latter stage of the training is caused by the network starting to act as an iterative solver, where the prediction only requires a smaller neighborhood rather than the whole domain.

7.7.2 Training Noise

Pfaff et al. [34] demonstrated that noise injection during training improved the unroll stability and lowered overall error. The interpretation is that the error accumulating during unrolling can be reduced by simulating this error in the network input. It leads the network to learn correcting the error, thus lowering the overall accumulated error occurring during unrolling.

Although Pfaff et al. [34] use noise injection in the prediction of dynamical systems, we test the training noise in our proposed solution as well. We analyzed the distribution of the noise after 8 unroll iterations and decided to model the error within 10 %. The training noise is set to the 8-th root of the error. We model the training noise as the multiplicative noise, because the unroll error depends on the predicted wavefield. The following equation describes the training noise:

$$\tilde{u^k} = u^k + u^k * \text{noise}, \tag{7.4}$$

where u^k is the predicted wavefield at the k-th iteration. The noise is implicitly added to the residual R^k by injecting the noise in the wavefield u^k.

Table 7.5: Injecting the 1.2 % training noise, described by Equation (7.4), simulates the prediction error. Using the training noise forces the model to learn correcting its prediction error during unrolling.

Noise [%]	0	0.6	1.2	2.4	4.8
MSE	14.58	14.56	**14.37**	15.76	15.43

As shown in Table 7.5, by injecting the 1.2 % noise during training, the model's performance improved. Hence, the model learned to reduce the prediction error from previous iterations. Any higher training noise ended up with worse results.

7.7.3 Unroll Iterations vs. Message Pass Count

This section examines the trade-off between unroll iterations and message passing count. With every unroll iteration, a new residual is computed from the predicted wavefield. On the contrary, a Graph Block only passes a message (hidden state) to the next block. In the following experiments, we set a fixed total message passing count to 60, so the ratio between the unroll iterations and graph blocks is the only one that changed.

As Table 7.6 shows, unroll iterations are essential for the wavefield prediction. More Graph Blocks (GB) result in more learnable parameters – increasing the model's capacity. Nevertheless, the model with 5 Graph Blocks has the lowest error. We suppose that the unroll iterations are crucial, due to the addition of residual to the network's input. We observed, that the model could not predict more complex wave reflections and interactions without the residual as an input. We assume it is due to the network's inability to learn a Laplacian operator. This experiment was conducted after the design choices. For that reason, the proposed solution has 10 Graph Blocks instead of 5.

Table 7.6: In this experiment, we fixed the total message passing count to 60. Exchanging the unroll iterations for message pass count resulted in a higher error. We assume that the unroll iterations are essential because the network is not able to compute the residual itself. Without the residual as an input, the model is not able to predict complex wave reflections. GB refers to a Graph Block, which corresponds to a single message passing.

	60 Unrolls, 1 GB	**20 Unrolls, 3 GBs**	**12 Unrolls, 5 GBs**	**6 Unrolls, 10 GBs**	**4 Unrolls, 15 GBs**
MSE	46.27	14.52	**14.00**	19.69	21.66

Chapter 8

Conclusion

We proposed a novel iterative solver based on the graph neural networks capable of solving large domains such as 512×512, thus moving beyond "toy" problems. Unfortunately, it is slower than Helmnet as well as the reference solver k-Wave. In addition, Helmnet achieves a lower error by almost two orders of magnitude compared to our solutions. Nevertheless, our model is able to perform a super-resolution, where it reached a lower error than the baseline method – linear interpolation.

Even though our model was only trained on samples with a single source, it can predict a wavefield with multiple sources, proving that it learned the interaction between the sound waves. Additionally, our model can perform an inference in the 512×512 computational domain, although the model was only trained in the 96×96 domain.

We demonstrated that the graph neural networks are able to solve a second-order time-independent PDE within a large computational domain. Also, we outlined the problem of training an effective model on an irregular grid. Lastly, we tested a neural solver against the Lottery Ticket Hypothesis [10]. We were only able to produce winning tickets with a low level of sparsity $P_m \geq 72\%$. The winning tickets are not sparse enough to utilize sparse multiplication, though the iterative pruning can be used to improve the accuracy of the model.

Our model outperforms the linear interpolation at the super-resolution task on an irregular grid. Additionally, our model is capable of predicting a wavefield even with the down-sampled Laplacian. After downsampling the Laplacian, our model's accuracy barely drops compared to the k-Wave solver. To the best of our knowledge, our method is the only PDE solver capable of working with just three points per wavelength.

In the future, given the flexibility of Graph Networks, emphasis can be put on training on the data in the 2D space and later translating it to the 3D space. We hypothesize that AGS could replace the spectral method to compute the residual with denser sampling.

we expect that the results from this work will be used to produce a scientific paper with the collaboration of the UCL Biomedical Ultrasound Group, SC@FIT, and Graph@FIT groups. Preliminary results of this work were already published at the Excel@FIT2022 student conference, where our work won *all* three awards.

