Deep PDE Solvers for Subgrid Modelling and Out-of-Distribution Generalization

PATRICK CHATAIN

Department of Mathematics and Statistics
FACULTY OF SCIENCE
McGill University, Montreal

MARCH 2024

A thesis submitted to McGill University
in partial fulfillment of the
requirements for the degree of
Master of Science

ABSTRACT

Climate and weather modelling (CWM) is an important area where machine learning (ML) models are used for subgrid modelling: making predictions of processes occurring at scales too small to be resolved by standard solution methods. These models are expected to make accurate predictions, even on out-of-distribution (OOD) data, and are additionally expected to respect important physical constraints of the ground truth model. While many specialized ML PDE solvers have been developed, the particular requirements of CWM models have not been addressed so far, marking the motivation behind this thesis. In this work, we explore the intersection of machine learning and PDEs, proposing and developing a novel model architecture which matches or exceeds the performance of standard ML models, and which demonstrably succeeds in OOD generalization. The architecture is based on expert knowledge of the structure of PDE solution operators, which also permits the model to obey important physical constraints.

ABRÉGÉ

La modélisation du climat et de la météo (CWM) est un domaine important où les modèles d'apprentissage automatique (ML) sont utilisés pour la modélisation sous-maille qui consiste en la prédictions sur des processus se produisant à des échelles trop petites pour être résolues par les méthodes de solution standard. Ces modèles doivent faire des prédictions précises, même sur des données hors distribution (OOD), et sont également contraints de respecter les contraintes physiques importantes du modèle de vérité terrain. Bien que de nombreux solveurs d'EDP (équations aux dérivées partielles) utilisant le ML aient été développés, les exigences particulières des modèles CWM n'ont pas encore été abordées, ce qui motive cette thèse. Dans ce travail, nous explorons l'intersection de l'apprentissage automatique et des EDP, en proposant et en développant une nouvelle architecture de modèles qui égalise ou dépasse les performances des modèles ML standards, et qui manifestement, réussit à se généraliser dans les situations OOD. L'architecture est basée sur la connaissance experte de la structure des opérateurs de solution des EDP, ce qui permet également au modèle de respecter les contraintes physiques importantes.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

The next list describes several abbreviations that will be later used within the body of the document:

AI: Artificial Intelligence

CFD: Computational Fluid Dynamics

ConvN: Convolutional Network

CWM: Climate and Weather Modelling

FCN: Fully Connected Network

ML: Machine Learning

MLP: Multilayer Perceptron

MSE: Mean Squared Error

OOD: Out-of-Distribution

PDEs: Partial Differential Equations

PINNs: Physics Informed Neural Networks

SGD: Stochastic Gradient Descent

CONTENTS

ACKNOWLEDGEMENTS

First and foremost, I extend my deepest gratitude to my supervisor, Adam. Your constant support, expert guidance, and invaluable mentorship have been the cornerstones of my research journey. Your academic insights and dedication have not only enriched this thesis but have also fostered my intellectual growth. Your financial support through research grants and scholarships has made pursuing this degree a reality, for which I am immensely thankful. I would also like to express my profound appreciation to McGill University for providing me with the opportunity to pursue this master's degree. In particular, I would like to thank the many professors and instructors at the university who shaped my academic journey. Your commitment to education and your passion for teaching have inspired me to strive for excellence.

To my parents, Alejandro and Christine, I owe an immeasurable debt of gratitude. Your unwavering belief in my potential, your sacrifices, and your continuous encouragement throughout my academic journey have been the pillars of my success. Your love and support have sustained me through the challenges, and I dedicate this achievement to you. To my girlfriend, Emily, your unconditional support, patience, and understanding have been my comfort during the highs and lows of this journey. Your continuous encouragement and belief in my abilities have been my constant source of motivation and none of this would have been possible without you. Lastly, to all my friends and family, thank you for always being there for me when things got hard. Whether it was going for a drink, playing soccer, or making a simple phone call, you always made things more enjoyable.

This master's thesis is the culmination of years of dedication, hard work, and the support of a remarkable community of individuals. I am deeply grateful for every opportunity, every lesson, and every person who has been part of this transformative experience. Thank you from the bottom of my heart.

CONTRIBUTION OF AUTHORS

This thesis is entirely authored by me, reflecting my comprehensive research during the duration of my Master's program. The foundation of this work is grounded in a paper that I co-authored with my supervisor, Adam Oberman, which was submitted to a conference workshop. The paper served as a pivotal starting point for the broader and more detailed exploration presented in this thesis, which significantly expands on the concepts presented in the paper, offering a deeper and more nuanced understanding of the subject matter, solely developed through my dedicated research efforts.

CHAPTER 1

INTRODUCTION

In the contemporary landscape of computational science, the quest for efficient and accurate solutions to partial differential equations (PDEs) stands as a cornerstone of advancement across diverse scientific and engineering domains. PDEs, the mathematical models that describe the continuum of physical phenomena ranging from fluid dynamics to electromagnetic fields, pose significant challenges due to their complexity and the often intractable nature of their analytical solutions. This thesis aims to bridge the gap between machine learning (ML) and traditional numerical methods by exploring the potential of machine learning algorithms in approximating PDE solution operators, particularly focusing on the critical aspect of subgrid modelling.

This thesis is structured to first lay the theoretical groundwork, introducing the basics of machine learning as well as defining the fundamental concepts of PDEs and the challenges inherent in their numerical solutions. It then delves into the research problem and its connection to PDEs, ML, and subgrid modelling. We then define our proposed model and test it against basic standard ML models. Finally, we present our results and set the direction for future research.

1.1 Partial Differential Equations, Numerical methods, and Stability

In short, partial differential equations are mathematical models that explain physical processes. They occur in a variety of applications like financial modelling, potential fields (gravitational, electrostatic, magnetostatic, etc.), probability densities, heat flow problems, and biological phenomena [Larsson and Thomée, 2009]. Solving these models, however, is not an easy task, and sometimes it is even impossible to obtain analytical solutions. When this occurs, numerical methods are a way to obtain approximations of solutions in an accurate way.

Finite difference methods dominated the early development of numerical analysis of partial differential equations. In this method, space is broken down into a finite grid of points, and the approximation of the differential equation is accomplished by replacing derivatives with difference quotients, reducing the differential equation problem to a finite linear system of algebraic equations [Larsson and Thomée, 2009]. While a simple method,

many mathematical details must be carefully addressed for the method to work correctly. One of the most important conditions is stability, since in an unstable finite differences scheme, solutions do not converge but rather "blow up" to infinity [Larsson and Thomée, 2009]. We will see later how having a stable scheme is crucial for solving PDEs.

1.2 MACHINE LEARNING

Machine learning is a field of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform a task without using explicit instructions. Instead, these systems learn and make decisions based on patterns and inferences from data [Shalev-Shwartz and Ben-David, 2014]. These models can be incredibly powerful, and thus machine learning has become a trending topic in today's world. Every year, numerous models are created that improve efficiency over previous models. Thus, as the field progresses and research in ML continues, we can generate models that we never thought would be possible before. ChatGPT, for example, might be the most significant advancement in recent years, with the whole world now constantly using it to increase productivity.

1.3 ML FOR PDES

With the vast range of uses of ML models and the powerful computational capacity available these days, it is no surprise that ML models can also be used to solve PDEs. Recently, there has been a lot of research into the intersection of these two areas and how we can leverage ML to help us solve PDE problems more accurately and more efficiently.

In recent years, many novel ML models have been developed to approximate PDE solution operators. Probably the most well-known is PINNs (Physics Informed Neural Networks), which uses known laws of physics described by general nonlinear partial differential equations to define the ML model architecture [Raissi et al., 2019]. PINNs demonstrate the key element that we are after; implementing PDEs structure directly into the model's architecture with the goal of obtaining more accurate solution operators. ML for PDEs is an active area of research, and numerous novel models have shown promising results. We discuss them further in Chapter 4.

1.4 SUBGRID MODELLING

Subgrid modelling is a concept used primarily in numerical simulations, especially in fields like computational fluid dynamics (CFD), meteorology, and climate and weather

modelling. Subgrid modelling addresses the challenge of simulating physical phenomena that occur at scales smaller than the grid resolution of the model (or, more practically, smaller than the scale at which we have observations) [Brasseur and Jacob, 2017]. In other situations, it might also be computationally impractical, or even impossible, to resolve all the scales of motion or changes due to limited computational resources. Subgrid models aim to solve these problems by approximating the effects of these smaller-scale phenomena on the larger scales that the grid can resolve.

1.5 Climate and weather modelling

In this section, we introduce the main motivation behind our work; Climate and Weather Modelling (CWM). CWM is an important research area which consists of the study of predictive models for physical and atmospheric processes. These processes are represented by time-dependent partial differential equations of fluid mechanics [Mcsweeney and Hausfather, 2018]. In order to model these PDEs, traditional climate and weather models break the ocean, atmosphere, and land up into many grid points and features that are too small or complex to be explicitly calculated in the model are approximated using coarser grids [Brasseur and Jacob, 2017, Balaji et al., 2022].

Recently, ML approaches have been used to make better approximations of these subgrid processes [Weyn et al., 2019, Bretherton et al., 2022, Watt-Meyer et al., 2021]. For example, Bolton and Zanna [2019] applied deep learning to ocean modelling, and found that they could decrease the data resolution by a factor of 5 to 10 while maintaining accuracy and conservation of momentum. However, these models fail to generalize to out-of-distribution (OOD) data, and they can violate physical constraints [Kashinath et al., 2021], two requirements of the CWM models. Thus, our motivation behind this work is to propose a new mechanism to develop ML models for climate and weather processes that also satisfies these requirements. We note that we are not creating climate models themselves, but rather proposing a mathematical framework that can address the limitations of current climate and weather models.

1.6 Research Objectives

Numerous Neural PDE solvers exist, each with its own unique approach and application. However, many of these models grapple with two critical limitations: Firstly, they often can't learn directly from data, necessitating a complete knowledge of the governing equations — a requirement that's infeasible in complex scenarios like weather and climate modelling. Secondly, they are constrained by the curse of dimensionality, compelling

them to resolve at finer scales and thereby excluding the use of subgrid models. While recent advancements in neural PDE solvers are promising, they often don't incorporate the comprehensive controls that traditional PDE solvers offer, such as stability, convergence guarantees, and the preservation (or near-preservation) of qualitative solution attributes like energy, mass, and linearity, to name a few. In this study, we focus on a specific PDE problem pertinent to climate, atmospheric, and oceanic modelling.

Standard machine learning models have shown efficacy in learning PDE solution operators. However, when applied to climate and weather models, their limitations become evident. Specifically, they might not always adhere to the fundamental governing laws of physical systems, and their generalizability to out-of-distribution data can be questionable, as pointed out by Kashinath et al. [2021]. It is imperative for PDE solvers in this context to be not only physically consistent and scientifically rigorous but also data-efficient, requiring fewer input data points. Moreover, they should be adept at making reliable predictions for unseen and non-stationary scenarios, such as changing climates.

Subgrid models address some of these challenges by solving PDEs at coarser resolutions, thus necessitating fewer data points to capture the complete physical process. The importance of these models in climate and weather studies is underscored by the fact that no single resolution can capture all climate-relevant phenomena [Balaji et al., 2022]. Additionally, subgrid models offer computational advantages over fully-resolved grid models in many climate areas, for example, orographic precipitation [Leung and Ghan, 1995]. Bolton and Zanna [2019] implemented neural networks to model turbulent processes and subsurface flow fields at coarser resolutions, but their results showed a need for generalization and better subgrid accuracy. A different approach outlined by Kashinath et al. [2021] is custom-designed neural network architectures to enforce physical constraints. However, until now, these have not been made to work.

The challenge of OOD generalization is pervasive across the machine learning landscape. In scientific applications, the ability to apply models to data distributions distinct from the training set is of incredible value. Yet, this remains one of ML's most formidable challenges, with climate and weather models often falling short in their generalization to novel weather patterns [Kashinath et al., 2021]. Through our research, we aim to demonstrate that incorporating extensive domain knowledge and expected mathematical properties directly into the models can facilitate OOD generalization. We contend that a deep understanding of the underlying mathematical structures and domain-specific knowledge can guide the training process, rendering models that are not just data-driven but also grounded in theory, fostering a robust generalization performance for unseen data

distributions.

In this paper, we propose a pioneering architecture crafted for solving inverse problems in PDEs while aiming to exhibit exceptional out-of-distribution generalization. Our contribution is a hybrid neural network model that has many of the controls built into PDE solvers, but can also learn from data and overcome the grid limitations. Leveraging the mathematical properties inherent to PDEs, this innovative architecture promises to carve a pathway toward resolving the long-standing issue of OOD generalization by embedding structured knowledge into the learning process.

