BALANCING DATA FITTING AND PHYSICAL PROPERTIES: A COMPARATIVE STUDY ON PHYSICS LOSS COEFFICIENTS AND FOURIER ANALYSIS TECHNIQUES IN PINO MODELS FOR PDES

by

Davian R. Chin

u2204075@uel.ac.uk

A Dissertation

Submitted to the School of Architecture, Computing and Engineering

University of East London

In Partial Fulfillment of the Requirements

for the Degree Master of Science (MSc)

March 2023

Table of Contents

List of figures

List of tables

List of abbreviations, nomenclature, and symbols

PDE	Partial Differential Equation
FDM	Finite Difference Method
FFT	Fast Fourier Transform
DFT	Discrete Fourier Transform
DeepONet	Deep Operator Network
FNN	Fourier Neural Network
FNO	Fourier Neural Operator
IFNO	Implicit Fourier Neural Operator
PINN	Physics Informed Neural Network
PINO	Physics Informed Neural Operator
MSE	Mean Squared Error
GELU	Gaussian Error Linear Units
SGD	Stochastic Gradient Descent
ADAM	Adaptive Movement Estimation

Abstract:

Partial differential equations (PDEs) are a mathematical foundation for modelling physical processes. However, solving complex PDEs using traditional numerical simulations can be computationally expensive and limited in capturing intrinsic random phenomena. Neural networks offer an alternative for solving physical problems but often require extensive training data generated from costly simulations. Physics-Informed Neural Networks (PINNs) provide a promising surrogate model by combining numerical simulation methods with neural networks and demonstrate remarkable accuracy and efficiency in solving parameterized PDEs, even in incomplete data or ill-posed problem statements. However, PINNs face challenges in complex scenarios involving multi-physics, hyperparameter modification, and model generalization.

This paper introduces a novel approach based on Physics-Informed Neural Operators (PINOs) that integrates Fourier analysis techniques into the PINO framework, along with a custom loss function that combines the residual loss from the neural operator network and loss from enforcing physical conservation laws governing PDEs, and Fourier-based derivatives and a physics loss coefficient. By leveraging the strengths of numerical algorithms to generate training data and visualize experimental results, we present a novel design for a PINO framework and a comprehensive comparative study on the impact of varying the physics loss coefficient on the performance of the PINO models for solving PDEs.

Keywords: Neural networks, neural operators, partial differential equations, physics informed neural networks, physics informed neural operators, deep learning, scientific computing.

1. Introduction

In recent years, machine learning has achieved remarkable success in various fields due to the explosive growth of computing power (CPUs and GPUs) and big data technologies. However, until now, traditional machine learning algorithms have yet to fully use the underlying physics embedded in the problems they attempt to solve and ignore natural laws intrinsic to physical processes. On the other hand, traditional numerical computing methods, such as finite difference and finite element methods, have embedded the physical laws governing physical processes but have not used data generated from direct observations and measurements of physical systems (You et al., 2022, p. 2, Ayensa-Jiménez et al., 2020). Head-to-head, traditional solvers discretise the space, which leads to a trade-off between speed and accuracy. At the same time, data-driven methods can learn the trajectory of the equations directly from the data, making them much faster than traditional solvers. Furthermore, complex PDE systems require a fine discretisation, which makes them challenging and time-consuming for traditional solvers, while data-driven methods can be orders of magnitude faster (Li et al., 2021).

Physics informed neural network (PINN), proposed by (Raissi et al., 2019), solves PDE associated with physical processes using both the residual of the PDE and the initial and boundary conditions as loss functions. According to Vadyala et al. (2022), PINNs can solve supervised learning problems associated with physical processes while considering physical laws encoded into partial differential equations (PDEs). In the training process, the network minimises the residual of the PDE and constrains the final solution to the initial and boundary conditions (Oldenburg et al., 2022, p. 6-8, Wang et al., 2021, p. 5). PINNs present a way to harness the powerful predictive ability of data-driven models and the scientific rigour and explainability of physics-based models to generate highly accurate and meaningful solutions

to scientific problems (Ayensa-Jiménez et al., 2020, Wang, Bentivegna et al., 2020). PINNs have been gaining popularity in the scientific research community because of their potential to solve a wide variety of problems (Wang et al., 2022) without the need for expensive discretisation processes (Oldenburg et al., 2022, p.2) and their power to deal with complex physical problems where the governing equations are often known. However, analytical solutions may not be known (You et al., 2022). The first breakthrough in using neural networks for physical problems was the Physics Informed Neural Network (PINN) introduction by Raissi et al. (2019). Raissi et al. (2019) used a PINN to solve the forward and inverse problems involving nonlinear partial differential equations. Raissi et al. (2019) used a neural network to represent the solution to the differential equation and then used automatic differentiation to calculate the residual of the differential equation. They then used the PINN to solve physical problems in applied areas such as fluid mechanics and wave mechanics.

Further work in the field illustrates the usefulness of PINNs in science and engineering. Katsikis et al. (2022) used PINNs to solve partial differential equations associated with the force distribution in a rigid cylindrical rod fixed at one end and variations of the Cantilever Beam problem with tremendously high accuracy. Wang et al. (2020a) used a super-resolution (SR) technique based on a neural network constrained by the advection-diffusion equation representing governing physics laws to reconstruct high-resolution images from lower-resolution images with great accuracy than standard SR techniques.

PINN models are not without drawbacks; Wang et al. (2022) stated limitations of the PINN that are of particular concern to scientists: these models could benefit from improvements in multi-scale dynamic or multi-physics problems. However, one challenge they face is the difficulty in transmitting initial and boundary conditions in the hidden layers of the network (Goswami et al., 2022). Additionally, to improve their generalisation ability, they often require re-optimisation to adapt to new problem settings.

Wang et al. (2022, p. 2) stated that the main reason for the problems that PINNs face is the fully connected network structure at its foundation that cannot learn functions with high frequencies. In addition, it also suffers from unpredictable behaviour of its loss function regarding its convergence rate. The issues raised by Wang et al. (2002) above surfaced in the work of Degen et al. (2022), where the authors studied the quantification of uncertainty in a 3D multi-physics problem in Geophysics. The authors attempted to develop a surrogate model to predict the physical state of the Earth's subsurface by solving complex coupled differential equations with related uncertainty estimates (Degen et al., 2022, p. 1). However, when exposed to hyperparameter sensitivity analysis, PINNs yielded high physics loss and poor results in uncertainty quantification analysis when trained with noisy data (Degen et al., 2022, p. 5).

Furthermore, the authors raised two additional concerns with PINNs: 1) loss of authenticity as it relates to the original meaning of the problem and 2) inefficiency as it relates to its internal algorithm (Degen et al., 2022, p. 5). The inefficiency of PINNs, together with its intolerance to hyperparameter selection, changes to initial and boundary conditions (Li et al., 2021), and its dependence on knowledge of the underlying PDE renders PINNs unsuitable for real-time predictions (Goswami et al., 2022 p. 2). Moreover, it puts a high cost, like that of traditional computational methods, to dealing with complex problems (Li et al., 2021).

In the search for a solution to the limitations of PINN models, the idea of a neural operator arose. The year 2019 saw the first appearance of neural networks for solving PDEs in the form of DeepONets (Goswami et al., 2022, p.1). According to Li et al. (2021), the neural operator is a method that solves the mesh-dependency issue of traditional finite-dimensional operator methods by producing a single set of network parameters that are invariant to different discretisation. In other words, neural operators can transfer solutions between meshes without retraining. Since then, neural operator networks have found tremendous

success in solving problems such as uncertainty quantifications, autonomous systems, real-time applications, and complex multi-scale problems that have eluded PINNs and numerical simulations (Goswami et al., 2022, p.1). Unlike PINNs trained to optimise PDE constraints subject to circumstances, neural operators learn the mapping between the input space of the PDE and the corresponding function space (the space of all integral solutions for the PDE) (Li et al., 2022). Moreover, it satisfies the universal approximation theorem, so it can efficiently train faster and operate on high dimensional data (Goswami et al., 2022; Li et al., 2022). Furthermore, PINOs work well where the underlying PDE is unknown (You et al., 2022; Li et al., 2021).

Despite their success, neural operators are still data-driven models, and in previous cases, we assumed the availability of an adequate amount of training data. With numerous training examples, we can rely on numerical simulations for training data that carry high costs (Li et al., 2022). Furthermore, due to the cost of evaluating integral operators, neural operators have yet to produce efficient numerical algorithms that can match the success of convolutional or recurrent neural networks in the finite-dimensional setting (Li et al., 2021). However, by including the PDE constraints and physics-based loss, the neural operator can train on little or no data and gain significant improvements. Li et al. (2021) also showed that we might remove much of the computational costs by employing the Fast Fourier Transform method in designing the neural operator network. The Physics Informed Neural Operator (PINO) was introduced by (Wang et al., 2021) and addressed the limitations of the purely data-driven model. Wang et al. (2022) and You et al. (2022) state that PINOs have an advantage over PINNs in that they are resolution-invariant and generalise and optimise better on multi-scale (multi-physics) problems. With resolution invariance, prediction accuracy is unaffected by the input's resolution (You et al., 2022, p. 3). Hence, the model maintains its performance regardless of whether high, low, or mixed-resolution input data are available. Once trained,

the model can make predictions without regard for the resolution quality of the input and does not require additional computational resources (You et al., 2022, p. 4).

This article will expand on the present research employing PINOs to solve partial differential equations. First, we will design and implement a PINO model like the one proposed by Li et al. (2021). Then, we use the Finite Difference Method (FDM) to create the dataset to train and test our PINO model and adopt performance-boosting choices predicated on insights originating in current literature, such as selecting hyperparameters, activation function and loss function design, to optimise the model. Most notable of the choices made is introducing the physics loss coefficient for controlling the contribution of physical conservation laws to the learning mechanism. Finally, we apply the model to the two-dimensional heat conduction equation and assess the performance of the model and the impact of introducing the physics loss coefficient and other hyperparameter choices.

2. Literature Review

2.1. The Finite Difference Method for solving the Two-Dimensional Heat Equation

The finite difference approximation scheme is a numerical method used to approximate the derivatives of a function. By extension, it can approximate the solution to differential equations. The basic idea is to approximate the derivative of a function at a specific point by embedding the point on a grid and using the values of the function at nearby grid points to compute an approximate value of the derivative at the point (Burden and J. Douglas Faires, 2010, p. 684). This technique allows us to calculate derivatives without explicitly knowing the function's analytical form. For example, the finite difference approximation of the first derivative of a function *f(x)* is:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}, \qquad (1)$$

This version is the *forward difference* method. Similarly, the *backward difference* and *central difference* methods (Burden and J. Douglas Faires, 2010, p. 685) are:

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h}, \qquad (2)$$

and

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}, \qquad (3)$$

respectively with local truncation error $O(h)$. This means that the error reduces significantly as h gets closer to zero.

The central difference algorithm is commonly used in practice because it produces more accurate results than the forward and backward algorithms. The reasoning behind this is that the central difference algorithm considers the values of the function at points both forward and backwards in time. As shown below, we may consider the central difference algorithm as a weighted average of the forward (2) and backward (3) processes.

$$\overline{f'(x)} = \frac{\frac{f(x_0+h)-f(x_0)}{h} - \frac{f(x_0)-f(x_0-h)}{h}}{2} = \frac{f(x_0+h)+f(x_0-h)-2f(x_0)}{2h}$$

In practice taking weighted averages is a standard procedure for reducing errors. As such, it is clear that the central difference technique will produce minor average errors and hence more accurate approximations. This additional information allows the central difference algorithm to better approximate the function's derivative, leading to improved accuracy. Additionally, we can improve the accuracy by taking smaller step sizes; however, we must be cautious as reducing the step size comes with increased computational costs.