Bibliography

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018. URL http://arxiv.org/abs/1803.08375.

[2] Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural operator: Graph kernel network for partial differential equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020. URL https://openreview.net/forum?id=fg2ZFmXFO3.

[3] Alan Ayala, Stanimire Tomov, Miroslav Stoyanov, and Jack Dongarra. Scalability issues in fft computation. In *Parallel Computing Technologies: 16th International Conference, PaCT 2021, Kaliningrad, Russia, September 13–18, 2021, Proceedings*, page 279–287, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-86358-6. doi: 10.1007/978-3-030-86359-3_21. URL https://doi.org/10.1007/978-3-030-86359-3_21.

[4] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018. URL http://arxiv.org/abs/1806.01261.

[5] Alfredo Bermudez, Luis Hervella-Nieto, Andrés Prieto, and R. Rodrıguez. An optimal perfectly matched layer with unbounded absorbing function for time-harmonic acoustic scattering problems. *Journal of Computational Physics*, 223: 469–488, 05 2007. doi: 10.1016/j.jcp.2006.09.018.

[6] Norbert N. Bojarski. The k-space formulation of the scattering problem in the time domain. *Journal of the Acoustical Society of America*, 72:570–584, 1982.

[7] Norbert N. Bojarski. The k-space formulation of the scattering problem in the time domain: An improved single propagator formulation. *Acoustical Society of America Journal*, 77(3):826–831, March 1985. doi: 10.1121/1.392051.

[8] Benjamin Paul Chamberlain, James Rowbottom, Maria Goronova, Stefan Webb, Emanuele Rossi, and Michael M Bronstein. Grand: Graph neural diffusion. *Proceedings of the 38th International Conference on Machine Learning, (ICML) 2021, 18-24 July 2021, Virtual Event*, 2021.

[9] Steffen Eger, Paul Youssef, and Iryna Gurevych. Is it time to swish? comparing deep learning activation functions across NLP tasks. *CoRR*, abs/1901.02671, 2019. URL http://arxiv.org/abs/1901.02671.

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

[11] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *CoRR*, abs/1903.01611, 2019. URL http://arxiv.org/abs/1903.01611.

[12] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1263–1272. JMLR.org, 2017.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[14] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734 vol. 2, 2005. doi: 10.1109/IJCNN.2005.1555942.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. doi: 10.1109/ICCV.2015.123.

[16] David S. Hersh and Howard M. Eisenberg. Current and future uses of transcranial focused ultrasound in neurosurgery. *J Neurosurg Sci*, 62(2):203–213, Apr 2018.

[17] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL https://www.sciencedirect.com/science/article/pii/089360809190009T.

[18] Frank Ihlenburg and Ivo Babuska. Finite element solution of the Helmholtz equation with high wave number Part I: The h-version of the FEM. *Computers & Mathematics With Applications*, 30:9–37, 1995.

[19] Steven Kapturowski, Georg Ostrovski, Will Dabney, John Quan, and Remi Munos. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1lyTjAqYX.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[21] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

[22] Vibhor Krishna., Francesco Sammartino., and Ali Rezai. A Review of the Current Therapies, Challenges, and Future Directions of Transcranial Focused Ultrasound Technology: Advances in Diagnosis and Treatment. *JAMA Neurol*, 75(2):246–254, 02 2018.

[23] Isaac E. Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998. doi: 10.1109/72.712178.

[24] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. doi: 10.1007/978-3-642-35289-8_3. URL https://doi.org/10.1007/978-3-642-35289-8_3.

[25] Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *Proceedings of ICLR'16*, April 2016. URL https://www.microsoft.com/en-us/research/publication/gated-graph-sequence-neural-networks/.

[26] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=c8P9NQVtmnO.

[27] Wenzhuo Liu, Mouadh Yagoubi, and Marc Schoenauer. Multi-resolution Graph Neural Networks for PDE Approximation. In *Artificial Neural Networks and Machine Learning – ICANN 2021*, volume 12893 of *Lecture Notes in Computer Science*, pages 151–163. Springer International Publishing, September 2021. doi: 10.1007/978-3-030-86365-4_13. URL https://hal.archives-ouvertes.fr/hal-03448278.

[28] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3282–3291, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/lu18d.html.

[29] Jaron Maene, Mingxiao Li, and Marie-Francine Moens. Towards understanding iterative magnitude pruning: Why lottery tickets win, 2021. URL https://arxiv.org/abs/2106.06955.