1.6.1 *Key contributions*

Our key contributions through this research are outlined as follows:

1. **OOD generalization for subgrid PDE solution operators:** By restricting to an architecture grounded in theory, we show that we can accurately approximate the true solution operator, even on OOD data.

2. **Accurate subgrid models:** Addressing the needs of scientific computing, we develop and integrate subgrid solvers into our model, which maintain accuracy even at reduced grid resolutions (data dimensionality).

3. **Physical constraints satisfied:** Our model hypothesis class has the benefits of traditional PDE solvers, which satisfy physical constraints by construction, and has the framework of a neural network training pipeline.

In the broader landscape, many are making significant strides, like establishing foundation models for climate and weather. Our paper's intent is to spotlight the OOD generalization challenge using a concise example where we can study precisely what happens. We illustrate the construction of an architecture adept for OOD generalization in subgrid models, hoping our insights will pave the way for future endeavours in more expansive projects.

CHAPTER 2

MACHINE LEARNING

Machine Learning stands as one of the most transformative fields in computer science and artificial intelligence. In its essence, ML is a subset of artificial intelligence that focuses on enabling computers to learn from data and make predictions or decisions without being explicitly programmed. At its core, ML algorithms seek to uncover patterns, correlations, and structures within datasets, allowing them to generalize from observed examples to unseen data. This chapter presents a brief literature review that navigates through the core elements of machine learning, encompassing neural networks, loss functions, and training optimization.

2.1 NEURAL NETWORKS: THE FOUNDATION OF DEEP LEARNING

Neural networks, inspired by the human brain's neural architecture, are the cornerstone of modern ML [Stanley et al., 2019]. These computational models consist of interconnected layers of artificial neurons, which process information and learn complex representations from data. While neural networks vary in architecture, they all share the same underlying concept.

Each layer of a neural network consists of a linear mapping composed with a non-linear activation:

$$f_i(x) = \sigma_i(W_i x + b_i),$$

where W_i is a matrix of learnable parameters (or weights), b_i is a (learnable) bias vector, and $\sigma_i(\cdot)$ is a non-linear activation function that is applied element-wise. Of course, the dimensions of W_i and b_i are chosen to match the input size and desired output size of the layer.

A Multilayer Perceptron (MLP), the simplest version of a neural network, is just a combination of many of these layers one after another [Kruse et al., 2022]. An MLP of K layers would then take the form

$$\begin{aligned} f_{MLP}(x) &= f_K(f_{K-1}(\dots f_2(f_1(x))\dots)) \\ &= \sigma_K(W_K\sigma_{K-1}(W_{K-1}\dots\sigma_2(W_2\sigma_1(W_1x+b_1)+b_2)\cdots+b_{K-1})+b_K). \end{aligned} \tag{2.1}$$

Here x is our input vector and the final output is our output vector. Each intermediate output obtained after applying f_i for $i \neq K$ is considered a hidden layer of the network. Hence the network described above would contain $K - 1$ hidden layers and one output layer. Below is a graphical representation for $K = 3$.

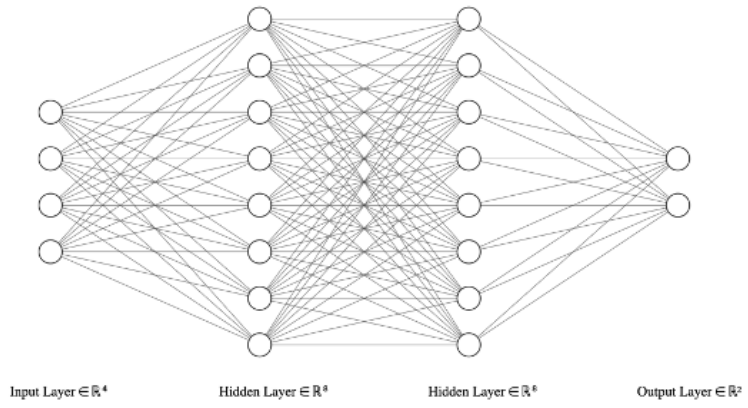


Figure 2.1: *An example of a multilayer perceptron with 2 hidden layers [LeNail, 2019].*

The activation functions at the end of each layer are crucial to the computational power of neural networks. While there are many common choices for the activation functions of hidden layers, the output layer will usually have a softmax or identity activation function, depending on the task of the network (usually identity for regression problems and softmax for classification problems) [Kiaei et al., 2023]. But more importantly, it is precisely the non-linearity between layers that allows neural networks to fit data in such accurate ways. In fact, the universal approximation theorem states that a neural network with a single hidden layer and suitable activation function is able to approximate any continuous function up to arbitrary accuracy [Kidger and Lyons, 2020, Cybenko, 1989].

2.2 Training Optimization: Navigating the Parameter Space

Having defined an architecture for a neural network (number of layers, activation functions, etc.) we now turn to the question of how to train the network in order to get the best-performing model. This involves optimizing the internal parameters in the model to minimize a given loss function. Loss functions quantify the disparity between predicted outcomes and actual data [Mohri et al., 2018]. The simplest example would be the L_2 loss, or mean squared error (MSE).

For a given network $f_\theta(x)$ (where θ represents the collection of parameters of the network),

we define the L_2 loss of a given $(input, output)$ pair as

$$L_2(x, y) = (x - y)^2.$$

The total loss of the model is then defined as an average of the MSE over all the training data [Shalev-Shwartz and Ben-David, 2014]:

$$L_{f_\theta} = \frac{1}{N} \sum_{i=1}^{N} (f_\theta(x_i) - y_i)^2. \tag{2.2}$$

There are many modern optimization algorithms for minimizing the loss, such as Adagrad, RMS Prop, and Adam [Duchi et al., 2011, Kingma and Ba, 2014]. The simplest of all, however, is stochastic gradient descent (SGD). This involves taking a small step in the direction of negative the gradient of L at every iteration. More precisely, we have that the parameters of the model are updated according to

$$\theta_{i+1} = \theta_i - \mu \nabla L_{f_\theta},$$

where μ is just a small constant (called the step-size).

Due to the nonlinearity of neural networks, this is a non-convex optimization problem which often might have many local minima [Kingma and Ba, 2014]. This is precisely the reason that SGD is implemented (where one data point is fed in at each iteration) instead of normal gradient descent (where all the data points are fed in at each iteration). In practice, batch gradient descent (where a few points are implemented at a time) is a very common practice.

2.3 A SIMPLE EXAMPLE: LINEAR REGRESSION

One simple problem that demonstrates this optimization pipeline is fitting a linear regression problem. Suppose we have a set of data,

$$D = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$$

and we want to fit a linear model as well as possible to the data:

$$f_W(\boldsymbol{x}) = W\boldsymbol{x}.$$

We can define our loss function as the L_2 loss defined above in (2.2) (the standard loss for least squares regression), and use a standard optimizer like SGD (as explained above) and we will solve the problem

$$\min_W \frac{1}{N}\sum_{i=1}^{N}(f_W(\boldsymbol{x}_i)-\boldsymbol{y}_i)^2 = \min_W \frac{1}{N}\sum_{i=1}^{N}(W\boldsymbol{x}_i-\boldsymbol{y}_i)^2.$$

Essentially, we are constraining our models (or hypothesis space) to the set of linear models, and we are finding the best possible parameters by optimizing the L_2 loss through gradient descent. At convergence, the optimal parameters found, $W^\star$, will then give us our optimal model

$$f^\star(\boldsymbol{x}) = W^\star\boldsymbol{x}.$$

Of course, there are other standard ways to solve linear regression problems, including even analytical solutions, but these methods often have strong requirements like the covariance matrix being invertible. Therefore, by setting up the problem through a machine-learning perspective we can obtain very accurate solutions using an efficient optimization pipeline.

2.4 Architecture: The key to best-performing models

The previous example was just a very simple problem that can be tackled with machine learning. In practice, we have incredibly complex problems that require models with deep architectures and large numbers of layers (like in (2.1)). Designing such models is not an easy task and picking the best architecture (number of layers, type of layers, activation functions, etc.) for a neural network is an open problem. There is no methodology for finding the optimal architecture for a given task, and the best architectures and hyperparameters are often found through experimentation. Nevertheless, the field of ML and deep learning has advanced enough for certain models to become the standard when it comes to particular tasks. For example, convolutional neural networks for image classification [Krizhevsky et al., 2012] and transformers for large language models [Vaswani et al., 2017]. Each of these types of models has properties that cater to their respective objectives, so we can see that even though there is no optimal methodology to find the best architecture, domain knowledge of the task or goal of the model plays an important role in designing a good architecture.

CHAPTER 3

STABILITY THEORY

Partial Differential Equations are a fundamental tool for modelling a wide range of physical phenomena and natural processes in various scientific disciplines, including physics, biology, and economics [Larsson and Thomée, 2009]. They describe the evolution of quantities that depend on multiple independent variables and their partial derivatives. However, solving PDEs analytically is often an unfeasible task due to their complexity, and therefore mathematicians have come up with diverse numerical methods that provide practical and efficient solutions. These methods, of course, come with their own set of challenges and considerations, particularly concerning stability and consistency.

This chapter aims to introduce the Forward Euler Method, a numerical method for solving PDEs. It emphasizes the significance of stability and consistency in ensuring its convergence, as well as deriving explicit conditions that ensure stability. It is important to note that while our discussions will use the heat equation as an example, the concepts and principles outlined are universally applicable to a diverse array of PDEs.

3.1 THE CONTINUOUS PROBLEM

Start with the heat equation with Dirichlet boundary conditions:

$$\begin{aligned} u_t &= u_{xx}, && \text{in} \quad [0,1] \times \mathbb{R}_+ \\ u(\cdot, 0) &= v(x), && \text{in} \quad [0,1] \\ u(0,t) &= u(1,t) = 0, && \text{in} \quad \mathbb{R}_+. \end{aligned} \tag{3.1}$$

The solution to this problem can be obtained through separation of variables and Fourier Series [Joel, 2007], but we are more interested in the behaviour of the system as time evolves. More precisely we want to examine the behaviour of the L_2 norm of the solution

$\|u\|^2 = \int_0^1 u^2 dx$ as $t \to \infty$. Taking the derivative of the norm with respect to t we get

$$\begin{aligned}
\tfrac{\partial}{\partial t}\int_0^1 u^2\,dx &= \int_0^1 \frac{\partial}{\partial t}u^2\,dx \\
&= 2\int_0^1 uu_t\,dx \\
&= 2\int_0^1 uu_{xx}\,dx && \text{from (3.1)} \\
&= 2\left(uu_x\Big|_0^1 - \int_0^1 u_x^2\,dx\right) && \text{integration by parts} \\
&= -2\int_0^1 u_x^2\,dx && \text{using the boundary conditions} \\
&\leq 0
\end{aligned}$$

which shows that the solution norm is non-increasing in time [Olof, 2013]. More precisely we have that

$$\|u(x,t)\| \leq \|u(x,0)\| = \|v(x)\|. \tag{3.2}$$

This is a very desirable property since it means that a small perturbation of the initial state $v(x)$ won't cause a big discrepancy in the solution. Indeed, note that if we have u_1 and u_2 satisfying (3.1) with $v(x) = g_1(x)$ and $v(x) = g_2(x)$ respectively, then $u_1 - u_2$ also satisfies (3.1) with $v(x) = g_1(x) - g_2(x)$. Then by (3.2) we have that

$$\|u_1 - u_2\| \leq \|g_1 - g_2\|$$

so the difference in the solutions is bounded by the difference in the initial conditions, and we say that the system is stable.

3.2 The Discrete Problem

Now take (3.1) and discretize it by introducing mesh points $\{(x,t)\} = (x_j, t_n)$ where we have $x_j = jh$ and $t_n = nk$ [Larsson and Thomée, 2009]. Here h is the mesh width in x (this is our discrete approximation of dx) and k is the time-step (discrete approximation of dt). Now, using the notation $U_j^n := u(x_j, t_n)$ we define our discrete approximations to the partial derivatives as:

- $u_x \approx \partial_x U_j^n := \frac{U_{j+1}^n - U_j^n}{h}$,

- $u_x \approx \bar{\partial}_x U_j^n := \frac{U_j^n - U_{j-1}^n}{h}$, and
- $u_t \approx \partial_t U_j^n := \frac{U_j^{n+1} - U_j^n}{k}$.