Figure 2.1 below shows a typical grid used for explicitly approximating the derivative of a function using the finite difference numerical scheme:

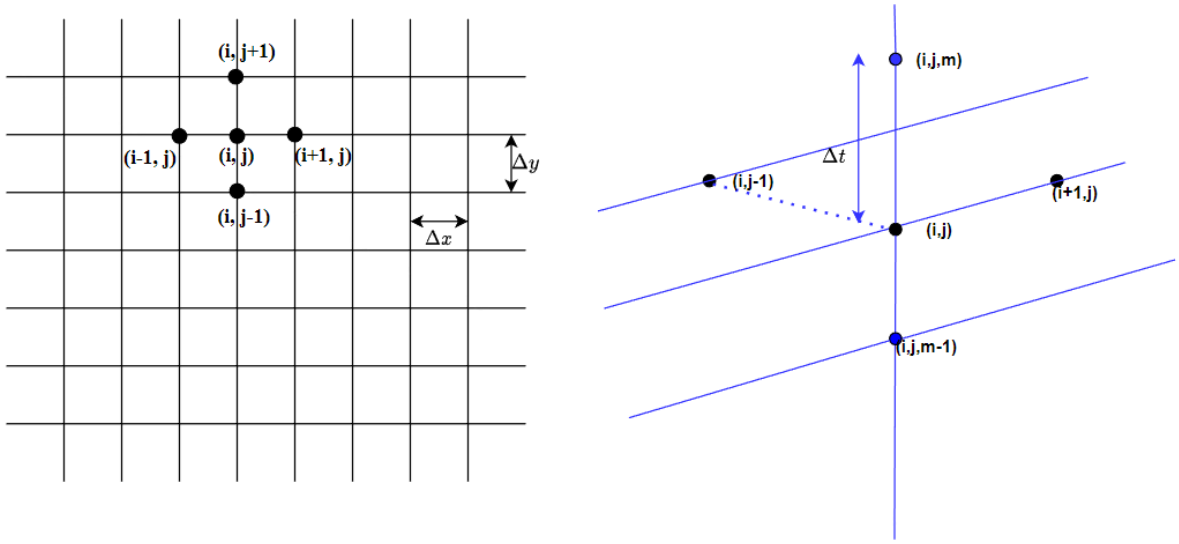

Figure 2.1.1 A typical discretisation grid used to approximate the derivatives of a function $f(x)$ motivated by the work of Aslak, Tveito and Winther (2008, p. 120)

Following from Burden and J. Douglas Faires (2010, p. 119) and Luciano Maria Barone et al. (2013, p. 272-273), and imposing the grid in Figure 2.1.1 above, we have the following approximations of the first and second derivatives of a function $\varphi(x, y, t)$:

$$\frac{\partial \varphi}{\partial t}(x,y,t) \approx \frac{{\omega_{i,j}}^{m+1} - {\omega_{i,j}}^{m}}{k}$$

$$\frac{\partial^2 \varphi}{\partial x^2}(x,y,t) \approx \frac{{\omega_{i+1,j}}^{m} - 2{\omega_{i,j}}^{m} + {\omega_{i-1,j}}^{m}}{h^2}$$

$$\frac{\partial^2 \varphi}{\partial y^2}(x,y,t) \approx \frac{{\omega_{i,j+1}}^{m} - 2{\omega_{i,j}}^{m} + {\omega_{i,j-1}}^{m}}{h^2}, \qquad (4)$$

With approximation error $O(h^2)\ and\ O(k^2)$.

We will now apply the finite difference method to develop an approximation scheme for the two-dimensional heat equation with the Dirichlet boundary conditions describe below:

$$\frac{\partial U}{\partial t}(x,y,t) - \alpha \nabla^2 U(x,y,t) = 0, \qquad t \in (0,\infty)$$

$$U(x,0,t) = U(x,M,t) = 0, \qquad x \in (0,N), t \in (0,\infty)$$

$$U(0,y,t) = U(N,y,t) = 0, \qquad y \in (0,M), t \in (0,\infty)$$

$$U(x,y,0) = f(x,y), \qquad x \in (0,N), y \in (0,M)$$

Figure 2.1.2 below, shows the scenario described by this boundary value problem.

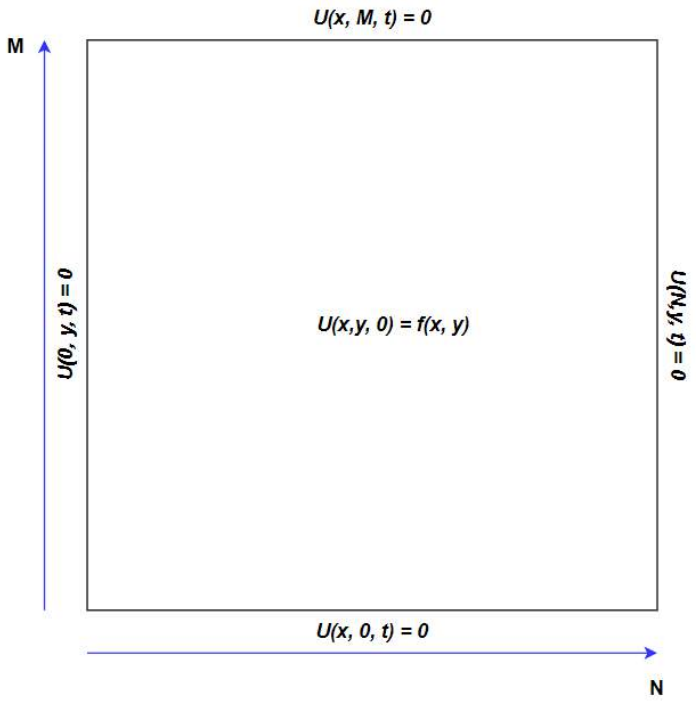

Figure 2.1.2 shows the initial setup of rectangular metal plate with initial (max temperature) heat source at the centre.

The symbol ∇^2, the *Laplacian* operator is defined as:

$$\nabla^2 = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)$$

More explicitly we have the following boundary value problem:

$$\frac{\partial U}{\partial t} - \alpha\left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}\right) = 0, \quad t \in (0, \infty)$$

$$U(x, 0, t) = U(x, M, t) = 0, \qquad x \in (0, N), t \in (0, \infty)$$

$$U(0, y, t) = U(N, y, t) = 0, \qquad y \in (0, M), t \in (0, \infty)$$

$$U(x, y, 0) = f(x, y), \qquad x \in (0, N), y \in (0, M)$$

Using the approximation formulas in (4), we can now rewrite the heat conduction equation as:

$$\frac{{\omega_{i,j}}^{m+1} - {\omega_{i,j}}^{m}}{k} \approx \alpha\left(\frac{{\omega_{i+1,j}}^{m} - 2{\omega_{i,j}}^{m} + {\omega_{i-1,j}}^{m}}{h^2} + \frac{{\omega_{i,j+1}}^{m} - 2{\omega_{i,j}}^{m} + {\omega_{i,j-1}}^{m}}{h^2}\right)$$

and with a few algebraic steps, the approximation can be reduced to (Powers, 2009, p. 420):

$${\omega_{i,j}}^{m+1} \approx (1 - 4\gamma){\omega_{i,j}}^{m} + \gamma\left({\omega_{i+1,j}}^{m} + {\omega_{i-1,j}}^{m} + {\omega_{i,j+1}}^{m} + {\omega_{i,j-1}}^{m}\right), \qquad (5)$$

with

$$\gamma = \frac{k\alpha}{h^2}$$

To ensure the stability of the approximator in (5), we need to ensure that:

$(1 - 4\gamma) \geq 0$ which leads to $\gamma \leq \frac{1}{4}$ or $\frac{k\alpha}{h^2} \leq \frac{1}{4}$ (Powers, 2009, p. 421). Therefore, we arrive at the inequality $k \leq \frac{h^2}{4\alpha}$ that defines the criterion for a stable solution.

The local truncation error associated with the approximator is $O(k + h^2)$ and we may define it as:

$$\varepsilon_{i,j,m} = \epsilon_m - \alpha(\epsilon_i + \epsilon_j)$$

where $\epsilon_m = \frac{k}{2}\frac{\partial^2 U}{\partial t^2}(x_i, y_j, \tilde{t}_m)$, $\epsilon_i = \frac{h^2}{12}\frac{\partial^4}{\partial x^4}(\tilde{x}_i, y_j, t_m)$ and $\epsilon_j = \frac{h^2}{12}\frac{\partial^4 U}{\partial y^4}(x_i, \tilde{y}_j, t_m)$ with $\tilde{t}_m \in (t_m, t_{m+1})$, $\tilde{x}_i \in (x_{i+1}, x_{i-1})$ and $\tilde{y}_i \in (y_{i+1}, y_{i-1})$.

For the steady-state (time-independent) solution, we have the Laplacian part being identically zero:

$$\alpha \nabla^2 U = \alpha\left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}\right) = 0, \alpha > 0$$

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0, \qquad (6)$$

From (6) we obtain the approximation formula:

$$\omega_{i+1,j} + \omega_{i-1,j} + \omega_{i,j+1} + \omega_{i,j-1} - 4\omega_{i,j} = 0$$

$$\omega_{i,j} = \frac{1}{4}\left(\omega_{i+1,j} + \omega_{i-1,j} + \omega_{i,j+1} + \omega_{i,j-1}\right)$$

In summary, we have developed the theory behind the numerical approximation scheme we will use for our experiment. The finite difference approximation uses the finite difference method to compute the derivatives of an unknown function over a discretisation grid. As such, it helps approximate the solution of differential equations. We have shown that a numerical scheme can be developed using the finite difference method to solve the two-dimensional heat conduction equation, along with criteria for obtaining accurate solutions (small values) and stable solutions (stability criterion for the temporal step).

2.2. Application of the Fourier Transform and the Convolution Theorem to the Two-Dimensional Heat Equation

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the Discrete Fourier Transform (DFT), which transforms a signal in the time domain into a signal in the frequency domain (Bracewell, 2000). In simpler terms, the DFT takes a signal and breaks it down into its constituent frequencies, which can help analyse the signal, removing noise or

compressing data.

Consider a function $f: \mathbb{R} \to \mathbb{R}$. We define a Fourier transform $\hat{f}: \mathcal{H} \longrightarrow \mathcal{H}$ as:

$$\hat{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt, \qquad (7)$$

According to Aslak, Tveito and Winther (2008, p. 368-370), we can regard the Fourier transform as an operator:

$$\mathcal{F}: f \mapsto \mathcal{F}\{f\} = \hat{f}, \qquad (8)$$

which satisfies the linearity criteria:

1. $\mathcal{F}\{\alpha f + \beta g\} = \alpha \mathcal{F}\{f\} + \beta \mathcal{F}\{g\} = \alpha \hat{f}(\omega) + \beta \hat{g}(\omega)$
2. $\mathcal{F}\{f'\}(\omega) = i\omega \mathcal{F}\{f\}(\omega) = i\omega \hat{f}(\omega)$
3. $\mathcal{F}\{f''\}(\omega) = i\omega \mathcal{F}\{f'\}(\omega) = (i\omega)^2 \mathcal{F}\{f\}(\omega) = -\omega^2 \mathcal{F}\{f\}(\omega) = -\omega^2 \hat{f}(\omega)$ (9)
4. $\mathcal{F}\{xf\}(\omega) = i \frac{d}{d\omega} \mathcal{F}\{f\}(\omega) = i\hat{f}'(\omega)$
5. $\mathcal{F}\{f(ax)\}(\omega) = \frac{1}{a} \mathcal{F}\{f\}\left(\frac{\omega}{a}\right)$

where f, g are functions and $\alpha, \beta \in \mathbb{R}$.