[30] Claudio Mancinelli, Marco Livesu, and Enrico Puppo. Gradient Field Estimation on Triangle Meshes. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*. The Eurographics Association, 2018. ISBN 978-3-03868-075-8. doi: 10.2312/stag.20181301.

[31] T. Douglas Mast, Laurent P. Souriau, Donald L. Liu, Makoto Tabei, Adrian I. Nachman, and Robert C. Waag. A k-space method for large-scale models of wave propagation in tissue. *IEEE Trans Ultrason Ferroelectr Freq Control*, 48(2):341–354, Mar 2001.

[32] Jorge J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In G.A. Watson, editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 105–116. Springer Berlin Heidelberg, 1978.

[33] Gregory A. Newman and David L. Alumbaugh. Frequency-domain modelling of airborne electromagnetic responses using staggered finite differences. *Geophysical Prospecting*, 43:1021–1042, 1995.

[34] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

[35] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5105–5114, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[36] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378: 686–707, February 2019. doi: 10.1016/j.jcp.2018.10.045.

[37] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

[39] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

[40] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR, 2020. URL http://proceedings.mlr.press/v119/sanchez-gonzalez20a.html.

[41] Aliaksei Sandryhaila and José M. F. Moura. Discrete signal processing on graphs: Graph fourier transform. In *2013 IEEE International Conference on Acoustics,*

Speech and Signal Processing, pages 6167–6170, 2013. doi: 10.1109/ICASSP.2013.6638850.

[42] Franco Scarselli, Sweah Liang Yong, Marco Gori, Markus Hagenbuchner, Ah Chung Tsoi, and Marco Maggini. Graph neural networks for ranking web pages. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, WI '05, page 666–672, USA, 2005. IEEE Computer Society. ISBN 076952415X. doi: 10.1109/WI.2005.67. URL https://doi.org/10.1109/WI.2005.67.

[43] Bhavya R. Shah, Vance T. Lehman, Timothy J. Kaufmann, Daniel Blezek, Jeff Waugh, Darren Imphean, Frank F. Yu, Toral R. Patel, Shilpa Chitnis, Richard B. Dewey, Joseph A. Maldjian, and Rajiv Chopra. Advanced MRI techniques for transcranial high intensity focused ultrasound targeting. *Brain*, 143(9):2664–2672, 09 2020.

[44] C.E. Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37 (1):10–21, jan 1949. doi: 10.1109/jrproc.1949.232969. URL https://doi.org/10.1109/jrproc.1949.232969.

[45] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[46] Arnold Sommerfeld. Die greensche funktion der schwingungslgleichung. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 21:309–352, 1912. URL http://eudml.org/doc/145344.

[47] Antonio Stanziola, Simon R. Arridge, Ben T. Cox, and Bradley E. Treeby. A helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. *Journal of Computational Physics*, 441:110430, Sep 2021. ISSN 0021-9991. doi: 10.1016/j.jcp.2021.110430. URL http://dx.doi.org/10.1016/j.jcp.2021.110430.

[48] Makoto Tabei, T. Douglas Mast, and Robert C. Waag. A k-space method for coupled first-order acoustic propagation equations. *J Acoust Soc Am*, 111(1 Pt 1):53–63, Jan 2002.

[49] Hong Hui Tan and King Hann Lim. Review of second-order optimization techniques in artificial neural networks backpropagation. In *Materials Science and Engineering Conference Series*, volume 495 of *Materials Science and Engineering Conference Series*, page 012003, April 2019. doi: 10.1088/1757-899X/495/1/012003.

[50] Nils Thuerey, Philipp Holl, Maximilian Müller, Patrick Schnell, Felix Trost, and Kiwon Um. Physics-based deep learning. *CoRR*, abs/2109.05237, 2021. URL https://arxiv.org/abs/2109.05237.

[51] Bradley E. Treeby and Benjamin T. Cox. k-Wave: MATLAB toolbox for the simulation and reconstruction of photoacoustic wave fields. *Journal of Biomedical Optics*, 15(2):1 – 12, 2010. doi: 10.1117/1.3360308. URL https://doi.org/10.1117/1.3360308.

[52] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *6th International Conference on Learning Representations*, 2017.

[53] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7794–7803, 2018. doi: 10.1109/CVPR.2018.00813.

[54] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.*, 38(5), oct 2019. ISSN 0730-0301. doi: 10.1145/3326362. URL https://doi.org/10.1145/3326362.

[55] Ziming Wang, Tao Cui, and Xueshuang Xiang. A neural network with plane wave activation for helmholtz equation, 2020. URL https://arxiv.org/abs/2012.13870.

[56] Jeremy Watt, Reza Borhani, and Aggelos K. Katsaggelos. *Machine Learning Refined: Foundations, Algorithms, and Applications*. Cambridge University Press, 2016. doi: 10.1017/CBO9781316402276.