We can then turn back to (3.1) and use the forward Euler method to define a discrete approximation to the equation as

$$\begin{aligned} \partial_t U_j^n &= \partial_x \bar{\partial}_x U_j^n, \quad \text{for} \quad j \in \{0, 1, \dots, J\}\ ,\ n \in \mathbb{N} \\ U_j^0 &= v(x_j), \qquad\quad \text{for} \quad j \in \{0, 1, \dots, J\} \\ U_0^n &= U_J^n = 0, \qquad\quad \text{for} \quad n \in \mathbb{N}, \end{aligned} \tag{3.3}$$

where $J = h^{-1}$ is the number of intervals in the space mesh [Larsson and Thomée, 2009]. Plugging our discrete operators into (3.3) we get that

$$\frac{U_j^{n+1} - U_j^n}{k} = \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{h^2}$$

and then we can solve for U_j^{n+1} to get

$$U_j^{n+1} = \lambda U_{j+1}^n + (1 - 2\lambda) U_j^n + \lambda U_{j-1}^n, \tag{3.4}$$

where we defined $\lambda = \frac{k}{h^2}$.

Equation (3.4) defines the core algorithm that we will use to generate the numerical solution to our problem, and there are some important properties we would like it to satisfy before we continue with our discussion. These properties are consistency and stability and are defined below.

Definition 3.2.1 (Consistency)**.** *A numerical method is said to be* ***consistent*** *if its discrete operator converges towards the continuous operator.*

In other words, if we replace the continuous operator in (3.1) with its discrete analogue in (3.3), the discrepancy should tend to zero as the step sizes tend to zero [LeVeque, 2007]. Mathematically this takes the form

$$\begin{aligned} \tau_j^n &= \partial_t u_j^n - \partial_x \bar{\partial}_x u_j^n - (u_t(x_j, t_n) - u_{xx}(x_j, t_n)) \\ &= (\partial_t u_j^n - u_t(x_j, t_n)) - (\partial_x \bar{\partial}_x u_j^n - u_{xx}(x_j, t_n)) \\ &= \left(\tfrac{u(x_j, t_n + k) - u(x_j, t_n)}{k} - u_t(x_j, t_n) \right) - \left(\tfrac{u(x_j - h, t_n) - 2u(x_j, t_n) + u(x_j + h, t_n)}{h^2} - u_{xx}(x_j, t_n) \right). \end{aligned} \tag{3.5}$$

Now using the Taylor series expansion for u we get that

$$u(x_j, t_n + k) = u(x_j, t_n) + ku_t(x_j, t_n) + \frac{k^2}{2}u_{tt}(x_i, \bar{t_n})$$

for some $\bar{t_n} \in (t_n, t_n + k)$ and thus rearranging we get that

$$\frac{u(x_j, t_n + k) - u(x_j, t_n)}{k} = u_t(x_j, t_n) + \frac{k}{2}u_{tt}(x_j, \bar{t_n}).$$

On the other hand, we have that

$$u(x_j + h, t_n) = u(x_j, t_n) + hu_x(x_j, t_n) + \frac{h^2}{2}u_{xx}(x_j, t_n) + \frac{h^3}{6}u_{xxx}(x_j, t_n) + \frac{h^4}{24}u_{xxxx}(\bar{x_j}, t_n)$$

for some $\bar{x_j} \in (x_j, x_j + h)$ and similarly that

$$u(x_j - h, t_n) = u(x_j, t_n) - hu_x(x_j, t_n) + \frac{h^2}{2}u_{xx}(x_j, t_n) - \frac{h^3}{6}u_{xxx}(x_j, t_n) + \frac{h^4}{24}u_{xxxx}(\tilde{x_j}, t_n)$$

for some $\tilde{x_j} \in (x_j - h, x_j)$.

Hence adding the last two equations we get that

$$u(x_j - h, t_n) + u(x_j + h, t_n) = 2u(x_j, t_n) + h^2u_{xx}(x_j, t_n) + \frac{h^4}{24}u_{xxxx}(\bar{x_j}, t_n) + \frac{h^4}{24}u_{xxxx}(\tilde{x_j}, t_n),$$

$$\frac{u(x_j - h, t_n) - 2u(x_j, t_n) + u(x_j + h, t_n)}{h^2} = u_{xx}(x_j, t_n) + \frac{h^2}{24}\left(u_{xxxx}(\bar{x_j}, t_n) + u_{xxxx}(\tilde{x_j}, t_n)\right).$$

Replacing these results into (3.5) we get that

$$\tau_j^n = \frac{k}{2}u_{tt}(x_j, \bar{t_n}) - \frac{h^2}{24}\left(u_{xxxx}(\bar{x_j}, t_n) + u_{xxxx}(\tilde{x_j}, t_n)\right)$$

and thus $\tau_j^n \to 0$ as $k, h \to 0$ so the method is consistent.
It is important to note though that we assumed smoothness conditions for u when applying its Taylor series. Namely, we assume that u is C^2 in t and C^4 in x.

Definition 3.2.2 (Stability). *A numerical method is said to be **stable** if for any $T > 0$, there exists a constant $C > 0$, which depends only on T, such that for $n = 0, 1, \ldots, \frac{T}{\Delta t}$ we have that*

$$\left(\Delta x \sum_i |u_i^n|^2\right)^{\frac{1}{2}} < C.$$

In simpler words, stability means that the L_2 norm of u^n at every time-step less than T is bounded by T [Lee and Jeong, 2017]. This is a very important property since it guides us in determining whether small disturbances in the initial conditions will eventually dissipate

or amplify, offering insight into the system's long-term behaviour.

Now take (3.4) and note that if $\lambda \leq \frac{1}{2}$ this is a convex combination of previous solution states and thus we have that

$$\begin{aligned} |U_j^{n+1}| &= |\lambda U_{j+1}^n + (1-2\lambda)U_j^n + \lambda U_{j-1}^n| \\ &\leq |\lambda U_{j+1}^n| + |(1-2\lambda)U_j^n| + |\lambda U_{j-1}^n| \\ &= \lambda|U_{j+1}^n| + (1-2\lambda)|U_j^n| + \lambda|U_{j-1}^n| \\ &\leq \sup_j |U_j^n|. \end{aligned}$$

This holds for all j so we get that $\sup_j |U_j^{n+1}| \leq \sup_j |U_j^n|$, or more precisely

$$\|U_j^{n+1}\|_\infty \leq \|U_j^n\|_\infty$$

which clearly implies that for all n:

$$\|U_j^n\|_\infty \leq \|U_j^0\|_\infty = \|v(x)\|_\infty.$$

This is the discrete analogue of (3.2) and we can see that under the condition $\lambda \leq \frac{1}{2}$ the numerical solution is stable.

Theorem 3.2.3 (Lax Equivalence Theorem)**.** *A consistent numerical method is convergent if and only if it is stable.*

We proved that under the step-sizes condition $\frac{k}{h^2} \leq \frac{1}{2}$, the Forward Euler Method is both consistent and stable. Therefore, by the Lax Equivalence Theorem, the method is also convergent. For a proof of the theorem see [Tekriwal et al., 2021].

It is crucial to remark on the importance of the $\lambda \leq \frac{1}{2}$ bound since we cannot ensure the stability (and hence the convergence) of the numerical method when $\lambda > \frac{1}{2}$. As an example, set an initial value problem as

$$v_j := U_j^0 = v(x_j) = (-1)^j \sin(\pi j h), \quad \text{for} \quad j = 0, 1, \ldots, J$$

[Larsson and Thomée, 2009] which satisfies the boundary conditions $U_0^n = U_J^n = 0$ (recalling that $h = J^{-1}$) and then applying the forward Euler method we get by induction that

$$U_j^n = (1 - 2\lambda - 2\lambda\cos(\pi h))^n v_j. \tag{3.6}$$

The base case is trivial and then note that

$$\begin{aligned} v_{j-1} &= (-1)^{j-1}\sin(\pi(j-1)h) \\ &= (-1)^{j-1}(\sin(\pi jh)\cos(-\pi h) + \cos(\pi jh)\sin(-\pi h)) \\ &= -(-1)^{j}(\sin(\pi jh)\cos(-\pi h) + \cos(\pi jh)\sin(-\pi h)). \end{aligned}$$

Similarly we get that $v_{j+1} = -(-1)^{j}(\sin(\pi jh)\cos(\pi h) + \cos(\pi jh)\sin(\pi h))$ and thus we have that

$$\begin{aligned} v_{j-1} + v_{j+1} &= -2(-1)^{j}(\sin(\pi jh)\cos(\pi h)) \\ &= -2v_j\cos(\pi h). \end{aligned}$$

Now let $\bar{\lambda} = 1 - 2\lambda - 2\lambda\cos(\pi h)$ and then using (3.4) we get that

$$\begin{aligned} U_j^{n+1} &= \lambda U_{j+1}^n + (1-2\lambda)U_j^n + \lambda U_{j-1}^n \\ &= \lambda(\bar{\lambda})^n v_{j+1} + (1-2\lambda)(\bar{\lambda})^n v_j + \lambda(\bar{\lambda})^n v_{j-1} \\ &= \lambda(\bar{\lambda})^n (v_{j+1} + v_{j-1}) + (1-2\lambda)(\bar{\lambda})^n v_j \\ &= -2\lambda(\bar{\lambda})^n v_j\cos(\pi h) + (1-2\lambda)(\bar{\lambda})^n v_j \\ &= (1 - 2\lambda - 2\lambda\cos(\pi h))(\bar{\lambda})^n v_j \\ &= (\bar{\lambda})^{n+1} v_j \\ &= (1 - 2\lambda - 2\lambda\cos(\pi h))^{n+1} v_j \end{aligned}$$

so we proved (3.6) by induction.

Then we can easily see that by letting h be small enough $|1 - 2\lambda - 2\lambda\cos(\pi h)| \geq \gamma > 1$ and thus we get that

$$\begin{aligned} \|U_j^n\| &= \|(1 - 2\lambda - 2\lambda\cos(\pi h))^n v_j\| \\ &= |1 - 2\lambda - 2\lambda\cos(\pi h)|^n \|v_j\| \\ &\geq \gamma^n \|v_j\| \end{aligned}$$

which goes to infinity as n goes to infinity, and thus the solution becomes unstable.

CHAPTER 4

LEARNING PDES WITH DATA

Having traversed the fundamental concepts in PDEs, stability theory, and numerical methods, we now stand ready to define our research problem. This chapter marks the transition from theory to application as we use all of our previous tools to address a pressing research problem that emerges at the intersection of mathematics and cutting-edge technology: the learning of PDEs through the integration of Machine Learning techniques, Stability Theory, and subgrid methods.

4.1 THE PDE PROBLEM

Start with a parabolic PDE of the form $u_t = a(x)u_{xx} + b(x)u_x + c(x)u + d(x)$ and define a PDE operator S that takes in as inputs an initial condition $v(x)$ and a vector of coefficient functions $(a(x), b(x), c(x), d(x))$, and outputs the solution to the underlying PDE $u(x,t)$. S is then the "forward" PDE operator. For example, for the heat equation with non-constant coefficients:

$$\begin{aligned} u_t &= a(x)u_{xx}, && \text{in} \quad [0,1] \times \mathbb{R}_+ \\ u(\cdot, 0) &= v(x), && \text{in} \quad [0,1] \\ u(0,t) &= u(1,t) = 0 && \text{in} \quad \mathbb{R}_+ \end{aligned} \tag{4.1}$$

we would have that S takes in $v(x)$ and $a(x)$ and returns the solution $u(x,t)$. If for example $a(x) = 1$ and $v(x) = \sin(\pi x)$ we have that

$$S(v(x), a(x)) = \sin(\pi x)e^{-\pi^2 t},$$

where the solution is obtained by just solving the PDE through separation of variables [Joel, 2007].

4.2 THE ML PROBLEM

Now we consider the other direction. Given samples of solutions of a time-dependent PDE at several time slices, our goal is to learn a solution operator F that maps an initial condition to its corresponding solutions at the respective time points:

$$F(u(\boldsymbol{x}, 0); \theta) = u(\boldsymbol{x}, t), \quad \boldsymbol{x}, t \in \Omega \times T, \tag{4.2}$$

where θ are the parameters of the model, Ω is our space domain, and T is the set of times at which we have observations.

We assume that the functions are all solutions of some time-dependent advection-diffusion partial differential equation, with unknown coefficients, $a(\boldsymbol{x}, t)$:

$$\partial_t u(\boldsymbol{x}, t) = L(a(\boldsymbol{x}, t), \nabla_{\boldsymbol{x}} u(\boldsymbol{x}, t), \nabla^2_{\boldsymbol{xx}} u(\boldsymbol{x}, t)), \quad u(\boldsymbol{x}, 0) = u_0(\boldsymbol{x}),$$

along with some known boundary conditions.