We aim to use the Fourier transform method to solve partial differential equations, we will eventually need a want to reverse the transformation $\mathcal{F}\{f\}(\omega)$ to obtain the original function f. To do this, we follow from the results of Aslak, Tveito and Winther (2008, p. 374) to define the inverse Fourier transform $\mathcal{F}^{-1}$ as:

$$f(t) = \mathcal{F}^{-1}\{\hat{f}(\omega)\} = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega)\, e^{i\omega t} d\omega, \qquad (10)$$

The final part of this section details the use of the Fourier transform method to solve the two-dimensional heat equation.

Let us consider the two-dimensional heat equation in the form:

$$\frac{\partial U}{\partial t} = \alpha^2 \left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \right), \quad t \in (0, \infty), \qquad (11)$$

with initial condition:

$$U(x,y,0) = f(x,y), \qquad x \in (0,N), y \in (0,M)$$

Taking the Fourier transform of (10) we have:

$$\frac{\partial \widehat{U}}{\partial t}(\omega_x, \omega_y) = -\alpha^2\left({\omega_x}^2 + {\omega_y}^2\right)\widehat{U}, \quad t \in (0,\infty), \qquad (12)$$

Equation (12) above is a simple ordinary differential equation in $\widehat{U}$ and easily solved giving:

$$\widehat{U}\left(\omega_x, \omega_y\right) = \widehat{U}_0 e^{-\alpha^2({\omega_x}^2+{\omega_y}^2)t}, \qquad (13)$$

where $\widehat{U}_0 = \mathcal{F}\{U(x,y,0)\} = \mathcal{F}\{f(x,y)\} = \hat{f}(x,y)$.

We now have:

$$\widehat{U}\left(\omega_x, \omega_y\right) = \hat{f}(x,y) e^{-\alpha^2\left({\omega_x}^2+{\omega_y}^2\right)t}, \qquad (14)$$

The solution to the transform equation (13) is therefore given by: $U(x,y,t) =$

$\mathcal{F}^{-1}\{\widehat{U}\left(\omega_x, \omega_y\right)\} = \frac{1}{4\pi^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{i(\omega_x x+\omega_y y)} e^{-\alpha^2({\omega_x}^2+{\omega_y}^2)t}\hat{f}(x,y)d\omega_x d\omega_y,$ (15)

The solution to the two-dimensional heat equation given in (14) is quite complex and computationally expensive. In the following section, we will discuss a much simpler approach using Fourier transforms on convolutions. We start by defining a convolution and stating the Convolution Theorem as introduced by Kreyszig (2019, p. 527).

Let f and g be bounded, piecewise continuous and integrable functions on $\mathbb{R}$. We can define the convolution of f and g as:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(p)g(x-p)dp = \int_{-\infty}^{\infty} f(x-p)g(p)dp\,, \qquad (16)$$

and its corresponding Fourier transform as:

$$\mathcal{F}\{(f * g)\} = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(p)g(x-p)dp \cdot e^{-i\omega x}dx\,, \qquad (17)$$

By setting $x = p + q$ we obtain:

$$\mathcal{F}\{(f * g)\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(p) \cdot g(q) dp \cdot e^{-i\omega(p+q)} dq$$

which we can split into two separate integrals and simplify to get:

$$\mathcal{F}\{(f * g)\} = \sqrt{2\pi} \cdot \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

Finally, by taking the inverse transform, we obtain:

$$(f * g)(x) = \int_{-\infty}^{\infty} \hat{f}(\omega) \cdot \hat{g}(\omega) \cdot e^{i\omega x} d\omega\,, \qquad (18)$$

We can extend the results in (15) - (18) to solve the 2D heat equation. We will discuss this in detail in this final part of this section.

From (14) we have

$$\widehat{U}(\omega_x, \omega_y) = \hat{f}(x, y) e^{-\alpha^2(\omega_x{}^2 + \omega_y{}^2)t}$$

We would like to find a function $S(x, y, t)$ such that:

$$\mathcal{F}\{S(x, y, t)\} = \hat{S}(\omega_x, \omega_y, t) = e^{-\alpha^2(\omega_x{}^2 + \omega_y{}^2)t}$$

By using the inverse Fourier Transform on $\hat{S}(\omega_x, \omega_y)$ we get

$$S(x, y, t) = \frac{1}{4\alpha^2\sqrt{\pi t}} e^{-\frac{x^2+y^2}{4\alpha^2 t}}$$

We can now rewrite (14) as a product of Fourier transforms:

$$\widehat{U}(\omega_x, \omega_y) = \hat{f}(x, y) \cdot \hat{S}(x, y, t)$$

and by the Convolution Theorem, we know that we can write the inverse Fourier transform of the product of Fourier transforms as a convolution of two functions:

$$U(x, y, t) = \mathcal{F}^{-1}\{\hat{f}(x, y) \cdot \hat{S}(x, y, t)\} = f(x, y) * S(x, y, t)$$

We can now express the solution of the 2D heat equation as:

$$U(x,y,t) = f(x,y) * S(x,y,t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} S(x-p, y-q, t) f(p,q)\, dpdq$$

$$= \frac{1}{4\alpha^2\sqrt{\pi t}} \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-\left\{\frac{(x-p)^2+(y-q)^2}{4\alpha^2 t}\right\}} dpdq, \qquad (19)$$

In the form (19), we can approximate the solution using the FTT algorithm.

2.3. Neural Network methods for solving partial differential equations.

Partial differential equations (PDEs) are mathematical models that describe the behaviour of physical systems. They contain one or more partial derivatives of an unknown function that depends on at least two variables. Typically, one of these variables deals with time t (temporal variable), and the remaining variables deal with space (spatial variable(s)) (Henner, Belozerova and Nepomnyashchy, 2019). Essential PDEs are the wave equations, which can model the vibrating string and the vibrating membrane, the heat equation for temperature in a bar or wire, and the Laplace equation for electrostatic potentials. These equations are fundamental to our understanding of the physical world. Consequently, PDEs arise in various applications, from dynamics and elasticity to heat transfer, electromagnetic theory, and quantum mechanics (Kreyszig, 2019).

Solving PDEs can pose a challenge, often requiring numerical methods. Traditional numerical methods, including finite difference, finite element, and spectral methods, have been used for many years to solve PDEs (Oden, 2013 Katsikis, Muradova and Stavroulakis, 2022). However, these methods can be computationally intensive and may not be suitable for problems with complex geometries or high-dimensional data (Oldenburg et al., 2022; Oden, 2013). Therefore, there has been growing interest in using neural network methods to solve PDEs in recent years. Neural networks are a type of machine learning algorithm that can approximate complex functions and patterns in data. The ability of neural networks to

approximate functions makes them a promising tool for solving PDEs, particularly for problems with high-dimensional data and complex geometries (Han and Jentzen, 2017 Katsikis, Muradova and Stavroulakis, 2022). Neural networks comprise multiple interconnected layers of artificial neurons, each performing a simple operation on its input and producing an output (Sharma and Pradeep Kumar Garg, 2021). The neurons' outputs in one layer function as inputs to the neurons in the next layer until it produces a final output. These intermediary layers are often known as the "hidden layers" of the neural network.

The ability of neural networks to approximate functions makes them a promising tool for solving PDEs. One common approach is to use neural networks to learn the solution to a PDE from a set of training data (Raissi, Perdikaris and Karniadakis, 2019; Sirignano & Spiliopoulos, 2018). This approach involves selecting a suitable neural network architecture, an appropriate loss function, and a set of optimal hyperparameters to achieve superior performance. Another approach employs a hybrid architecture that combines traditional numerical methods with one or more neural networks to learn a correction term to the solution produced by a numerical method. This hybrid approach produces more accurate results than the numerical method while remaining computationally efficient (Raissi, Perdikaris and Karniadakis, 2019). Generating high-quality training data is critical for training neural networks to solve PDEs (Sirignano & Spiliopoulos, 2018). The training data must accurately represent the behaviour of the PDE in the problem domain. Some techniques commonly used for generating training data include the finite difference method, which discretises the PDE over a grid and solves it numerically at each grid point (Burden and J. Douglas Faires, 2010, p. 684), the Monte Carlo method, which randomly samples the PDE over the domain (Luciano Maria Barone et al., 2013). Adversarial networks train a neural network to generate data indistinguishable from the actual data distribution (Goodfellow et al., 2020). After generating the training data, it is necessary to pre-process it to ensure its high

quality (Wiens & Shenoy, 2017). This process may involve removing outliers, smoothing the data, and normalising the input and output variables.

Additionally, techniques such as data augmentation generate additional training data by applying transformations to the existing data. However, the accuracy of the neural network solution crucially depends on the quantity and quality of the training data. Inaccurate and unstable solutions can result from insufficient or low-quality data (LeCun, Bengio and Hinton, 2015). Hence, designing the training data generation process is essential to ensure adequate training of neural networks on high-quality data.

Neural network methods for solving PDEs have shown promising results for problems with high-dimensional data and complex geometries. However, they also have some challenges, such as the need for large amounts of training data, the difficulty in choosing the appropriate architecture and hyperparameters, and the lack of theoretical guarantees on their accuracy and stability (Han, Jentzen and E, 2018 and Yang, Zhang and Karniadakis, 2020). Nonetheless, the potential benefits of neural network methods make them an active area of research in computational mathematics.

To summarise, partial differential equations (PDEs) serve as mathematical models to explain the workings of physical systems and play a crucial role in various fields, including dynamics, elasticity, heat transfer, electromagnetic theory, and quantum mechanics. While using traditional numerical methods to solve PDEs is customary, they may prove computationally demanding and impractical for intricate geometries or high-dimensional data. Conversely, neural networks have demonstrated exceptional precision and efficiency in solving PDEs that involve complex geometries and high-dimensional data. Although neural network methods present specific challenges, their advantages make them a thriving field of research in computational mathematics.

2.4. Using Physics-Informed Neural Networks to solve Partial Differential Equations

A promising approach for solving PDEs involves using physics-informed neural networks (PINNs). These networks combine the accuracy and efficiency of neural networks with the physical principles governing the modelled system. PINNs are a hybrid approach between traditional numerical methods and machine learning (Sirignano & Spiliopoulos, 2018). The trained neural network satisfies both the PDE and the boundary conditions of the problem (Raissi, Perdikaris and Karniadakis, 2019; Sirignano & Spiliopoulos, 2018). The fundamental concept behind PINNs is to include the physics of the problem in the neural network architecture by enforcing the PDE and boundary conditions as constraints on the network (Vadyala, Betgeri and Betgeri, 2022) by adding terms to the loss function that penalises any deviation from the PDE and boundary conditions. By training the network to minimise this loss function, the PINN can learn the solution to the PDE and accurately predict the physical system's behaviour (Raissi, Perdikaris and Karniadakis, 2019).

PINNs have several advantages over traditional numerical methods for solving PDEs. First, they are highly adaptable and can solve PDEs in complex geometries or with high-dimensional data (Han, Jentzen and E, 2018). Second, they are computationally efficient, allowing faster computation than traditional numerical methods (Vadyala, Betgeri and Betgeri, 2022). Third, PINNs are generalisable and can solve a wide range of PDEs without requiring extensive modifications to the neural network architecture. To implement a PINN, we first select a neural network architecture. Usually, a fully connected neural network with numerous hidden layers is preferred, though alternative architectures, such as convolutional neural networks, may also be considered (Zhao et al., 2021). The network's input is the spatial and temporal coordinates of the system under consideration, and the output is the solution to the PDE (Raissi, Perdikaris and Karniadakis, 2019). Next, we add physical

constraints to the loss function of the network. These constraints can take various forms depending on the specific PDE (Sirignano & Spiliopoulos, 2018). For instance, we can use the continuity equation for fluid flow to constrain the divergence of the velocity field. We then minimise the resulting loss function using gradient-based optimisation methods such as stochastic gradient descent (Zhu et al., 2019).

Training physics-informed neural networks (PINNs) can prove computationally intensive, mainly when dealing with extensive datasets. Moreover, the accuracy and convergence of the solution depend significantly on the neural network architecture and hyperparameters chosen (Raissi & Karniadakis, 2018; Li et al., 2021). Additionally, the PINN method demands a profound comprehension of fundamental physics, and accurately formulating the physics-informed loss function may take more work (Goswami et al., 2022). Finally, Wang, Yu and Perdikaris (2022) attributed the instability and inaccuracy of fully connected PINNs to multi-scale interactions in the PINNs loss function for problems with high-frequency or multi-scale features, leading to stiffness in the gradient flow dynamics. Despite these drawbacks, they perform superbly when the conditions are suitable for applying a PINN model. Ngom and Marin (2021) developed a new neural network model called PIFNN that uses a fully interpretable neural network (FNN) with a residual-based loss function to solve differential equations with periodic boundary conditions. The FNN uses cosine as the activation function, mimics a Fourier decomposition, and preserves the periodic nature of the solution. The PIFNN model outperformed traditional neural networks in accuracy and required fewer iterations while conserving the learned task's properties outside the training domain.

2.5. Operator Learning applied to solving Partial Differential Equations

Parametrising PDEs allows us to use neural network methods to solve them (Li et al., 2021). However, using traditional tools to solve PDEs involves repeatedly using costly analytical or

computational tools as independent simulations for every domain geometry, input parameter, or initial and boundary conditions. More recently, researchers began applying machine learning tools to solve PDEs. However, most existing tools can only accommodate a fixed set of input parameters or initial and boundary conditions. Mathematically, if we consider Banach spaces $(\mathcal{U}, \mathcal{V}, \mathcal{S})$ and $\mathcal{N}: \mathcal{U} \times \mathcal{S} \longrightarrow \mathcal{V}$ be a differential operator, then we can write a parametric differential equation in the form:

$$\mathcal{N}(u, s) = 0$$

where $u \in \mathcal{U}$ is a set of parameters (input function) and $s \in \mathcal{S}$ is the unknown solutions of the PDE (Wang, Wang and Perdikaris, 2021, p. 3). Solving these parametric PDE problems involves learning the solution operator $\widetilde{G}: u \in \mathcal{U} \times \theta \in \Theta \longrightarrow s(u) \in \mathcal{S}$ that maps input space $\mathcal{U}$ to the corresponding latent solution space $\mathcal{S}$ of the PDE system, parametrised by a set of weights $\theta \in \Theta$ (Rosofsky, Al Majed and Huerta, 2022). The solution operator $\tilde{G}$ is a mapping between infinite-dimensional function spaces, instead of functions that map finite-dimensional vector spaces; it is an integral Hilbert-Schmidt operator, whose kernel is learned from paired observations (Wang, Wang and Perdikaris, 2021). Neural operators are resolution-independent (Li et al., 2022), but they require significant training data sets, and their implementation is often slow and computationally expensive (Wang, Wang and Perdikaris, 2021). Recently, DeepOnet operator learning architectures have arisen, inspired by the universal approximation theorem for operators, and while architectures still require large, annotated data sets of input-output observations, they have a simple and intuitive model architecture that is fast to train and allow for a continuous representation of the target output functions that is independent of resolution (Wang, Wang and Perdikaris, 2021). Operator learning finds application in solving complex physical problems (Li et al., 2022) where the natural governing law is unknown (You et al., 2022) or has no analytical representation. Additionally, operator learning has broad applicability across diverse

applications, including fluid mechanics, heat transfer, bioengineering, materials, and finance (Raissi, Yazdani and Karniadakis, 2020). For example, in materials science, operator learning can be used to model the mechanical response of materials, considering factors such as material heterogeneity and defects (Raissi, Perdikaris and Karniadakis, 2019; Sirignano & Spiliopoulos, 2018). You et al. (2022) proposed a data-driven approach to predict material responses without using conventional constitutive models. The approach utilises high-fidelity simulation and experimental measurements to learn implicit mappings between loading conditions and the resultant displacement and damage fields. The authors introduce a novel deep neural operator architecture, called the Implicit Fourier Neural Operator (IFNO), to model complex responses due to material heterogeneity and defects. The IFNO captures long-range dependencies in the feature space by modelling the increment between layers as an integral operator, which is parameterised directly in the Fourier space to allow for fast Fourier transformation (FFT) and accelerated learning techniques. The authors demonstrate the effectiveness of the proposed approach for hyperelastic, anisotropic, and brittle materials, as well as for learning material models directly from digital image correlation (DIC) tracking measurements.

Operator learning has the potential to be applied to optimisation problems, allowing for the efficient search of optimal input parameters that satisfy certain constraints (Raissi, Perdikaris and Karniadakis, 2019). In this experiment, we will restrict the integral operator as a convolution operator applied as a linear transformation in the Fourier domain (Li et al., 2021, p. 3).

2.6. Generalising the solution of PDEs using Physics-Informed Neural Operators

Physics-Informed Neural Operators (PINOs) are a recently introduced concept in deep learning for solving PDEs. A PINO is a neural network trained to learn a differential operator rather

than directly learning the solution of a PDE. The differential operator can then be used to compute the solution of the PDE at any point in the domain (Li et al., 2022). The idea behind PINOs is that by learning the differential operator, the neural network can capture the underlying physics of the problem and generalise to unseen data (Rosofsky, Al Majed and Huerta, 2022). These properties of PINOs are beneficial in cases where the exact solution is not available or is difficult to compute. Alternately, PINOs are the integration of operator learning and physics-informed settings. PINO reduces the labelled dataset requirement for training the neural operator. As a result, it facilitates faster solution convergence (Goswami et al., 2022). Furthermore, by combining the physics and data constraints, PINOs obtain better optimisations leading to more authentic learning (Li et al., 2022 Rosofsky, Al Majed and Huerta, 2022). Raissi et al. (2019) showed the application of PINOs to various fields, such as fluid dynamics, heat transfer, and image segmentation, obtaining superior results.

PINOs are designed based on the Fourier Neural Network (FNO) architecture which applies the Fast Fourier Transform (FFT) to data, pushes it through a fully connected neural network in the Fourier space, and then applies the inverse FFT to transform to bring the data back to the original space. It combines this with information from the PDE, initial and boundary conditions, and other conservation laws allowing the model to learn faster from small amounts of training data (Rosofsky, Al Majed and Huerta, 2022, p. 2). Rosofsky, Al Majed, and Huerta (2022) found that the PINO model achieved remarkably high accuracy and enabled the model to perform zero-shot resolution, that is, the ability to predict high-resolution data while training on low-resolution data.

Despite the promising results that PINOs have shown in solving PDEs, several challenges and limitations still need to be addressed. Firstly, theoretical challenges associated with physics-informed neural operators (PINOs) are a need for a mathematical theory, overfitting due to insufficient data, and generalisation issues beyond the training data. Although PINOs can

generalise well for small changes in the input, they may need to be more reliable for extrapolations or more significant changes (Raissi, Yazdani and Karniadakis, 2020). Further development of mathematical theory is necessary to understand the behaviour and properties of PINOs. Secondly, PINOs may require a high computational cost, careful hyperparameter tuning, and specialised hardware to train on large-scale problems (Raissi, Perdikaris and Karniadakis, 2019). Finally, PINOs may need to be more suitable for certain PDEs or problems with complex geometries or boundary conditions. Obtaining high-quality data is crucial for training PINOs, and it may be limited or expensive to acquire in some cases. Additionally, the lack of interpretability of PINOs makes it challenging to comprehend how they generate solutions, thereby restricting their application in areas where interpretability is crucial. Addressing these challenges and limitations will be crucial for the widespread adoption of PINOs in solving PDEs and advancing the field of computational physics.

3. Methodology

3.1. The Computing Environment

We performed the numerical experiments on a 64-bit Ubuntu operating system version 22.10 with a 2.9 GHz 8-core Ryzen 7 4800H CPU and a single Nvidia RTX 3050 GPU. We used the Python programming language in the Spyder 5 and VS Code environments to implement the numerical algorithms for generating the dataset used in the experiments and visualise the results of the training, testing and prediction processes. Python has become increasingly popular in recent years due to its suitability for small "prototyping" tasks without explicit variable declaration or separate compilation, its availability on most computer systems, and a vast collection of packages covering a broad range of applications (Vadyala, Betgeri and Betgeri, 2022, p. 3). According to the Spyder Team (2018), Spyder is a free, open-source scientific computing software that delivers a unique blend of scientific package features, such as data

exploration, interactive execution, beautiful visualisation capabilities, and development tool functions. The choice of Spyder is also due in part to familiarity with previous development projects. Additionally, we used the VS Code editor to help with GitHub integration and code linting. We used the PyTorch deep learning framework to implement, train and evaluate the PINO model. According to PyTorch (2019), PyTorch is an open-source machine learning framework that simplifies the path from research prototyping to production deployment, offering scalable distributed training and performance optimisation (via dedicated GPU support (Sharda, Dursun, Delen and Efraim Turban, 2021, p. 404)) for research and production, various tools and libraries, and significant cloud platform support.

3.2. The General Model Architecture

The general architecture of our model consists of five main components:

Input data: The model inputs a rectangular region's heat conduction heat map. Before passing the input data to the Fourier Transform layer, we must format the data in an appropriate. We describe the data processing algorithm here. We stored the algorithms used to pre-process the input data in the preprocess.py file.

The *preprocess.py* file contains the data pre-processing and training functions necessary for the PINO model. These functions format the input data, define the loss function, and train the model. Here is a general description of the main components of the preprocess.py file:

- **Fourier Derivative Function**: The *fourier_derivative_2d()* function computes the numerical derivative of a 2D tensor using the Fourier transform. This function calculates the spatial derivatives of the output and target tensors in the energy conservation loss.

- **Loss Function**: The *loss_function()* calculates the overall loss for the model by combining two components: the operator loss and the physics loss. The operator loss is the mean squared error (MSE) between the model's output and the target PDE solution. The physics loss is the energy conservation loss calculated using the *energy_conservation_loss()* function.
- **Energy Conservation Loss**: The *energy_conservation_loss()* function computes the error based on the energy conservation law associated with the underlying physics of the problem. This function calculates the output and target tensors' time and spatial derivative errors and computes the energy conservation error based on these errors. The physics loss ensures that the model learns from the data and respects the underlying physics of the problem.
- **Train Function:** The train function handles the training of the PINO model using the provided datasets. It takes the model, loss function, optimizer, train, and test loaders, and other hyperparameters as inputs. During training, the function iterates through the epochs and updates the model parameters based on the calculated loss. Additionally, it evaluates the model on the test dataset and prints the training and test losses for each epoch. The function returns the trained model and the loss history for both training and test datasets.
- **Plot Loss Function:** The *plot_loss()* function generates a plot of the training and test loss history over the epochs. It takes the train and test loss histories as inputs and creates a graph displaying the losses. This function can also save the generated plot to a specified file path.

Fourier transforms: The Fourier transform layer converts the spatial information of the input data into the frequency domain, usually to simplify the computational effort involved in solving PDEs. Using an encoder neural network, we apply the Fast Fourier Transform (FFT)

to the input data. The encoder layer helps the model learn more efficiently by capturing global features and reducing the complexity of the problem. We stored the code for the encoder layer in the *fourier_transform_layer.py* file.

Neural operator: The neural operator is the core of the PINO algorithm. It is a deep-learning model designed to learn and approximate the underlying PDE operator. The neural operator layer comprises a fully connected neural network operating on the input data that has been transformed into the frequency domain by the Fourier Transform layer. The neural operator layer is responsible for learning the underlying physics of the problem, allowing it to approximate the solution to the PDE by applying a series of transformations to learn the relationship between the input heatmaps and their corresponding PDE solutions. The neural operator used the data-driven loss (MSE loss) to measure how well the model fits the data and the physics-based loss (energy conservation loss) to represent the underlying physics accurately. Our experiment uses a fully connected neural network with four hidden layers and GELU activation between them. We stored the code for the neural operator in the *neural_network.py* file described below.