The ML problem is then to learn a function $F : \mathcal{D} \to \mathcal{Y}$ through a dataset

$$\mathcal{D} = \{u_i(\boldsymbol{x}, 0), \{u_i(\boldsymbol{x}, t_k)\}_{k=1}^K\}_{i=1}^N,$$

where $u_i(\boldsymbol{x}, 0)$ and $u_i(\boldsymbol{x}, t_k)$ are the initial conditions and their respective solutions at times t_k. In essence, we wish to generate a solution operator such that it approximates the solutions with high fidelity, i.e.,

$$F(u(\boldsymbol{x}, 0); \theta) \approx u(\boldsymbol{x}, t_k), \quad \forall t_k \in T,$$

where θ are the parameters of the learned operator and T is the set of times at which we have observations.

At its core, our ML problem will be similar to the linear regression problem discussed in Section 2.3. The solution operator we are modelling (the heat equation) is linear in the data, and therefore we are just applying a sort of "deep linear regression." Each layer of our model will take the previous output and apply a convolution operator, which can equivalently be represented as matrix multiplications. The shorter the convolution kernel, the more sparse the matrix is. Combining all the layers will therefore just amount to combining all these matrices into a big matrix multiplication, and then we optimize it through ML algorithms.

The only technicality about our model is that the entries of these matrices will be non-linear in the parameters themselves (not in the data) to account for stability conditions. We discuss the architecture of our model in more detail in Chapter 5, but we can essentially view our approach as a linear model with a non-linear parametrization.

4.3 THE SUBGRID PROBLEM

Subgrid modelling is a crucial concept in the realm of computational science and numerical simulations, finding applications in diverse fields, from climate and weather modelling to fluid dynamics and material science. In scenarios where simulating physical phenomena at fine spatial resolutions becomes computationally infeasible, subgrid modelling steps in to bridge the gap. Typically, it involves the representation of smaller-scale, unresolved features within a coarser grid by incorporating parameterized models that capture the effects of these subgrid-scale processes. In climate and weather modelling, for instance, subgrid modelling techniques are indispensable for capturing complex atmospheric and oceanic interactions that occur at spatial scales beyond the reach of high-resolution numerical simulations [Brasseur and Jacob, 2017].

In this section, we delve into the domain of subgrid modelling in the context of Partial Differential Equations. Building upon the foundation of machine learning and PDEs, we adapt our machine learning problem to address the specific challenges posed by subgrid modelling. Our objective is to develop a solution operator that accurately represents the behaviour of PDEs on a coarser grid, leveraging the insights gained from a finer grid.

Similar to before, we start with a dataset consisting of sample values of m functions of the form $u_i(\boldsymbol{x}, t)$ for $\boldsymbol{x}$ in a physical domain, and $t \in [0, T]$. The functions are sampled at points $\boldsymbol{x}_j$ in a uniform grid in space of resolution $N_{\boldsymbol{x}}$, and at time intervals T^{fine}, consisting of N_T, uniformly spaced time intervals. Each function is represented by a vector of the form $U_{i,j,k} = u_i(\boldsymbol{x}_j, t_k)$ and our training dataset, on the fine grid, is of the form,

$$D^{fine} = \{U_{i,j,k} = u_i(\boldsymbol{x}_j, t_k) \quad \boldsymbol{x}, t \in G^{fine} \times T^{fine}, i = 1, \ldots, m\}.$$

We are now given a list of target subgrid resolutions (for example, from fully resolved to an 8 times smaller grid resolution) and coarsened data of the form,

$$D^{coarse} = \{U_{i,j,k} = u_i(\boldsymbol{x}_j, t_k) \quad \boldsymbol{x}, t \in G^{coarse} \times T^{coarse}, i = 1, \ldots, m\}.$$

The goal is to learn, from the fine grid data, a solution map for each of the target grids

$$F(u(\boldsymbol{x}, 0); \theta) = u(\boldsymbol{x}, t), \quad \boldsymbol{x}, t \in G^{coarse} \times T^{coarse}, \tag{4.3}$$

where θ are the parameters of the model and T^{coarse} is the set of times at which we have observations on the coarse grid.

Note that the function values on the fine (well-resolved) grid would be sufficient to solve the PDE with acceptable accuracy using standard numerical PDE methods if the

coefficients were known. However, building a solution operator on the coarse grid requires machine learning tools since there are no analytical formulas for the operator of a coarse grid. For example, in current climate models, simplified operators are approximated, but this leads to a known loss of accuracy [Rasp et al., 2018]. Thus, our goal is to learn a subgrid solution operator as defined in equation (4.2) that accurately approximates the ground truth solution, as represented by the fine grid PDE solver in our example, or by assimilated data in a full-scale weather or climate model.

4.4 Relevant Work

In recent years, there has been a lot of work on ML PDE models, physics-informed models, inverse problems, subgrid models, and out-of-distribution generalization. For example, Liu et al. [2022] build neural network models that integrate PDE operators directly in the model while retaining the large capacity neural network architecture. However, they solve a different problem: learning the solution operator of several different PDEs but with constant coefficients. Additionally, Long et al. [2018] explored the learning of coefficients for the solution operator, though they did not delve into the subgrid aspects. On the other hand, the potential issues of out-of-distribution generalization with neural networks, especially for data with varied spectra, were highlighted by Rahaman et al. [2019]. In addition, early attempts at using physics-informed neural networks (PINNs) as PDE solvers were presented by Karniadakis et al. [2021] and Shin et al. [2020]. While innovative, these PINNs occasionally struggled with accurately representing the solution operator and ensuring physical constraints.

The inverse problem in PDEs has been widely studied. Its goal is to learn the coefficients of an operator given input-output pairs [Stuart, 2010, Kaipio and Somersalo, 2006], but it does not address the subgrid aspects of a solver. On the other hand, homogenization of PDEs takes the extreme approach of replacing an operator with a spatially homogeneous one [Marchenko and Khruslov, 2008], an approach that is valid in fields like material science but not in weather and climate modelling, where the emphasis is on the heterogeneous nature of operators.

Moving onto the subgrid area, Li et al. [2020] introduced the Fourier neural operator, which supports varying grids. However, their focus diverged towards a different PDE challenge: learning the map from the coefficients to the solution, which is different from our case, where we want to allow for different initial data to evolve in time. Recent contributions from Pfaff et al. [2021] and Han et al. [2022] present a PDE solver on irregular meshes.

Several other works connect neural network architectures and solution operators for differential equations. Chen et al. [2018] proposed a neural network architecture based on ODE solvers, and Haber and Ruthotto [2017] focused on the stability aspects of the architecture. In addition, Ruthotto and Haber [2020] advanced in this domain and proposed architectures based on discretized PDE solvers.

CHAPTER 5

OUR MODEL

In this chapter, we delve into the intricate architecture of the proposed model, a symbiotic fusion between machine learning and the rich theoretical domain of partial differential equations. The model's foundation lies in its ability to encode the underlying physical laws governing the system under study while harnessing the flexibility and adaptability of machine learning techniques. Through a meticulous exploration of the properties satisfied by the PDEs modelling the physical process, this chapter unveils the architecture and construction principles of this hybridized model, showcasing its unique capacity to synthesize domain expertise with data-driven intelligence, thereby offering a promising avenue for tackling complex real-world challenges like climate and weather modelling.

As we have mentioned several times, in this work we focus on the heat equation as the foundational physical process of our research. Hence, it is also the pillar for the model's construction since we will integrate domain-specific knowledge from it into the model's architecture. However, it should be noted that while our immediate focus lies within this domain, our work is intended as a novel proof-of-concept approach to showcase the efficacy of our architecture in capturing and leveraging physical processes. We chose the heat equation since it is much simpler to analyze than a system of advection-diffusion PDEs, or the Navier-Stokes equations, yet complex enough to highlight the results. This deliberate choice aims not only to demonstrate the model's capability from both a practical and theoretical point of view but also to beckon further research and exploration in this direction, fostering an avenue for a more expansive and impactful application of this hybridized approach. The overarching goal would be to establish a robust and adaptable framework capable of accommodating a spectrum of physical processes, underscoring our aspiration for this work to serve as a catalyst, motivating future endeavours to model and simulate a diverse array of complex phenomena across various scientific disciplines.

5.1 MODEL PROPERTIES

Based on the physical properties of the process being modelled: $\partial_t(\boldsymbol{x}, t) - a(\boldsymbol{x})\Delta u(\boldsymbol{x}, t)$, our strategy is grounded in four theoretically desirable properties for our solution operator. These are locality, stability, linearity, and memory-less recurrence. We explain and implement each of those four properties into our model's architecture as follows:

5.1.1 *Locality*

PDEs are local operators since they depend on the derivatives of the functions. Based on this, we aim to integrate the same locality property into our model architecture. To achieve this, we structure each layer as a convolutional layer, which will ensure that output values are only affected by nearby input values.

For all grid resolutions, we require that the solution operator be the discretization of some coarser heat equation. For this reason, it is more restrictive than a standard convolutional neural network. The convolution kernel will be a diagonal multiple (corresponding to the unknown coarsened coefficients) of the fixed Laplacian operator. The Laplacian convolution kernel corresponds to

$$W_{Lap,1} = \frac{dt}{dx^2}[1, -2, 1], \qquad W_{Lap,2} = \frac{dt}{dx^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

in one and two dimensions, respectively.

5.1.2 *Stability*

As seen in Chapter 3, when solving any PDE numerically, we are bound by some stability constraints that are necessary for obtaining a convergent solution. Using the same procedure as in Chapter 3 we can generalize our result to the case of non-constant coefficients. Then for the heat equation with non-constant coefficients, assuming we take space intervals of dx (and equal in all dimensions) and time intervals of dt, we are bound by the stability constraint $0 \leq a(\boldsymbol{x}) \cdot \frac{dt}{dx^2} \leq \frac{1}{2 \cdot D}$ where D is the dimension of the data [Courant et al., 1967]. Thus, when one knows the coefficients $a(\boldsymbol{x})$, then one can simply pick dt and dx to satisfy the stability constraint.

In this case, we take the opposite approach. Given fixed values of dx and dt, we can bound the coefficients themselves by

$$0 \leq a(\boldsymbol{x}) \leq C_a = \frac{dx^2}{2D \cdot dt}. \tag{5.1}$$

This is a crucial constraint since the parameters of our model will take the place of the coefficients of the equation being modelled. In this way, we design our model precisely with the aim of learning the physical process that we are trying to approximate.

In order to satisfy the stability constraint, we bound the raw parameters learned by the

model with a scaled sigmoid function. This is, if the model's parameters are θ, then the values that we multiply with the output of the convolution layer are given by $C_a \cdot \sigma(\theta)$. This ensures that the parameters are bound by the stability region of the PDE and thus forces the model to find a solution in the parameter space in which the PDE itself is stable.

It is important to note that when coarsening our data to subgrids, the same stability constraint must be satisfied. Thus, we will always coarsen our data according to the same $\frac{dt}{dx^2}$ factor. More precisely, if we coarsen our data in space by a factor of λ_x, we will sample our time steps at intervals of $\lambda_t = \lambda_x^2$.

5.1.3 *Linearity*

Since the differential operator of the heat equation: $\partial_t(\boldsymbol{x}, t) - a(\boldsymbol{x})\Delta u(\boldsymbol{x}, t)$ is linear, we want our model to be linear in the data as well. This is achieved by requiring that our model be linear in the data U, which is not typically the case for neural networks. We achieve this by simply not including non-linear activation functions in our model. However, we note that to satisfy the stability constraints mentioned above, the model is nonlinear in the parameters θ.

5.1.4 *Memory-less recursion*

Our differential operator is time-independent, meaning that no matter what the starting time t_0 is, the physical process is the same. Naturally, we implement this property into our model by making each layer identical, ensuring the same physical process between each predicted time step.

5.2 The Complete Model

Putting all of it together, our model is then a composition $f_\theta(U_0) = l_0 \circ l_0 \circ \cdots \circ l_0(U_0)$ of repeated layers. The layer is defined as (i) the convolution of the data with the fixed (non-learnable) dimension-dependent Laplacian $W_{\text{Lap, dim}}$ defined above, followed by (ii) component-wise multiplication by weights bounded between zero and a fixed, given upper bound (determined by the PDE operator as explained in equation (5.1)), and finally (iii) this update is added back to the input vector U. We note that since the bound on the weights is achieved using a sigmoid nonlinearity, the model is linear in the data U, and nonlinear in the model parameters θ:

$$l_0(U): \quad U \longrightarrow \Big(\text{diag}(C_a \cdot \sigma(\theta))\text{conv}(W_{\text{Lap,dim}}, U)\Big) + U. \tag{5.2}$$

Thus, the number of weights in the model is on the order of the number of grid points (spatial data points) as shown in tables 7.1 and 7.2. The architecture is motivated by domain expertise: the coarsened solution of a time-independent PDE should be captured approximately by an operator which also looks like a coarse solution operator [Pavliotis and Stuart, 2008]. In the case where the PDE coefficients depend on time, we would have a similar structure but with different weights for each layer.

Another way to look at our model architecture is to visualize its resemblance to the forward Euler method for solving PDEs numerically. The core algorithm of the Euler method for solving the heat equation in one dimension is given by [Larsson and Thomée, 2009]

$$U_j^{n+1} = a(x)\lambda\left(U_{j+1}^n - 2U_j^n + U_{j-1}^n\right) + U_j^n, \tag{5.3}$$

where $U_j^n = u(x_j, t_n)$ represents the solution at space point j and time step n, $a(x)$ represents the non-constant coefficients of the equation, and $\lambda = \frac{dt}{dx^2}$.

We can then see that each layer in our model's architecture in equation (5.2) is built to resemble equation (5.3):

- $\lambda(U_{j+1}^n - 2U_j^n + U_{j-1}^n)$ is replaced by our fixed convolution operator $\text{conv}(W_{\text{Lap,dim}}, U)$.
- $a(x)$ is replaced by the bounded model parameters $\text{diag}(C_a \cdot \sigma(\theta))$.
- Adding this update back to U_j^n in equation (5.3) is analogous to adding the update to U in equation (5.2).

Thus, at its core, our model is designed to learn a solution operator that resembles the Euler method but at coarser grids, with coarser (fewer) coefficients as the resolution decreases.

CHAPTER 6

EXPERIMENTS

In this chapter, we describe the experiments conducted. We will dive into how we created the dataset for the experiments and how we define out-of-distribution data. We will also define the two baseline models that we will use to compare our proposed architecture and then define the experiments conducted. This chapter gets down to the mathematical details, detailing how we set up the experiments, compare across models, and interpret our comparison metrics.

6.1 DATASET GENERATION

We construct our training dataset by:

1. Generating m distinct initial conditions $u(\boldsymbol{x}, 0)$ from a fixed Fourier spectrum.
2. Generating n distinct initial conditions $\tilde{u}(\boldsymbol{x}, 0)$ from a *different* fixed Fourier spectrum.
3. Solving for $u(x, t)$ numerically using the forward Euler method (as described in Chapter 3) for $t \leq T$.
4. Averaging down the data in space by a factor of λ and sampling down the data in time by a factor of λ^2 to obtain the data for the subgrid problems.

6.1.1 *Initial conditions*

Our initial conditions $u(\boldsymbol{x}, 0)$ are of the form

$$u(x, 0) = \sum_{i=1}^{10} c_i sin(\pi i x) \quad 0 \leq x \leq 1 \tag{6.1}$$

in 1D, and of the form

$$u((x, y), 0) = \sum_{i=1}^{4} \sum_{j=1}^{4} c_{ij} sin(\pi i x) sin(\pi i y) \quad 0 \leq x \leq 1,\ 0 \leq y \leq 1 \tag{6.2}$$

in 2D, where the coefficients c_i and c_{ij} are obtained from fixed data distributions.
More precisely, we generate the functions through the following procedure:

1. **For the 1D data:** randomly generate a vector $w \in \mathbb{R}^{10}$ where each component of the vector is sampled from a uniform distribution $w_i \sim U[\frac{-0.5}{f(i)}, \frac{0.5}{f(i)}]$. The function $f(i)$ for $i \in \{1, \dots, 10\}$ is fixed, and will determine the decay of the coefficients.

   For the 2D data: randomly generate a matrix $W \in \mathbb{R}^4 \times \mathbb{R}^4$ where each entry of the matrix is sampled from a uniform distribution $W_{ij} \sim U[\frac{-0.5}{f(i,j)}, \frac{0.5}{f(i,j)}]$. Again the function $f(i,j)$ for $i \in \{1, \dots, 4\}, j \in \{1, \dots, 4\}$ is fixed, and will determine the decay of the coefficients.

2. **For the 1D data:** Generate a function $v(x) = \sum_{i=1}^{10} w_i sin(\pi i x)$ based on the generated vector w as Fourier coefficients.

   For the 2D data: Generate a function $v(x) = \sum_{i=1}^{4} \sum_{j=1}^{4} W_{ij} sin(\pi i x) sin(\pi i y)$ based on the generated matrix W as Fourier coefficients.

3. In both cases, normalize the function by setting $u(\boldsymbol{x}, 0) = v(x) \cdot \frac{2}{\max v(x) - \min v(x)}$. This will result in functions of the form (6.1) and (6.2) where the c_i and c_{ij} are just scaled versions of the w_i and W_{ij} respectively.

In this way, every time we run the process we will generate a new initial condition with amplitude of 2. However, each of these functions will be sampled from the same distribution since the w_i and W_{ij} respectively were sampled from the same distributions.

We also note that, by construction, all of our initial conditions (and hence the full solutions) satisfy the Dirichlet boundary conditions $u(0,t) = u(1,t) = 0$ and $u((0,y),t) = u((1,y),t) = u((x,0),t) = u((x,1),t) = 0$ for 1D and 2D respectively.

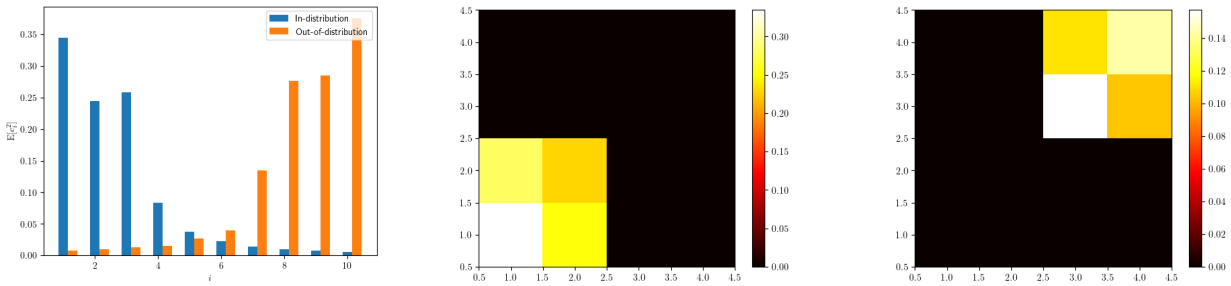
6.1.2 *Out-of-Distribution Data*

Generating the out-of-distribution data is now a simple task. We run the same process explained in the previous section but change the functions $f(i)$ and $f(i,j)$ that define the sampling distributions. In this way, the new data now corresponds to a different Fourier spectrum (distribution of the coefficients) than the original training data.

Thus, it is important to note that here we consider OOD data to be initial data with a different shape (Fourier spectrum) from data previously seen by the models. In a climate model, this would correspond to the problem of having the same physical dynamics, but a different distribution of the density of particles (e.g. a more oscillatory density profile). However, we assume that the solution operator (the coefficients of the equation of the underlying process) remains the same. A different OOD problem that we do not address

in this work would be where the underlying dynamics changed (resulting in different coefficients of the equation modelling the physical process), which corresponds to learning a different solution operator.

We can now visualize the spectra of both data distributions by calculating the empirical expected value of the square of the coefficients $\mathbb{E}[c_i^2]$ and $\mathbb{E}[c_{ij}^2]$ for 1D and 2D respectively. These are shown in Figure 6.1 below.



(a) *Distribution spectra for one-dimensional data*

(b) *In-distribution spectrum for two-dimensional data*

(c) *Out-of-distribution spectrum for two-dimensional data*

Figure 6.1: *Fourier Spectra for in-distribution and out-of-distribution data in one and two dimensions.*

6.1.3 *Solving in the fine grid*

Once our sets of initial conditions are generated, we solve the equation numerically to obtain a solution $u(\boldsymbol{x}, t)$ for all $t \leq T$. For this, we need to define two things first: the PDE to solve (i.e. the coefficients of the heat equation) and the discretization of the fine grid (i.e. define our $d\boldsymbol{x}$ and dt).

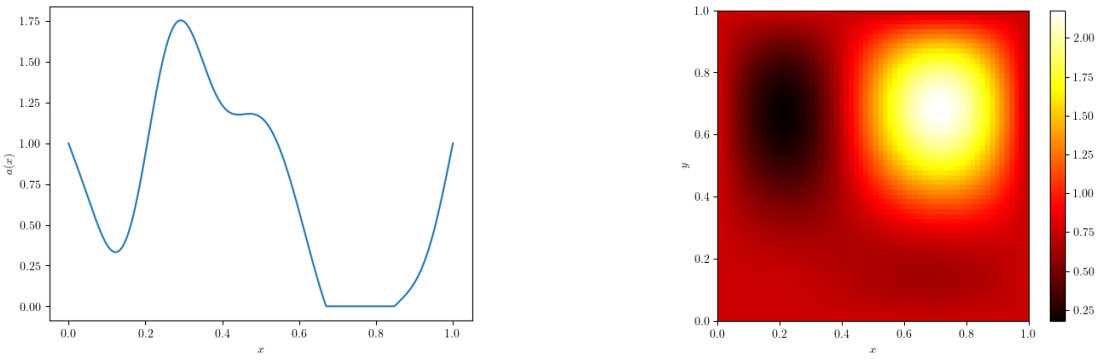
As mentioned several times before, we are solving the PDE: $\partial_t(\boldsymbol{x}, t) - a(\boldsymbol{x})\Delta u(\boldsymbol{x}, t) = 0$, so we need to define our coefficients $a(\boldsymbol{x})$ before being able to solve the equation numerically. We define the coefficients randomly by using the same procedure as we used to generate our initial conditions with some minor adjustments:

1. First, we generate another instance of a possible initial condition.
2. Second, we add 1 to all of the coefficients.
3. Finally, we bound the coefficients between ϵ and $2.5 - \epsilon$ for a small constant ϵ.

Step 3 is necessary since we need to know the maximum bound on the coefficients in order to solve numerically under a stable algorithm. Step 2 is just done so that the coefficients are more meaningful (otherwise we would have a lot of zeros). This bound on the coefficients is precisely the bound that will be implemented in our model parameters in equation (5.1).

More precisely, we will set $C_a = 2.5$. We will see the importance and effect of this bound in Section 7.6.

Below we show the coefficients generated, and for which the experiments were carried on.



(a) *1D heat equation* **(b)** *2D heat equation*

Figure 6.2: *Coefficients of the ground truth process.*

We now define our space domain to be $[0, 1]$ in 1D and $[0, 1]^2$ in 2D, which is consistent with the Dirichlet boundary conditions described before. We then discretize our space grid by making it have $N_{\boldsymbol{x}} = 256$ points in the 1D case and $N_{\boldsymbol{x}} = 64 \times 64$ points in the 2D case. Hence, in 1D we get $dx = \frac{1}{255}$ and in 2D we get $dx = dy = \frac{1}{63}$. Now, according to the stability constraints from Chapters 3 and 5, we must choose dt small enough so that $dt \leq \frac{dx^2}{2D \max a(\boldsymbol{x})}$ [Courant et al., 1967], where $\max a(\boldsymbol{x}) = 2.5$ by construction and D is the dimension of the data so $D \in \{1, 2\}$. To allow for leniency, we chose $dt = 3.05 \cdot 10^{-6}$ for the 1D case and $dt = 2.44 \cdot 10^{-5}$.

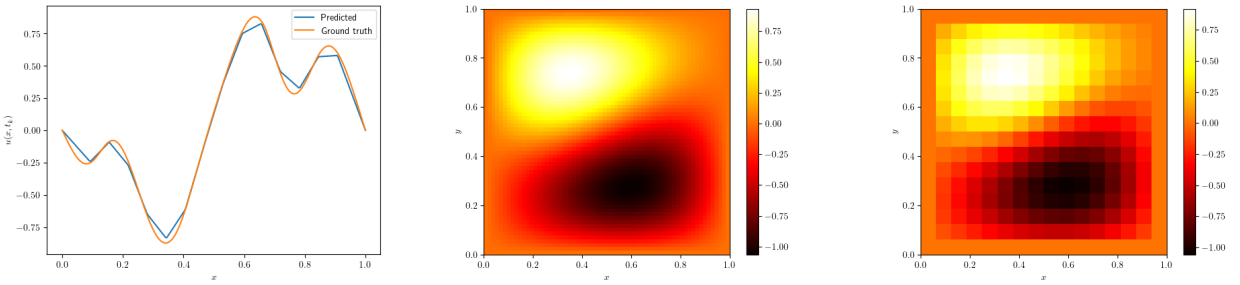
Having chosen dx and dt that guarantee stability, we now solve the PDE numerically to obtain a solution $u(\boldsymbol{x}, t)$ for $t \leq T$, where T must be large enough to allow for downsampling in time for the subgrids (explained in the next section). For our experiments, $T = 0.002$ and $T = 0.0156$ are sufficient for the one- and two- dimensional experiments respectively.