Inverse Fourier transform: The inverse transform layer transforms the output from the neural operator back to the spatial domain using the inverse Fourier transform. The inverse transform layer generates the model's prediction of the system's state at a given time. Using a decoder neural network, we applied the inverse Fourier transform to the neural network output, reconstructing the PDE solution in a format that can be easily interpreted and compared with the ground truth data. We stored the code for the decoder in the *inverse_fourier_transform.py* file.

Output data: The model outputs its prediction, which can be compared to the actual state of the system to evaluate its accuracy and performance.

The overall architecture is illustrated in the data flow diagram (Figure 3.2.1) below.

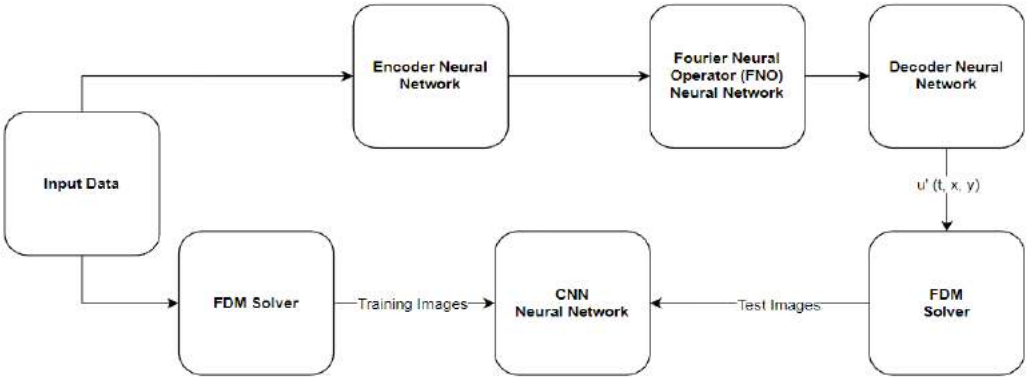

Figure 3.2.1 shows a data flow diagram to illustrate an overview of the full PINO model based on a Fourier Neural Network (FNO)

We intended to train the convolutional neural network (CNN) shown in figure 3.2.1 on the same training data as the PINO for the purpose of comparing the predictions of the PINO with the ground truth (numerical simulations). We did not implement the CNN in this paper due to time constraints and because we found a simpler and less sophisticated way of doing this. However, this approach could prove to have some advantages over our current method and may be worth investigating in future work.

3.3. Training Process

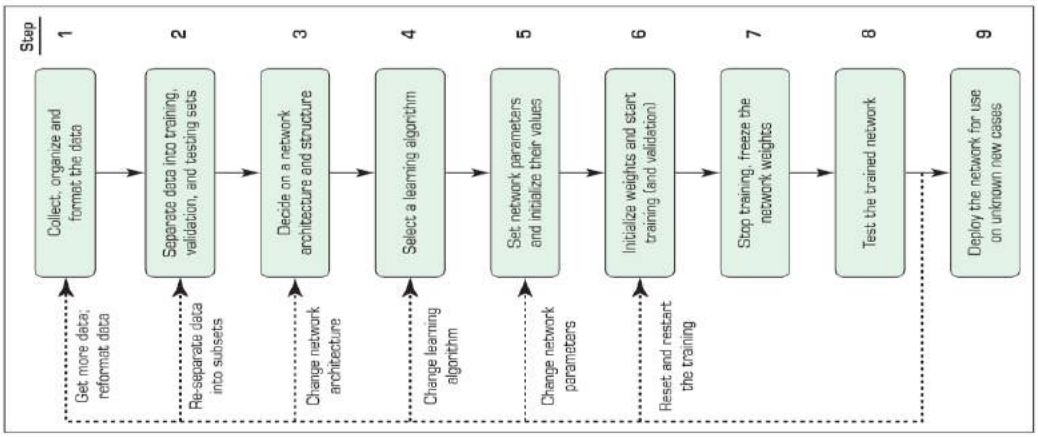

Figure 3.3.1 show the development process for a neural network-based system.(Sharda, Dursun, Delen and Efraim Turban, 2021, p. 370).

We trained our model with a dataset of heatmap images and their corresponding PDE solutions in NumPy's *'xxxxx.npz'* format for various initial conditions, different values for thermal diffusivity and physics loss coefficient. We then divided the dataset proportionately into 75% training and 25% testing (step 2 in Figure 3.3.1). We then selected various hyperparameters (as per the strategy defined in Section 3.4) to improve our model. We use the stochastic gradient descent algorithm (Sharma and Pradeep Kumar Garg, 2021, p. 69) and the Adam optimiser, with a backpropagation based on Fourier derivatives to minimise the loss function in both cases. We were motivated to make this choice of backpropagation technique by Wang, Wang and Perdikaris (2021), who took advantage of the efficiencies afforded by using the Fourier derivatives. We implemented the Fourier derivative in the definition of the physics loss function. Additionally, we defined our loss function based on Vadyala, Betgeri and Betgeri (2022) to include the losses from the operator and deviations from the governing physical laws. The equation below defines the loss function used in our experiment:

$$L = L_{op} + +\varepsilon L_{phys}$$

where ε is a special parameter called the '*physics loss coefficient*'. This parameter will play an important role in training our PINO model.

$L_{op,}$ the operator loss, is based on the mean squared error (MSE) defined by (Sharma and Pradeep Kumar Garg, 2021, p. 68):

$$L_{op} = \frac{1}{N}\sum\left(u_i - \tilde{G}(u_i, \theta)\right)^2$$

To motivate the physics loss, L_{phys}, we consider energy arguments (Aslak Tveito and Winther, 2008) based on the law of conservation of energy, and the finite difference method. The law of conservation of energy states, in simple terms, that the net heat flow into a region is equal to the change in internal energy of the region. Let us denote the actual temperature distribution as $u(x, y, t)$ and the predicted temperature distribution as $\tilde{G}_u(x, y, t)$. The objective is to minimize the difference between the predicted and actual energy in the system.

First, compute the heat flow into and out of the region for both the actual and predicted distributions using Fourier's law of heat conduction $Q = -k\nabla u$ (Gooch, 2011 and Narasimhan, 1999):

$$Q_{actual} = -k\left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right)$$

$$Q_{predicted} = -k\left(\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{u}}{\partial y}\right)$$

Next, compute the change in internal energy for the actual and predicted distributions:

$$\Delta U_{actual} = \rho c \frac{\partial u}{\partial t}$$

$$\Delta U_{predicted} = \rho c \frac{\partial \tilde{u}}{\partial t}$$

Now, formulate the energy conservation constraint for both actual and predicted distributions:

$$Q_{actual} - Q_{predicted} = k\left[\left(\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{u}}{\partial y}\right) - \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right)\right] = k\left[\left(\frac{\partial \tilde{u}}{\partial x} - \frac{\partial u}{\partial x}\right) + \left(\frac{\partial \tilde{u}}{\partial y} - \frac{\partial u}{\partial y}\right)\right]$$

$$\Delta U_{actual} - \Delta U_{predicted} = \rho c \frac{\partial u}{\partial t} - \rho c \frac{\partial \tilde{u}}{\partial t} = \rho c\left[\frac{\partial u}{\partial t} - \frac{\partial \tilde{u}}{\partial t}\right]$$

$$k\left[\left(\frac{\partial \tilde{u}}{\partial x} - \frac{\partial u}{\partial x}\right) + \left(\frac{\partial \tilde{u}}{\partial y} - \frac{\partial u}{\partial y}\right)\right] = \rho c\left[\frac{\partial u}{\partial t} - \frac{\partial \tilde{u}}{\partial t}\right]$$

$$\alpha\left[\left(\frac{\partial \tilde{u}}{\partial x} - \frac{\partial u}{\partial x}\right) + \left(\frac{\partial \tilde{u}}{\partial y} - \frac{\partial u}{\partial y}\right)\right] = \left[\frac{\partial u}{\partial t} - \frac{\partial \tilde{u}}{\partial t}\right]$$

where $\alpha = \frac{k}{\rho c}$.

We will therefore define our physics loss, L_{phys}, as:

$$L_{phys} = \int\int\int \left(\left|\frac{\partial u}{\partial t} - \frac{\partial \tilde{G}}{\partial t}\right| - \alpha\left(\left|\frac{\partial \tilde{G}}{\partial x} - \frac{\partial u}{\partial x}\right| + \left|\frac{\partial \tilde{G}}{\partial y} - \frac{\partial u}{\partial y}\right|\right)\right)^2 dxdydt$$

where $\tilde{G}$ is the output from the PINO and u is the corresponding training example from the training data. The goal is to minimize this loss function over the entire special domain and time.

L_{phys} quantifies the discrepancy between the actual and predicted energy in the system ensuring that the model adheres to the energy conservation law. The parameter ε represents the physics loss coefficient, a hyperparameter introduced to exert control over the influence of the physics error loss over the overall loss function. By modifying it, we can balance having an excellent fit for the data and satisfying the physics law requirements.

The GELU (Gaussian Error Linear Units) activation was used in all cases as introduced by Rosofsky, Al Majed and Huerta (2022). This choice of activation function is deliberate, as we require the activation function to have a non-zero second derivative and be sufficiently smooth. Hendrycks and Gimpel (2018, p. 2) first introduced the GELU function defined by:

$$GELU(x) = xP(X \leq x) = x\Phi(x) = \frac{x}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right), \qquad (20)$$

where $\Phi(x)$ is the Gaussian Cumulative distribution function. If $X \sim \mathcal{N}(0,1)$ we can be approximate (20) by:

$$GELU(x) \approx \frac{x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right]\right)}{2}, \qquad (21)$$

Hendrycks and Gimpel (2018, p. 2) were able to gain significant performance improvements when compared with more common activation functions such as ReLU (Rectified Linear Units) and the ELU (Exponential Linear Units) in various applications, including computer vision and speech recognition. In our experiments, we used the $GELU(x) = \frac{x}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ definition for implementing the activation function. In our experiment, we used the GELU function provided by PyTorch.

3.4. Hyperparameter and Optimisation Techniques

The hyperparameters used in the experiments include the learning rate, the physics loss coefficient, the number of epochs, the regularisation techniques to prevent overfitting (Sharda,

Dursun, Delen and Efraim Turban, 2021, p. 374), and the optimisation technique for operator learning. We modified hyperparameters based on the model's performance on the test dataset. Additionally, we will define and use a Fourier-based derivative method to facilitate backpropagation. This choice is motivated by Rosofsky, Al Majed and Huerta (2022, p. 2) to overcome the inefficiencies of automatic differentiation.

Table 3.4.1 below describes the hyperparameter strategy we used in our experiment.

Experiment A : Optimiser = SGD				**Experiment B : Optimiser = ADAM**			
Code	**Learning Rate**	**Physics Loss Coefficient**	**Thermal Diffusivity**	**Code**	**Learning Rate**	**Physics Loss Coefficient**	**Thermal Diffusivity**
A1	0.001	0.001	0.001	B1	0.001	0.001	0.001
A2	0.005	0.01	0.01	B2	0.005	0.01	0.01
A3	0.01	0.1	0.1	B3	0.01	0.1	0.1

Table 3.4.1 show the hyperparameter strategy for selecting the optimal model.

We selected the best performing combination of hyperparameters (based on the hyperparameter and optimiser strategy) as optimal and used for prediction and analysis. The basis for selection will be the combination with the highest accuracy (R^2 -Score) score.