Now for the fully resolved grid, we simply train our model with the data generated. More precisely, the initial conditions are our inputs, and the solutions at the first k time steps are our outputs. For the results presented, $k = 10$.

6.1.4 *Subgrid Data*

To obtain the coarse data for the subgrid problems, we average our data in space and sample it in time according to the stability constraints described before. We take our data and average it down in space by a factor of λ_x (in each dimension) and sample it down in

time by a factor of $\lambda_t = \lambda_x^2$. More precisely, the subgrid data has a dimension of $\frac{N_{\boldsymbol{x}}}{\lambda_x^D}$ where $D \in \{1, 2\}$ is the number of space variables and every time step is $\lambda_t \cdot dt$ apart where dt is the original, fine grid time interval. An example is shown below where we average our data by a factor of 16 in 1D and 4 in 2D.



(a) *1D data in fine and coarse grid (sizes* 256 *and* 16*)*

(b) *2D data in fine grid of size* 64×64

(c) *2D data in coarsened grid of size* 16×16

Figure 6.3: *Example of subgrid coarsening in one and two space variables.*

We note that by construction, the subgrids still follow the same stability constraints as the original fine grid since we averaged down in space and time by the proportional factors (i.e. the ratio $\frac{dt}{dx^2}$ remained the same). This will play a crucial role in our model learning as described in the results section later on.

We also note that we manually set the boundary end points to zero to satisfy the boundary conditions (i.e. just for the endpoints we set them manually to zero, rather than the average of their neighbouring points).

We will apply this subgrid coarsening to both the training/test data and the out-of-distribution data, and run the experiments on both data sets. For each subgrid, our data is thus our coarsened initial conditions as inputs, and our coarsened solutions at the corresponding (down-sampled) first k time-steps as outputs.

6.2 Baseline Neural Network Models

We are now in a position to define our off-the-shelf ML models for comparison. We conduct the experiments for both our proposed model architecture and for two baseline models which are: (1) a standard fully connected 2-layer ReLU neural network (FCN), and (2) a standard convolutional 2-layer ReLU neural network (ConvN). We chose these models given their simplicity and as a proxy for off-the-shelf ML models.

6.2.1 *FCN*

The fully connected network consists of 10 layers, each being a multilayer perceptron with a single hidden layer of size 32 and ReLU activation (similar to Figure 2.1 but with only one hidden layer). We note that for the 2D case, before feeding the data to each MLP, we must first flatten it to a 1D vector (so for example an 8×8 grid would be flattened to a vector of length 64). After passing the flattened vector through the MLP, we re-shape it back to its original dimensions.

It is important to note that layers are not repeated in this network, and the model learns different weights for every MLP.

6.2.2 *ConvN*

The convolutional network consists of 10 layers, each being a simple 2-layer convolutional neural network with 3×3 kernel, ReLU activation, and hidden layer with 16 channels, as shown in Figure 6.4 below.

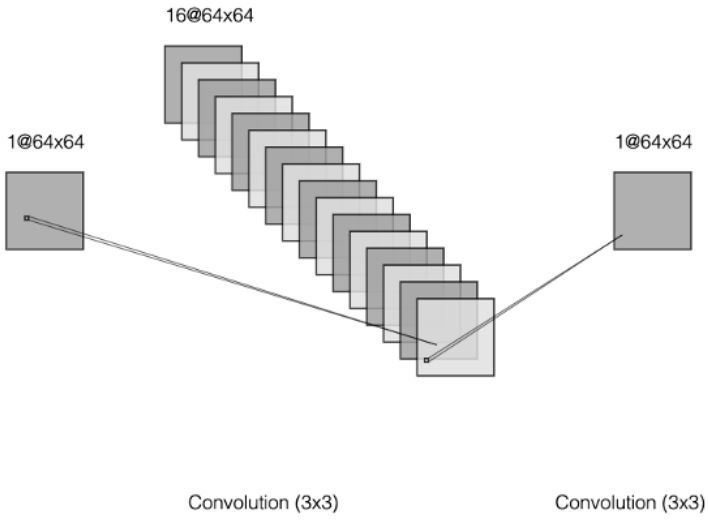


Figure 6.4: *Convolutional network corresponding to each layer of the ConvN model [LeNail, 2019].*

Again, we note that layers are not repeated in this network, and the model learns different weights in every layer. Logically, we apply 2D convolutions for the two-dimensional case and 1D convolutions for the one-dimensional case.

6.3 Error Measures

Since the norm of the solutions might vary within the dataset (and more importantly between the training/test data and the out-of-distribution data), we measure the error as the relative L^2 error with respect to the norm of the solutions, using a normalization that sets the variance of the initial data (as a function of $\boldsymbol{x}$) to be 1. Thus, the relative errors plotted in Figure 7.1 in the next chapter are calculated as the average normalized error of the predicted solutions. More precisely, we calculate

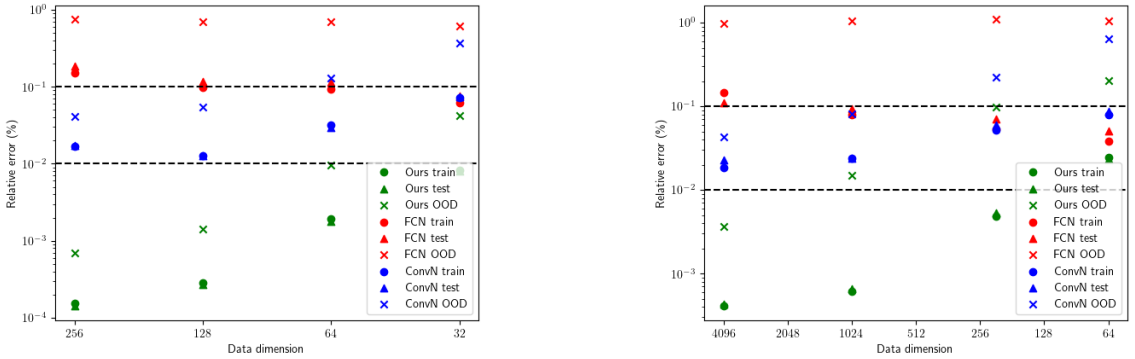
$$\epsilon = \left(\frac{1}{N} \sum_{i=1}^{N} \frac{\| f_\theta(u_i(\boldsymbol{x}, 0)) - u_i(\boldsymbol{x}, t) \|^2}{\sigma_\rho^2} \right)^{\frac{1}{2}},$$

where $\sigma_\rho^2 = \mathbb{E}\left[\int_0^T \int_{\boldsymbol{x}} \left(u(\boldsymbol{x}, t) - \bar{u}(\boldsymbol{x}, t) \right)^2 d\boldsymbol{x} dt \right]$ is a normalization factor that sets the variance of the initial data (as a function of $\boldsymbol{x}$) to be 1, and allows us to do a fair comparison across distributions. In short, we are measuring how far off, percentage-wise, the predicted solution is from the ground truth solution.

CHAPTER 7

RESULTS AND DISCUSSION

We conducted the experiments for 4 subgrids of varying resolution in both 1D and 2D, and Figure 7.1 below shows the results obtained. We can see from the figure that the FCN achieves 10% training and test error, nearly constant across grid resolutions. The error decreases slightly as the grid gets coarser, which is contrary to what we would expect. However, we note that as the grids get coarser, the models also have fewer parameters, as shown in tables 7.1 and 7.2. This is an indication that the FCN might be overfitting to the training data, and as the model parameters decrease, the overfitting is reduced. Nevertheless, the test error is not significantly larger than the training error, indicating that overfitting is not the main problem with the model. On OOD data, however, the FCN performs poorly, with a relative error close to 100%. Thus, the model is unable to adapt to distinct distributions and is learning aspects of the dataset itself rather than features and properties of the underlying physical process.



(a) *One space variable* **(b)** *Two space variables*

Figure 7.1: *In-distribution and out-of-distribution relative errors for subgrid models in one and two dimensions.*

Subgrid resolution	256	128	64	32
Parameters in our model	256	128	64	32
Parameters in FCN	166, 720	83, 520	41, 920	21, 120
Parameters in ConvN	1, 130	1, 130	1, 130	1, 130

Table 7.1: *Model parameters for our model and the standard neural networks in one dimension.*

Subgrid resolution	$4,096$	$1,024$	256	64
Parameters in our model	$4,096$	$1,024$	256	64
Parameters in FCN	$2,662,720$	$665,920$	$166,720$	$41,920$
Parameters in ConvN	$3,050$	$3,050$	$3,050$	$3,050$

Table 7.2: *Model parameters for our model and the standard neural networks in two dimensions.*

On the other hand, our model maintains high accuracy, with less than 1% training and test error, except on the coarsest grid in two dimensions. More importantly, on the out-of-distribution data, the model is also quite accurate, with below 10% error on all subgrid problems except the coarsest grid in two dimensions, which is just slightly higher. Thus, we have a 10 fold improvement in distribution versus the FCN and success versus failure on out-of-distribution data. This is a remarkable result since most current PDE models fail on OOD data, even though it is a very desirable property in areas like climate and weather modelling [Kashinath et al., 2021].

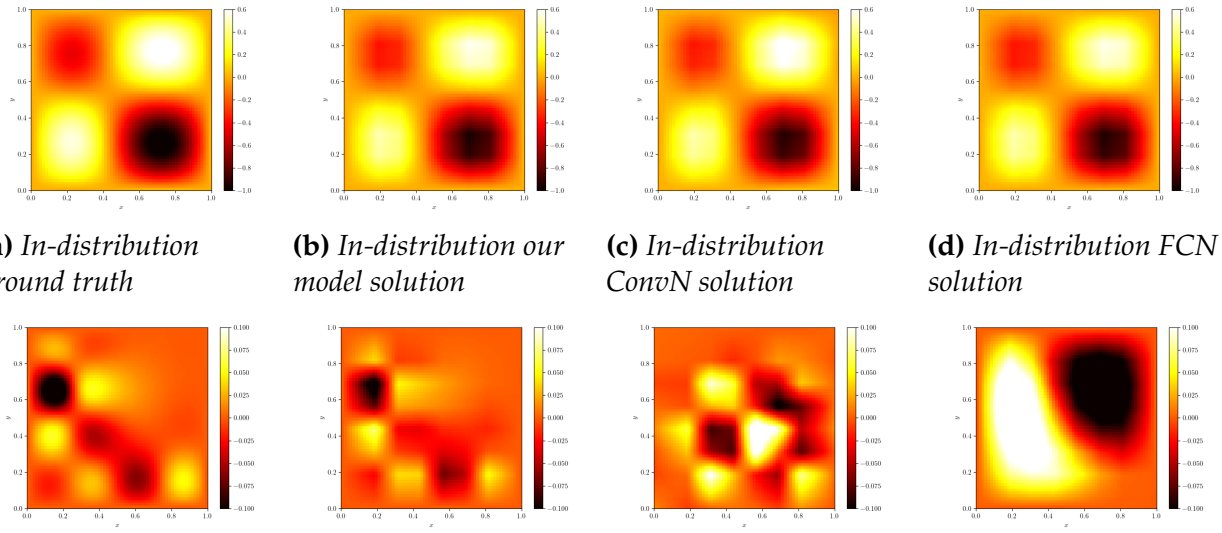
As for the ConvN model, we observe that it performs significantly better than the FCN for both in-distribution and out-of-distribution data, but it still underperforms significantly compared to our model. This result is not surprising, since the ConvN model contains one of the properties of our model: locality (created by the convolution layers), while the FCN shares none of the properties of our model. This reinforces the claim that incorporating physics knowledge of the underlying processes into the ML model framework creates better-performing models for simulating physical processes. In this particular case, having the locality property, which is inherent to PDEs, helps the ConvN outperform the FCN, while it still underperforms with respect to our model, which has an additional 3 properties derived from knowledge of the process being modelled.

7.1 Modelled solutions

We now show some of the modelled solutions across all three models tested. Figures 7.2 and 7.3 show an instance of the predicted solutions for both our model and the standard neural networks for both in-distribution and out-of-distribution examples. Figure 7.2 shows an example of a two-dimensional subgrid problem with resolution 256 (16×16) where we recall that the original fine grid had a resolution of 4096 (64×64). Interestingly, we can observe that even though Figure 7.1 showed that our model is around 10 times more accurate on average, both neural networks' relative error is still good enough to produce a visually similar solution for in-distribution data.

On OOD data, however, it is visually clear that the fully connected neural network does not learn an accurate solution operator and tries to adapt the new data to what it was trained on. Our model, on the other hand, can adapt to the new distribution with high accuracy, producing a solution that is visually similar in structure to the ground truth solution. As expected, the convolutional network fares in between, producing a solution with a similar structure but less accurate than our model and visually different from the ground truth solution.

We note that to compare accurately with the original solution in the fine grid, the subgrid-generated solutions by the models were linearly interpolated back to the original, fine grid resolution.



(a) *In-distribution ground truth*

(b) *In-distribution our model solution*

(c) *In-distribution ConvN solution*

(d) *In-distribution FCN solution*

(e) *Out-of-distribution ground truth*

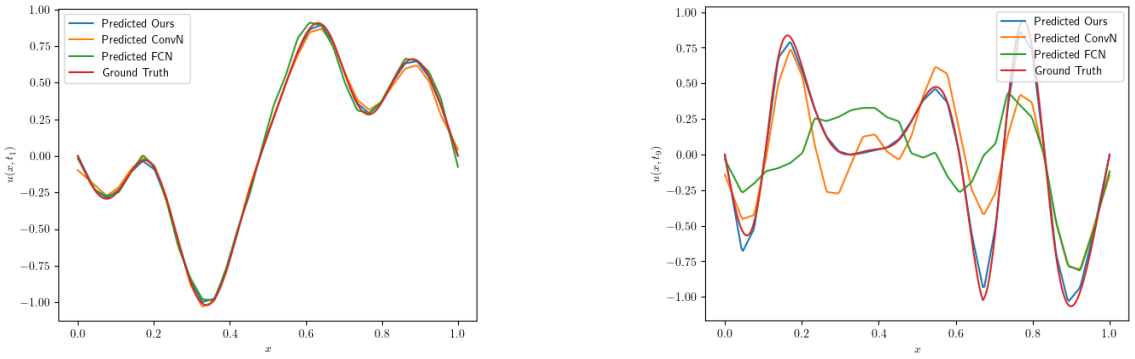
(f) *Out-of-distribution our model solution*

(g) *Out-of-distribution ConvN solution*

(h) *Out-of-distribution FCN solution*

Figure 7.2: *Two dimensional modelled solutions for an in-distribution and out-of-distribution example in a subgrid with a resolution of* 16×16.

Figure 7.3 shows an instance of a modelled solution in one dimension in a subgrid problem with a resolution of 32, where we recall that the original fine grid had a resolution of 256. Again, we observe the same phenomenon, with all models being able to produce a curve very close to the actual solution for in-distribution data (with our model being the most accurate) but only our model being able to adapt to out-of-distribution data accurately (with the FCN completely failing and the ConvN replicating some structure but failing to capture the full dynamics).

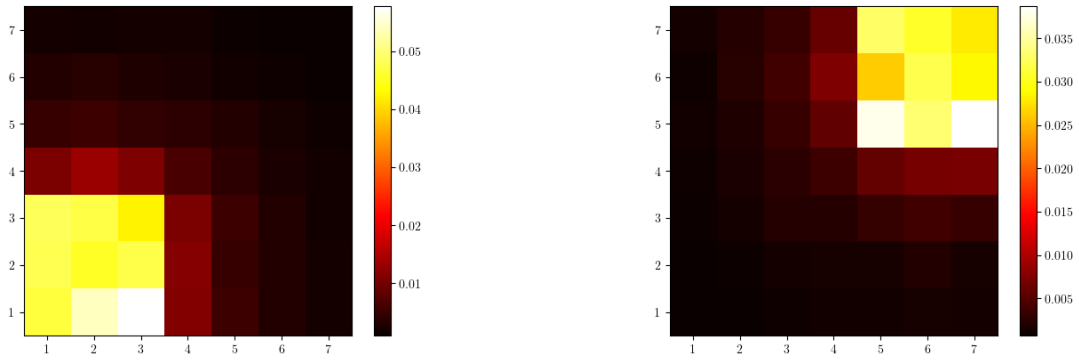


(a) *In-distribution solutions* **(b)** *Out-of-distribution solutions*

Figure 7.3: *One dimensional modelled solutions for an in-distribution and out-of-distribution example in a subgrid with a resolution of* 32.

7.2 Data complexity

In addition to the standard experiments, we also performed an ablation study on the data complexity used in the model. To do this, we changed our functions $f(i)$ and $f(i,j)$ that define our data-generating process to create data with more complex Fourier spectra. Figure 7.4 shows the empirical Fourier spectra for the in-distribution and out-of-distribution data used in the ablation study, which shows a jump in data complexity compared to the spectra used for the main results in Figure 6.1.

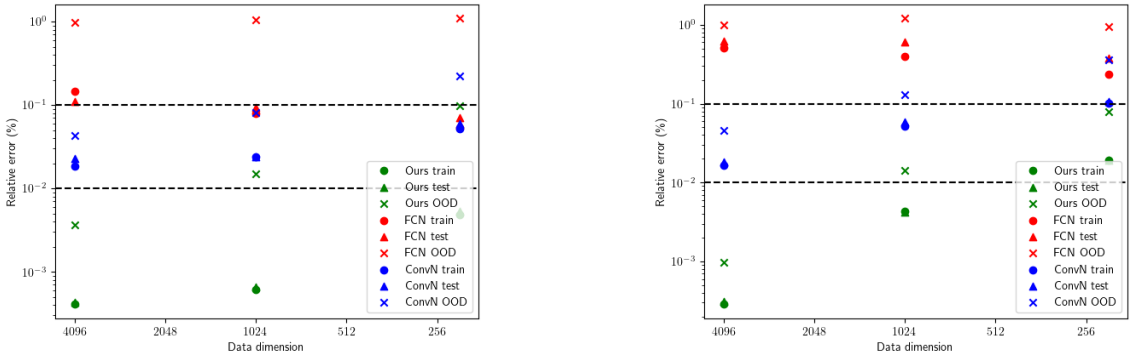


(a) *In-distribution spectrum* **(b)** *Out-of-distribution spectrum*

Figure 7.4: *Fourier spectra for in-distribution and out-of-distribution data in the ablation study.*

Figure 7.5 below shows the numerical results for the experiments under the complex data spectrum compared to those under the simple data spectrum (conducted before). We can see that under the new data spectrum, the fully connected neural network was unable to learn an accurate solution operator, with relative errors of around 50%. On the other hand, our model exhibited a stable pattern across both sets of data, demonstrating that it is resilient to changes in the data complexity. As usual, the convolutional network

stood in the middle, losing some accuracy when translating to the complex spectrum but performing significantly better than the fully connected network. We note that at this level of data complexity, it was not possible to resolve the data at the coarsest resolution, so we stopped at 256.

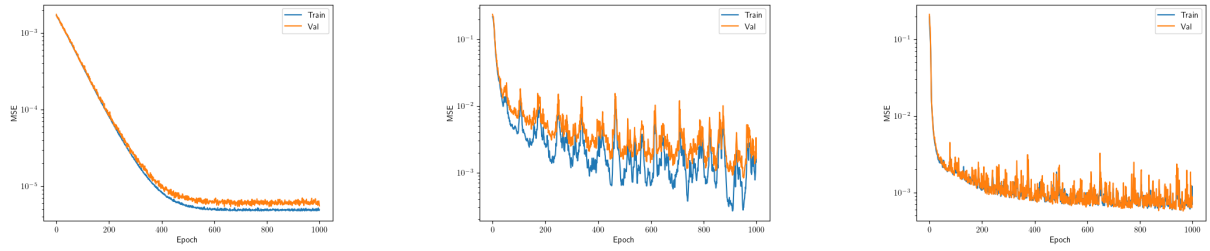


(a) *Simple data spectrum* **(b)** *Complex data spectrum*

Figure 7.5: *Subgrid errors for both simple Fourier spectra and complex Fourier spectra.*

7.3 TRAINING DYNAMICS

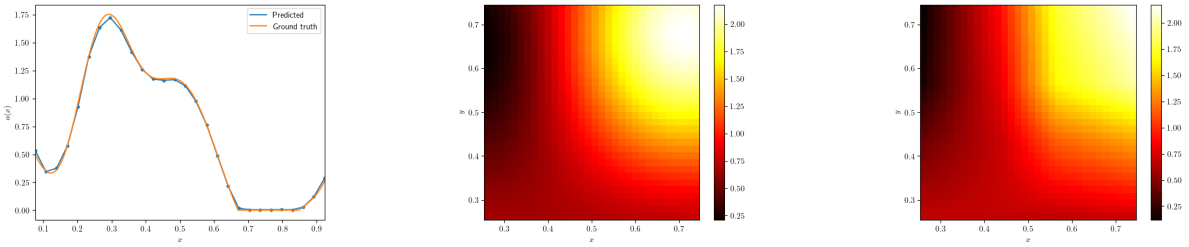
As remarkable as the results themselves, the training dynamics of the models display insights into the benefits of our model architecture and the incorporation of domain knowledge. Figure 7.6 displays a sample of the training dynamics for all three models. It shows that our model's dynamics are a lot smoother than those of the standard neural networks, leading to less volatility in the training and a more stable model. Furthermore, we recall that tables 7.1 and 7.2 showed that our model had significantly fewer parameters than the standard neural networks (as a function of the subgrid size), which is desirable for computational efficiency.



(a) *Training dynamics of our model* **(b)** *Training dynamics of the FCN model* **(c)** *Training dynamics of the ConvN model*

Figure 7.6: *Training dynamics of our model and the standard neural networks in a two-dimensional subgrid of resolution* 16×16.

Moreover, when we step inside to take a look at the learned parameters of our model, we can compare them with the actual coefficients of the equation being modelled. Figure 7.7 shows the learned coefficients of our model for a given subgrid compared to the actual coefficients of the heat equation modelled (where we again linearly interpolated back to the original grid size for comparison). We can see that the coefficients learned are very similar to those of the ground truth process, looking like a sort of (non-linear) average of them. This is not entirely surprising, since we expected the optimal solution operator in the coarsened grids to behave like a coarsened version of the heat equation. This underscores another advantage of our model architecture, which, by construction, has parameters that act as the coefficients of a coarsened solution operator.



(a) *Coefficients in 1D* **(b)** *Actual coefficients in 2D* **(c)** *Predicted coefficients in 2D*

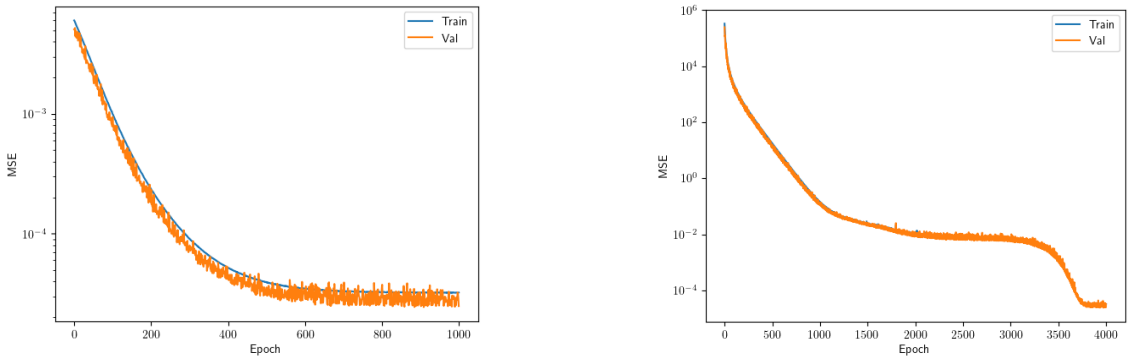
Figure 7.7: *Learnt coefficients of our model and the standard neural networks in a one-dimensional subgrid of resolution 32 and a two-dimensional subgrid of resolution* 16×16.

7.3.1 *Stability*

Now, we discuss probably the most fascinating result of all: How the training dynamics of the model relate to the stability constraints embedded in its architecture. Recall that the coefficients in our model were bounded between ϵ and $2.5 - \epsilon$ to ensure the stability of the solution operator. These same bounds are applied to our model architecture by bounding the coefficients in each layer through the constant $C_a = 2.5$ in equation (5.1). This means that the parameter space of our model is the same as the stability space of the original PDE. In other words, our model is constrained to look for a solution within a "stable" set of parameters. How important this bound is becomes clear when we change its value.