3.5. Dataset Generation

According to Sharda, Dursun, Delen and Efraim Turban (2021) the process of developing a neural network-based system is quite like the structure development scheme used in traditional

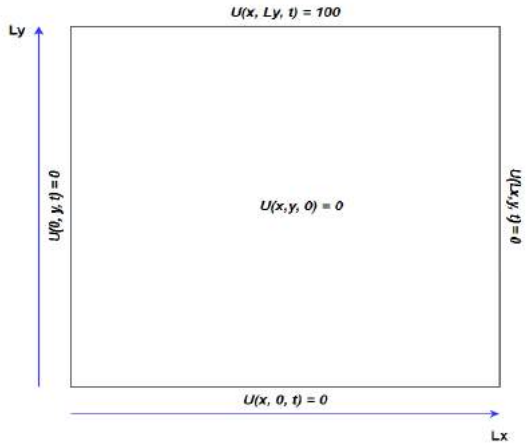

Figure 3.5.1 shows the context for our numerical experiments.

software engineering practice. The schema in Figure 3.3.1 describes the process of developing a neural network-based system graphically. Following the schema, we start by describing the way we collect our data.

The mathematical model representing the problem above is give by:

$$\frac{\partial U}{\partial t} - \alpha\left(\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2}\right) = 0, \quad t \in (0, t_{max})$$

$$U(x, 0, t) = U(x, M, t) = 0, \qquad x \in (0, L_x), t \in (0, t_{max})$$

$$U(0, y, t) = U(N, y, t) = 0, \qquad y \in (0, L_y), t \in (0, t_{max})$$

$$U(x, y, 0) = f(x, y), \qquad x \in (0, L_x), y \in (0, L_y)$$

We will now describe the discretisation process for our experiment. We will consider a rectangular region of dimension $(L_x, L_y) = (150, 250)$ and will allow the conduction process to run for a maximum time, t_{max}, of 500 seconds. We will take spatial discretisation units $(\Delta x, \Delta y) = (1, 1)$. We therefore have the discretisation table shown in Table 3.4.2 below:

Variable	**Value**
(x_0, y_0)	$(0,0)$
$\Delta t = h^2/4\alpha^2$	2.5
x_i	$i\Delta x,\ i < 150$
y_j	$j\Delta y,\ j < 250$
t_m	$m\Delta t,\ m < 200$
${\omega_{i,j}}^m$	$u\ (x_i, y_j, t_m)$

Table 3.4.2 show the discretisation set up for the data generation experiment.

We use the finite difference method to generate our training and test data, and to evolve the 2D heat equation in time according to the discretisation process above. We describe the algorithm below.

The Finite Difference Method

1. Discretise the rectangular domain into a grid with uniform spacing **Δx** and **Δy** and divide the time domain into discrete time steps of size **Δt**.
2. Use the forward difference formula to approximate for the time derivative and central differences formula for the second-order spatial derivatives as described in Section 2.1.

3. Set the initial temperature distribution $u(x, y, 0) = f(x)$.
4. Apply the Dirichlet boundary conditions as defined in Figure 5.4.
5. Iterate of a set of time steps and update the temperature u_{ij}^{m} at each grid point (x, y).

We implemented the finite difference algorithm above in Python and generated the training data that we needed to conduct our experiment. The code for this entire project can be found at: dave2k77/fourier_pino_model (github.com).

4. Results

4.1. Data Generation Results

The Finite Difference Method (FDM) algorithm described above generated the following sample images.

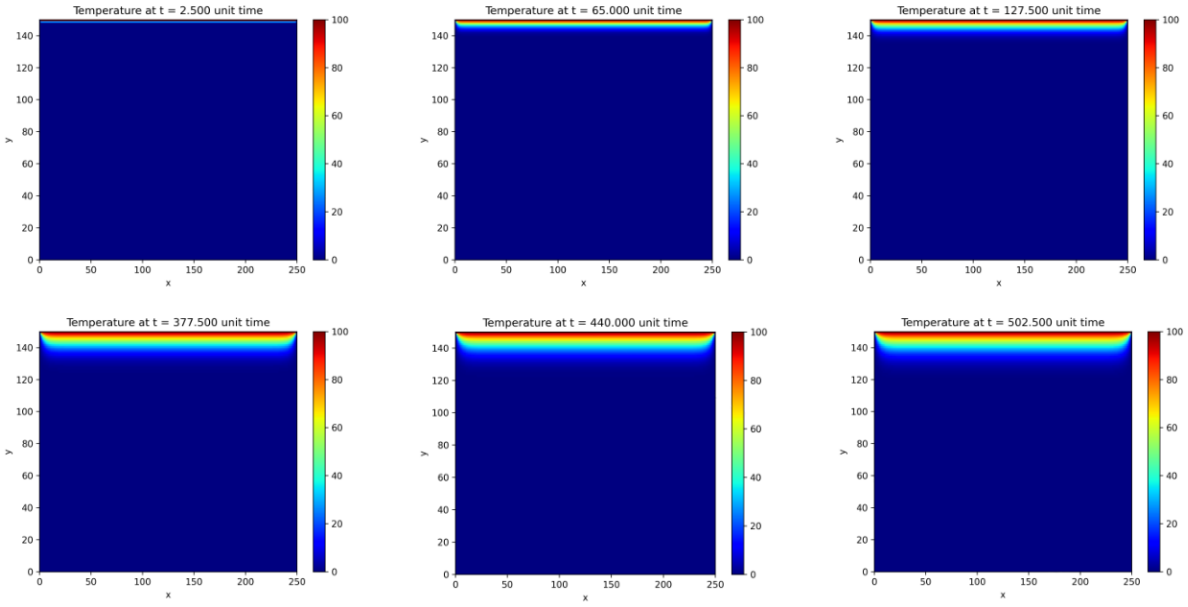

Figure 4.1.1 shows the heat evolution through a rectangular region for the case where $\alpha = 0.1$ and initial temperature distribution $u(x, y, 0) = 0$.

The heat map of a rectangular region with a hot top edge represents the diffusion of heat energy over time. The initial temperature distribution is zero, with a 0.1 thermal diffusivity coefficient.

The heat map displays the temperature distribution over the rectangular region, with colder regions represented by blue and hotter regions by yellow and green colours.

The images in the top row show the initial stages of heat energy diffusion from the hot top edge, gradually increasing the temperature of the nearby cold regions. This observation is consistent with the laws of thermodynamics, which dictate that heat energy will always move from hotter to colder regions (Schilling, Zhang and Bossen, 2019). However, as time progresses, a yellow front emerges, quickly turning to green and extending further into the colder blue regions, as seen in the images from the second row. This observation indicates that the heat energy has spread further into the colder regions, increasing the temperature of these regions. Overall, the heat map of a rectangular region with a hot top edge provides a helpful visualisation of heat energy diffusion over time and shows evidence that it preserves the natural physical laws that govern heat conduction so we will regard the data generated by the finite difference algorithm as the ground truth. Additionally, we will not ignore that other factors, such as thermal diffusivity and varying initial conditions, will affect heat transmission. As such, we will also take training examples generated by varying these parameters. Below are some examples of images generated by varying the thermal diffusivity of the rectangular region.

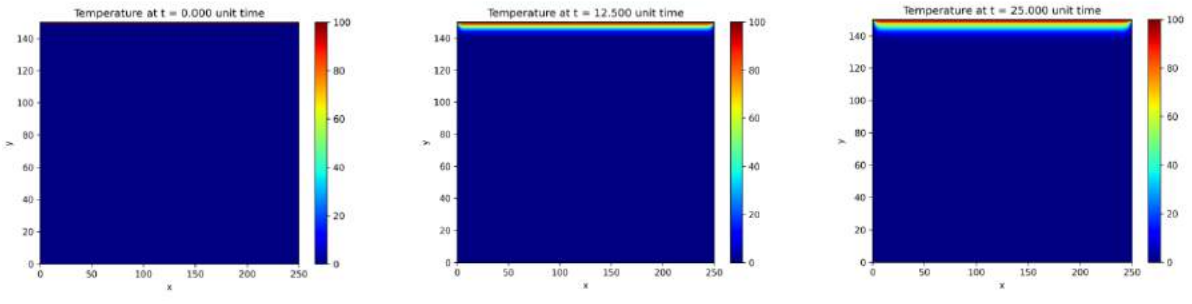

Figure 4.1.2 shows the heat evolution through a rectangular region for the case where $\alpha = 0.5$ and initial temperature distribution $u(x, y, 0) = 0$.

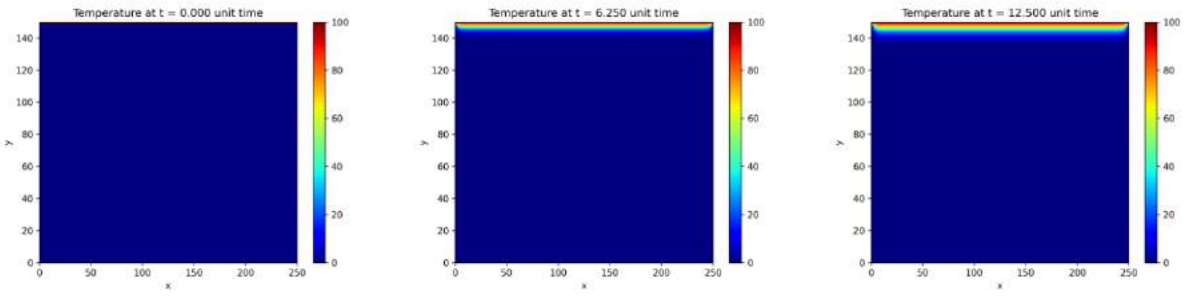

Figure 4.1.3 shows the heat evolution through a rectangular region for the case where α = 1.0 and initial temperature distribution *u(x, y, 0) = 0.*

Upon examining the time stamps, it is evident that enhancing the thermal diffusivity α leads to quicker rates of thermal conduction. Comparing the instances of α = 0.5 and α = 1.0, after a mere 6.25-time unit, the thermal conduction in the case where α = 1.0 is roughly equivalent to the situation where α = 0.5 after 12.5-time units.

We will now look at the effects of varying the initial temperature distribution.

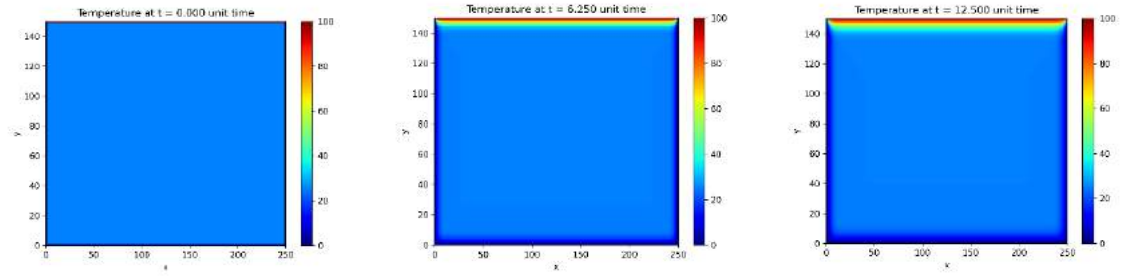

Figure 4.1.4 shows the heat evolution through a rectangular region for the case where α = 1.0 and initial temperature distribution *u(x, y, 0) = 25.*

In Figure 4.1.4, the initial temperature of the rectangular region is 25 ^{0}C at the start of the experiment. From the fundamental laws of physics (that heat transfer rate is directly proportional to the temperature difference (Bejan, 2013)), we can hypothesise that the rate of heat conduction should be lower than the case where the initial temperature distribution is zero.