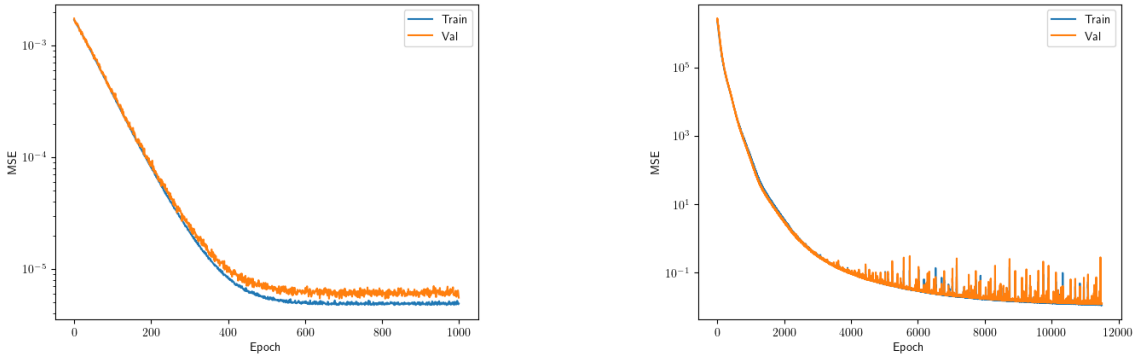
If we change C_a to different values, we are essentially allowing the model to search outside the stable parameter space. Figure 7.8 below shows a specific instance of the training dynamics for the 1D model with a subgrid size of 32 in which $C_a = 10$ (compared to the original $C_a = 2.5$). We can see that allowing the model to search a wider parameter space made it take longer to converge. In fact, it almost seemed to have reached a suboptimal solution between $2,000$ and $3,000$ epochs (and one could have even assumed convergence

by then), before moving towards the optimal solution. This suggests that searching in an unstable region might lead to suboptimal solutions. Figure 7.9 below shows the training dynamics for the 2D model in a subgrid of size 16×16 with $C_a = 2.5$ (stable) and $C_a = 10$ (unstable). We can see that the model converges to a sub-optimal solution under the unstable regime.



(a) $C_a = 2.5$ *(stable)* **(b)** $C_a = 10$ *(unstable)*

Figure 7.8: *Training dynamics for different bounding constants in 1D.*



(a) $C_a = 2.5$ *(stable)* **(b)** $C_a = 10$ *(unstable)*

Figure 7.9: *Training dynamics for different bounding constants in 2D.*

We now fixed the subgrid size (32 for 1D and 16×16 for 2D) and repeated the experiments with different values of C_a and also by changing the sigmoid function to a tanh function (to allow for negative values). In tables 7.3 and 7.4 below, we show the epochs required for convergence for both activations and bounding constants. Where no value is shown it means that the model did not converge to the optimal solution. We also conduct the experiment without any activation function (and hence the bounding constant is irrelevant in this case since the parameters are unbounded).

Activation	C_a	Epochs to optimal convergence
sigmoid	2.5	600
sigmoid	5	1,000
sigmoid	10	3,700
tanh	2.5	1,000
tanh	5	2,000
tanh	10	3,200
identity	–	2,000

Table 7.3: *Training dynamics for the 1D model with different bounding conditions.*

Activation	C_a	Epochs to optimal convergence
sigmoid	2.5	600
sigmoid	5	1,100
sigmoid	10	–
tanh	2.5	1,600
tanh	5	3,500
tanh	10	–
identity	–	3,700

Table 7.4: *Training dynamics for the 2D model with different bounding conditions.*

We can see that allowing the bounding constant to increase (and assuming random and uniform initialization of parameters) leads to slower convergence and can even lead to sub-optimal convergence of the model. This is a very interesting result since it shows that PDE models should search in the appropriate "stable" parameter space to guarantee the best results and computational efficiency.

An interesting case is when we have no activation function at all. While the model takes longer to converge, it does always converge to the optimal solution. This is likely due to the linearity of the solution operator modelled (heat equation) and the fact that removing the activation function makes the model linear in the parameters. This does not, however, diminish the importance of the result found for non-linear activations, since 1) almost all ML models are non-linear and 2) other climate and weather processes are non-linear (e.g. the Navier stokes equations [Rind, 1999]).

CHAPTER 8

CONCLUSION

In this work, we have explored the relationship between ML models, PDEs, and subgrid modelling, shedding light on the technicalities of their interplay and their potential in areas like climate and weather modelling. Our findings underscore the significant promise that neural PDE solvers hold, especially when equipped with the robust controls traditionally associated with PDE solvers.

We introduced a novel neural network architecture that is firmly grounded in theory and meticulously designed to uphold the physical constraints inherent to the PDE. This architecture consistently outperformed standard models and has three properties desired for climate and weather modelling:

1. **Accurate subgrid modelling:** Our proposed model outperformed the standard ML models in every subgrid tested.

2. **Satisfying physical constraints:** Our model satisfies the physical constraints of the process being modelled (represented by the boundary conditions of the PDE), by construction.

3. **Out of distribution generalization:** Our model succeeds in out-of-distribution generalization as opposed to the other basic ML models tested.

While our model just tests one physical process (the heat equation), it serves as a proof-of-concept that integrating domain knowledge of the physical process being modelled directly into the ML model's architecture can lead to better results in subgrid modelling and even out-of-distribution generalization. Furthermore, we also discovered that our model was able to adapt to more complex data distributions when compared to the standard ML models. This is also of great importance for climate and weather modelling since weather patterns are often represented by complex dynamics [Flato et al., 2014].

Finally, we discovered that the training dynamics of the model are intricately related to the stability of the PDE being modelled. More precisely, we found that by embedding stability knowledge in the model architecture itself, we can speed up convergence, which is not even guaranteed if we omit these crucial stability bounds.

While our results are fascinating, there are also limitations to our work. First, we only compare our model with very basic standard ML models (a 2-layer fully connected network and a 2-layer convolutional network). An interesting next step that would be of more interest to current research would be to test it against current state-of-the-art ML PDE models such as PINNs [Raissi et al., 2019]. Second, we only address one specific problem: the heat equation with non-constant coefficients. While this problem is simple enough to demonstrate our results and complex enough to outline their importance, the next step would be to expand the model architecture to general advection-diffusion equations, and later to all parabolic PDEs. Finally, our data is artificially constructed, and it would be interesting for future research to test our models with real datasets. Nevertheless, our research serves as a proof of concept that incorporating domain knowledge of PDEs directly into the model's architecture can lead to more efficient and accurate PDE solvers for subgrid modelling.

In summary, this thesis contributes to the growing field of scientific machine learning by showcasing a successful integration of machine learning with domain-specific knowledge in PDEs and subgrid modelling. The proposed model not only demonstrates improved efficiency and accuracy in solving PDEs but also lays a foundation for future research to build upon, with the potential to revolutionize the way complex physical phenomena are modelled and understood.

BIBLIOGRAPHY

V. Balaji, F. Couvreux, J. Deshayes, J. Gautrais, F. Hourdin, and C. Rio. Are general circulation models obsolete? *Proceedings of the National Academy of Sciences*, 119(47): e2202075119, 2022.

T. Bolton and L. Zanna. Applications of deep learning to ocean data inference and subgrid parameterization. *Journal of Advances in Modeling Earth Systems*, 11(1):376–399, 2019.

G. P. Brasseur and D. J. Jacob. *Parameterization of Subgrid-Scale Processes*, page 342–398. Cambridge University Press, 2017. doi: 10.1017/9781316544754.009.

C. S. Bretherton, B. Henn, A. Kwa, N. D. Brenowitz, O. Watt-Meyer, J. McGibbon, W. A. Perkins, S. K. Clark, and L. Harris. Correcting coarse-grid weather and climate models by machine learning from global storm-resolving simulations. *Journal of Advances in Modeling Earth Systems*, 14(2):e2021MS002794, 2022. doi: https://doi.org/10.1029/2021MS002794. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021MS002794. e2021MS002794 2021MS002794.

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM journal of Research and Development*, 11(2):215–234, 1967.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

G. Flato, J. Marotzke, B. Abiodun, P. Braconnot, S. C. Chou, W. Collins, P. Cox, F. Driouech, S. Emori, V. Eyring, et al. Evaluation of climate models. In *Climate change 2013: the physical science basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*, pages 741–866. Cambridge University Press, 2014.

E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, dec 2017.

X. Han, H. Gao, T. Pfaff, J. Wang, and L. Liu. Predicting physics in mesh-reduced space with temporal attention. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=XctLdNfCmP.

F. Joel. Solution of the heat equation by separation of variables. *Lecture Notes of Math 267, Department of Mathematics at the University of British Columbia*, 2007.

J. Kaipio and E. Somersalo. *Statistical and computational inverse problems*, volume 160. Springer Science & Business Media, 2006.

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

K. Kashinath, M. Mustafa, A. Albert, J. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh, et al. Physics-informed machine learning: case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A Mathematical Physical and Engineering Sciences*, 379(2194):20200093, 2021.

A. A. Kiaei, M. Boush, D. Safaei, S. Abadijou, N. Baselizadeh, N. Salari, and M. Mohammadi. Active identity function as activation function. *Preprints*, 2023.

P. Kidger and T. Lyons. Universal approximation with deep narrow networks. In *Conference on learning theory*, pages 2306–2327. PMLR, 2020.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

R. Kruse, S. Mostaghim, C. Borgelt, C. Braune, and M. Steinbrecher. Multi-layer perceptrons. In *Computational intelligence: a methodological introduction*, pages 53–124. Springer, 2022.

S. Larsson and V. Thomée. *Partial Differential Equations With Numerical Methods*, volume 45. Springer, Chalmers University of Technology and University of Gothenburg 412 96 Göteborg Sweden, 2009.

J. Lee and M. S. Jeong. Stability of finite difference schemes on the diffusion equation with discontinuous coefficients. *Massachusetts Institute of Technology, Cambridge*, 2017.

A. LeNail. Nn-svg: Publication-ready neural network architecture schematics. *J. Open Source Softw.*, 4(33):747, 2019.

L. R. Leung and S. Ghan. A subgrid parameterization of orographic precipitation. *Theoretical and Applied Climatology*, 52:95–118, 1995.

R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 10 2020.

X.-Y. Liu, H. Sun, M. Zhu, L. Lu, and J.-X. Wang. Predicting parametric spatiotemporal dynamics by multi-resolution pde structure-preserved deep learning. *arXiv preprint arXiv:2205.03990*, 2022.

Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-net: Learning PDEs from data. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3208–3216. PMLR, 10–15 Jul 2018.

V. A. Marchenko and E. Y. Khruslov. *Homogenization of partial differential equations*, volume 46. Springer Science & Business Media, 2008.

R. Mcsweeney and Z. Hausfather. Q&a: How do climate models work. *Carbon Brief*, 2018. URL https://www.carbonbrief.org/qa-how-do-climate-models-work/.

M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

R. Olof. Heat Equation. *Lecture Notes of DN2255, KTH computer science and communication*, 2013.

G. Pavliotis and A. Stuart. *Multiscale methods: averaging and homogenization*. Springer Science & Business Media, 2008.

T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=roNqYL0_XP.

N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5301–5310. PMLR, 09–15 Jun 2019.

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

S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the National Academy of Sciences*, 115(39):9684–9689, 2018.

D. Rind. Complexity and climate. *science*, 284(5411):105–107, 1999.

L. Ruthotto and E. Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, 62(3):352–364, Apr 2020.

S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

Y. Shin, J. Darbon, and G. E. Karniadakis. On the convergence and generalization of physics informed neural networks. *arXiv e-prints*, pages arXiv–2004, 2020.

K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

A. M. Stuart. Inverse problems: a bayesian perspective. *Acta numerica*, 19:451–559, 2010.

M. Tekriwal, K. Duraisamy, and J.-B. Jeannin. A formal proof of the lax equivalence theorem for finite difference schemes. In *NASA Formal Methods Symposium*, pages 322–339. Springer, 2021.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

O. Watt-Meyer, N. D. Brenowitz, S. K. Clark, B. Henn, A. Kwa, J. McGibbon, W. A. Perkins, and C. S. Bretherton. Correcting weather and climate models by machine learning nudged historical simulations. *Geophysical Research Letters*, 48(15):e2021GL092555, 2021. doi: https://doi.org/10.1029/2021GL092555. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021GL092555. e2021GL092555 2021GL092555.

J. A. Weyn, D. R. Durran, and R. Caruana. Can machines learn to predict weather? using deep learning to predict gridded 500-hpa geopotential height from historical weather data. *Journal of Advances in Modeling Earth Systems*, 11(8):2680–2693, 2019. doi: https://doi.org/10.1029/2019MS001705. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2019MS001705.