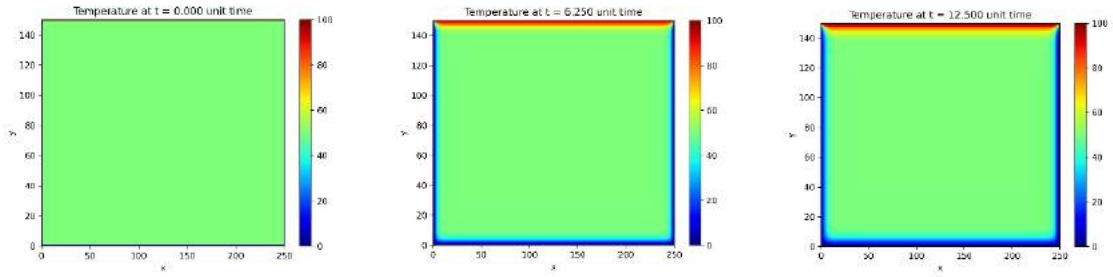

Figure 4.1.5 shows the heat evolution through a rectangular region for the case where α = 1.0 and initial temperature distribution *u(x, y, 0) = 50.*

Therefore, we expect the rate to be even slower in Figure 4.1.5, where the initial temperature value is 50 ^{0}C.

Observing the heat maps shown, we see that heat energy starts to move from the central position towards the cold edges, with the top edge showing the most significant amount of heat conduction. As time progress, the heat energy seems to diffuse at a slower pace towards the cold edges, as seen by more notable colour dispersion near the edges, indicating that the energy flow will eventually reach steady state ($\nabla^2 u = 0$).

Comparing both cases, the observation seem to support our original hypothesis on the rate of heat transfer as the heat conduction in Figure 4.1.4 is seen to be greater than that in Figure 4.15.

4.2. Numerical Results from the PINO Model

The PINO model was trained over 100 epochs with a thermal diffusivity α=1 for various values of the physics loss coefficient. We observed that increasing the number of epochs did not have any significant impact on the results we obtained and so, we choose to fix this parameter. Also, it is standard practice to try and set certain physical constants to one and thereby attain unit unitless parameters to simplify calculations and standardise the experiments (Tegmark et al.,

2006). Following from the previous argument, we chose to fix α at 1 for our experiments. Our results are show and discussed in this section. Additionally, we included some graphs for analysis and comparison. The performance of these models was evaluated using training and test loss per epoch, along with test accuracy measured by the r-squared score.

Experiment A:

Test Number	Physics Loss Coefficient	Training Loss (Loss per Epoch)	Test Accuracy (R^2 Score)	Test Loss (Loss per Epoch)
1	0.001	13.9897	0.9239	13.3647
2	0.010	14.5905	0.9229	13.4685
3	0.100	10.1694	0.9093	19.2715

Table 4.2.1 shows the results from training the PINO model to solve the 2D heat equation with the stochastic gradient descent optimiser, and varying physics loss coefficients.

Experiment A examined three values for the physics loss coefficient, precisely 0.001, 0.010, and 0.100. The results suggest that a lower physics loss coefficient yields better test accuracy (R^2 Score) and lower test loss. The results corroborate this observation by the reporting the highest test accuracy of 0.9239 and the lowest test loss of 13.3647, and both attained at a physics loss coefficient of 0.001.

In the context of training loss, a higher physics loss coefficient correlates with lower training loss values, as evidenced by the lowest training loss of 10.1694 observed at the highest physics loss coefficient of 0.100. However, it is essential to emphasise that a lower training loss does not necessarily ensure better test accuracy or a lower test loss.

Regarding test accuracy (R^{2-}Score), the R^2 score measures the proportion of the variance in the dependent variable that the independent variables can predict. This experiment achieves the highest accuracy score of 0.9239 at the lowest physics loss coefficient (0.001). However, as

the physics loss coefficient increases, the accuracy declines, suggesting that a higher physics loss coefficient might adversely affect the model's generalisation ability.

The test loss demonstrates a similar trend to that of test accuracy. Lower physics loss coefficient values correspond to lower test loss values, with the lowest test loss of 13.3647 occurring at a physics loss coefficient of 0.001. As the physics loss coefficient rises, so does the test loss. This pattern implies that the model may overfit the training data when employing a higher physics loss coefficient, as indicated by the increased test loss of 19.2715 at a physics loss coefficient of 0.100.

Overall, the findings of this experiment suggest that a lower physics loss coefficient is generally more advantageous for achieving better test accuracy and lower test loss. Nonetheless, the relationship between the physics loss coefficient and training loss is more intricate, and a reduced training loss does not guarantee improved model performance. Therefore, additional experimentation with other hyperparameters and optimiser configurations may be necessary to gain further insights into the optimal settings for the physics loss coefficient.

The table below shows the results after training the PINO model with the ADAM optimiser and various physics loss coefficients.

Experiment B:

Test Number	Physics Loss Coefficient	Training Loss (Loss per Epoch)	Test Accuracy (R^2 Score)	Test Loss (Loss per Epoch)
1	0.001	6.3250	0.9491	8.7191
2	0.010	4.5144	0.9716	5.4936
3	0.100	2.9497	0.9714	5.9209

Table 4.2.2 shows the results from training the PINO model to solve the 2D heat equation with the ADAM optimiser, and varying physics loss coefficients.

The physics loss coefficient, a key hyperparameter, influences a model's performance by balancing physics-based constraints and data-driven learning. In Experiment B, we considered three distinct physics loss coefficient values: 0.001, 0.010, and 0.100. In contrast to Experiment A, Experiment B demonstrates that higher physics loss coefficient values resulted in improved test accuracy (R^2 Score) and reduced test loss. For example, we obtained the optimal accuracy score of 0.9716 and the minimum test loss of 5.4936 at a physics loss coefficient of 0.010.

The physics loss coefficient affects training loss values in Experiment B, with higher coefficients leading to lower losses. Consistent with the results from Experiment A, the lowest training loss of 2.9497 is observed at the highest physics loss coefficient of 0.100.

In Experiment B, we obtained the highest accuracy score of 0.9716 with a physics loss coefficient of 0.010. However, the test accuracy scores for physics loss coefficients of 0.010 and 0.100 are similar (0.9716 and 0.9714, respectively), suggesting that the model's generalisation ability is not too sensitive to select a physics loss coefficient of 0.100 as opposed to 0.010.

Regarding test loss, the values in Experiment B decrease as the physics loss coefficient increases to 0.010 but then rise slightly for a physics loss coefficient of 0.100. As a result, we achieved the lowest test loss of 5.4936 at a physics loss coefficient of 0.010. This pattern is distinct from the findings of Experiment A, where the test loss consistently decreased as the physics loss coefficient decreased.

Overall, the outcomes of Experiment B suggest that a physics loss coefficient of 0.010 is generally more favourable for attaining better test accuracy and lower test loss when utilising the ADAM optimiser. This finding contrasts with Experiment A's, where a lower physics loss coefficient proved more advantageous when using the SGD optimiser. Therefore, when

selecting the optimal hyperparameters for a model, it is essential to consider the interactions between the optimiser and the physics loss coefficient. Further experimentation with other hyperparameters and optimiser configurations may offer additional insights into the ideal choices for the physics loss coefficient.

In both experiments, the physics loss coefficient played a vital role in balancing the trade-off between fitting the data and maintaining the physical properties of the model. Lower coefficients, such as 0.001 and 0.01, improve test accuracy, whereas higher coefficients (e.g., 1 and 3) reduce the model's performance in solving PDEs. Comparing the results, we can see that as the physics loss coefficient increases, the training loss per epoch also increases (except for Test 3, which has a lower training loss but higher test loss than Test 1 and Test 2). However, upon comparing the results of Experiment A (SGD optimizer) and Experiment B (ADAM optimizer), we can decide on the best model and associated hyperparameter choices by considering the highest accuracy, lowest test loss, and avoidance of overfitting. Experiment A achieved the best test accuracy (R^2 score) of 0.9239 and the lowest test loss of 13.3647 with a physics loss coefficient of 0.001. However, in Experiment B, the highest test accuracy (R^2 score) of 0.9716 and the lowest test loss of 5.4936 were attained at a physics loss coefficient of 0.010. Figure 6.6 shows the graphs of the best performing model from each of the two experiments.

The model in Experiment B performs better regarding test accuracy and test loss. Additionally, the training loss values in Experiment B are consistently lower than those in Experiment A, which could suggest a more efficient learning process. Finally, the relatively small difference in accuracy scores between physics loss coefficients of 0.010 and 0.100 in Experiment B (0.9716 and 0.9714, respectively) indicates that the model may not be prone to overfitting when using the ADAM optimizer, even with a higher physics loss coefficient.

Considering these observations, the best model is the one from Experiment B, which utilizes the ADAM optimizer with a physics loss coefficient of 0.010. This choice leads to higher accuracy and lower test loss while demonstrating a robust performance less susceptible to overfitting.

The results from both experiments show that smaller physics loss coefficients yield better test accuracies, it does not necessarily mean that the physical laws are less important or unnecessary for the best results. The physics loss coefficient is crucial in balancing the data fitting and enforcing physical properties in the PINO models. A smaller coefficient indicates that the model focuses more on fitting the data, while a more significant coefficient emphasizes enforcing the physical laws. The fact that the best-performing models have coefficients within the range of 0.001 and 0.01 suggests an optimal balance between fitting the data and incorporating physical constraints. This balance allows the model to achieve better generalization and performance while accounting for the underlying physical principles. Additionally, physical laws are still, to some extent, enforced even with the smaller coefficients, the physical laws are still being implemented. The physics loss coefficient weighs the importance of the physical laws relative to the data fitting in the loss function. A smaller coefficient does not imply that the physical laws are irrelevant; instead, it indicates that the model requires a specific balance between data fitting and enforcing the physical laws to achieve optimal performance.

Overall, the model is performing increasingly well with time, with decreasing training and test errors. But the level of overfitting observed increases with the increase in the physics loss coefficient. The graphs below show this very well.

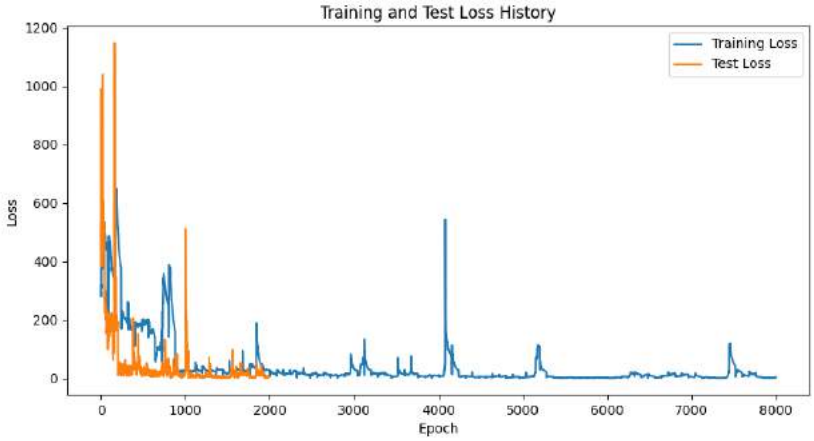

Figure 4.2.1a shows the training and test loss history for the best model from Experiment A.

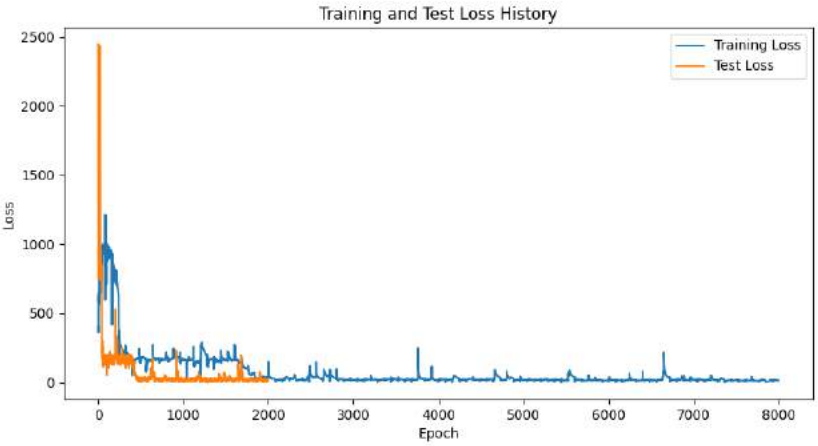

Figure 4.2.1b shows the training and test loss history for the best model from Experiment B.

Figure 4.2.1 shows the graph comparing the training and test loss per epoch against epochs for physics loss for the best model from Experiment A and Experiment B.

Analysing the graphs in Figure 4.2.1 above, the model from Experiment B is the best model of the two, showing lower overall test loss (shorter orange spikes), higher accuracy and looks less likely to suffer from overfitting (lower frequency of orange spikes).

The comparative heatmap image showcases the predicted solution of the PDE using the model and the ground truth (heatmap generated by the finite difference method). In addition,

the image provides a visual comparison of the model's performance in predicting the PDE solution and the distribution of errors between the predicted solution and the ground truth.

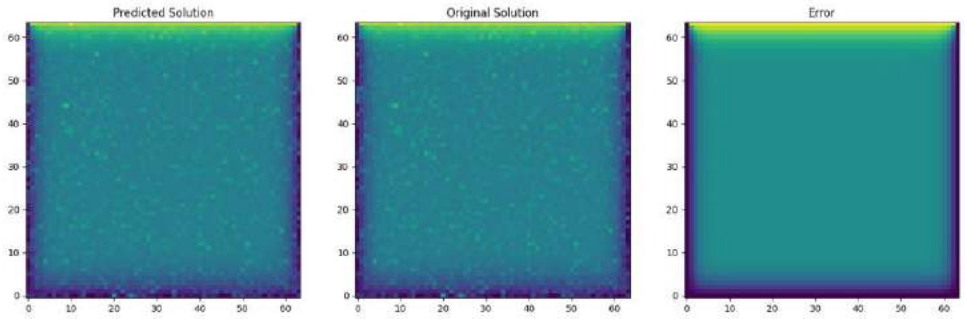

Figure 4.2.2 shows a comparison between the prediction of the best model and the corresponding ground truth, along with the error distribution associated with making the prediction at time step 0.

Upon inspection, the predicted heatmap appears almost identical to the ground truth, indicating that the model is highly accurate in capturing the underlying behaviour of the PDE. This similarity suggests that the model has successfully learned the relationships and patterns within the data and can generalise to new, unseen data points.

The error distribution heatmap, predominantly cool with a uniform light shade of blue centrally and darker blues at the left, right, and bottom edges, reveals that the model's predictions are mostly accurate. The cool colours imply that the errors are relatively low in these areas. The presence of light shades of yellow and green at the top indicates slightly higher errors in this region compared to the rest of the heatmap. However, these errors still appear relatively small and do not significantly impact the overall accuracy of the predicted solution.

5. Conclusion

This study introduced a novel Physics-Informed Neural Operator (PINO) framework for solving partial differential equations (PDEs), which combines the strengths of numerical simulations and neural networks. By incorporating Fourier analysis techniques and a custom loss function, our approach effectively balances the trade-off between data fitting and enforcing physical properties. Through a comprehensive comparative study, we demonstrated the significant impact of varying the physics loss coefficient on the performance of PINO models for solving PDEs. Our experimental results revealed that the optimal range for the physics loss coefficient lies between 0.001 and 0.01, with smaller coefficients yielding better test accuracies. This finding suggests an optimal balance between fitting the data and incorporating physical constraints, allowing the PINO model to achieve better generalization and performance while accounting for underlying physical principles. However, the results also highlighted that the performance of PINO models decreases when the physics loss coefficient is increased beyond this optimal range, indicating that overemphasizing the physics loss may harm the model's performance in solving PDEs.

This study sheds light on the influence of the physics loss coefficient on the performance of PINO models. Nevertheless, manual selection and fine-tuning of this coefficient can be laborious and might not guarantee optimal outcomes. Therefore, future research could consider developing methods for automatically selecting the ideal physics loss coefficient value for a specific problem by employing adaptive learning rates or other hyperparameter optimization techniques. While our investigation centred on the physics loss coefficient as a crucial hyperparameter in PINO models, other hyperparameters like network architecture, learning rate, and activation functions can also significantly impact performance. Subsequent work

could systematically examine the effect of these hyperparameters on PINO models for solving PDEs and determine optimal configurations for various problem contexts.

The current study focused on applying PINO models to the two-dimensional heat conduction equation. However, real-world issues often entail multiple physical processes governed by coupled PDEs. Expanding the PINO framework to accommodate multi-physics problems and examining the impact of physics loss coefficients on these more complex situations could constitute a valuable research direction.

In practical applications, complete and noise-free data are often scarce. Upcoming research could explore the resilience of PINO models in handling incomplete and noisy data and devise methods to enhance the model's performance in such challenging situations. This issue could be tackled by integrating data assimilation techniques or implementing Bayesian approaches within the PINO framework.

Model generalization and transfer learning are critical aspects of the practical utility of PINO models. Future studies could delve into the factors affecting the generalization capabilities of PINO models and investigate the application of transfer learning techniques to improve their performance on related tasks or when data is limited.

In conclusion, this study introduces a promising approach for solving PDEs using PINO models, providing valuable insights into the role of physics loss coefficients in balancing data fitting and physical properties. Moreover, the results lay the foundation for further exploring PINO model performance and developing more advanced techniques to enhance their applicability and robustness in addressing complex real-world problems.

6. Reference

1. Aslak Tveito and Winther, R. (2008). *Introduction to Partial Differential Equations*. Springer Science & Business Media.

2. Ayensa-Jiménez, J., Doweidar, M.H., Sanz-Herrera, J.A. and Doblaré, M. (2021). Prediction and identification of physical systems by means of Physically Guided Neural Networks with meaningful internal layers. *Computer Methods in Applied Mechanics and Engineering*, 381(1), p.113816.

3. Ayensa-Jiménez, J., Sanz-Herrera, J.A., Doweidar, M.H. and Doblaré, M. (2020). IDENTIFICATION OF STATE FUNCTIONS BY PHYSICALLY-GUIDED NEURAL NETWORKS WITH PHYSICALLY-MEANINGFUL INTERNAL LAYERS. [online] Available at: https://arxiv.org/pdf/2011.08567.pdf [Accessed 30 Dec. 2022].

4. Bejan, A. (2013). *Convection Heat Transfer*. John Wiley & Sons.

5. Bracewell, R. (2000). *The Fourier Transform and Its Applications*. 3rd ed. McGraw-Hill Science, Engineering & Mathematics.

6. Burden, R.L. and J. Douglas Faires (2010). *Numerical Analysis*. Cengage Learning.

7. Chen, X., Chen, X., Zhou, W., Zhang, J. and Yao, W. (2020). The heat source layout optimization using deep learning surrogate modeling. *Structural and Multidisciplinary Optimization*, 62(6), pp.3127–3148.

8. Degen, D., Cacace, M. and Wellmann, F. (2022). 3D multi-physics uncertainty quantification using physics-based machine learning. *Scientific Reports*, 12(1).

9. Gooch, J.W. (2011). Fourier's Law of Heat Conduction. *Encyclopedic Dictionary of Polymers*, pp.323–323.

10. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, 63(11), pp.139–144.

11. Goswami, S., Bora, A., Yu, Y. and Karniadakis, G.E. (2022). Physics-Informed Deep Neural Operator Networks. [online] Available at: https://doi.org/10.48550/arXiv.2207.05748 [Accessed 26 Dec. 2022].

12. Han, J. and Jentzen, A. (2017). *Deep learning-based Numerical Methods for high-dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations*. [online] Communications in Mathematics and Statistics, pp.349–380. Available at: https://arxiv.org/pdf/1706.04702.pdf [Accessed 2 Mar. 2023].

13. Han, J., Jentzen, A. and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34), pp.8505–8510.

14. Hendrycks, D. and Gimpel, K. (2018). *GAUSSIAN ERROR LINEAR UNITS (GELUS)*. [online] Available at: https://arxiv.org/pdf/1606.08415v3.pdf [Accessed 5 Mar. 2023].

15. Henner, V., Belozerova, T. and Nepomnyashchy, A. (2019). *Partial Differential Equations*. CRC Press.

16. Kamrava, S., Sahimi, M. and Tahmasebi, P. (2021). Simulating fluid flow in complex porous materials by integrating the governing equations with deep-layered machines. *npj Computational Materials*, 7(1), pp.7–127.

17. Katsikis, D., Muradova, A.D. and Stavroulakis, G.E. (2022). A Gentle Introduction to Physics-Informed Neural Networks, with Applications in Static Rod and Beam Problems. *Journal of Advances in Applied & Computational Mathematics*, 9, pp.103–128.

18. Kreyszig, E. (2019). *Advanced Engineering Mathematics*. 10th ed. New York, United States of America: Wiley.

19. LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep Learning. *Nature*, [online] 521(7553), pp.436–444. Available at: https://www.nature.com/articles/nature14539.

20. Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. and Anandkumar, A. (2021). *FOURIER NEURAL OPERATOR FOR PARAMETRIC PARTIAL DIFFERENTIAL EQUATIONS*. arXiv preprint arXiv:2010.08895.

21. Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K. and Anandkumar, A. (2022). Physics-Informed Neural Operator for Learning Partial Differential Equations. [online] Available at: https://arxiv.org/pdf/2111.03794 [Accessed 1 Jan. 2023].

22. Luciano Maria Barone, Marinari, E., Organtini, G. and Federico Ricci Tersenghi (2013). *Scientific Programming*. World Scientific.

23. Maxwell, R.M., Condon, L.E. and Melchior, P. (2021). A Physics-Informed, Machine Learning Emulator of a 2D Surface Water Model: What Temporal Networks and Simulation-Based Inference Can Help Us Learn about Hydrologic Processes. *Water*, 13(24), p.3633.

24. McGowan, E., Gawade, V. and Guo, W. (Grace) (2022). A Physics-Informed Convolutional Neural Network with Custom Loss Functions for Porosity Prediction in Laser Metal Deposition. *Sensors*, 22(2), p.494.

25. Narasimhan, T.N. (1999). Fourier's heat conduction equation: History, influence, and connections. *Journal of Earth System Science*, 108(3), pp.117–148.

26. Ngom, M. and Marin, O. (2021). Fourier neural networks as function approximators and differential equation solvers. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 14(6), pp.647–661.

27. Oden, J.T. (2013). *Finite Elements of Nonlinear Continua*. Courier Corporation.

28. Oldenburg, J., Borowski, F., Öner, A., Schmitz, K.-P. and Stiehm, M. (2022). Geometry aware physics informed neural network surrogate for solving Navier–Stokes equation (GAPINN). *Advanced Modeling and Simulation in Engineering Sciences*, 9(8).

29. Powers, D.L. (2009). *Boundary Value Problems*. Academic Press.

30. PyTorch (2019). *PyTorch*. [online] Pytorch.org. Available at: https://pytorch.org/.

31. Raissi, M. and Karniadakis, G.E. (2018). Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357, pp.125–141.

32. Raissi, M., Perdikaris, P. and Karniadakis, G.E. (2019). Physics-informed Neural networks: a Deep Learning Framework for Solving Forward and Inverse Problems Involving

Nonlinear Partial Differential Equations. *Journal of Computational Physics*, 378, pp.686–707.

33. Raissi, M., Yazdani, A. and Karniadakis, G.E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481), pp.1026–1030.

34. Rosofsky, S.G., Al Majed, H., and Huerta, E.A. (2022). Applications of physics informed neural operators. [online] Available at: https://arxiv.org/pdf/2203.12634.pdf [Accessed 26 Dec. 2022].

35. Schilling, A., Zhang, X. and Bossen, O. (2019). Heat flowing from cold to hot without external intervention by using a 'thermal inductor'. *Science Advances*, [online] 5(4), p.eaat9953. Available at: https://advances.sciencemag.org/content/5/4/eaat9953.

36. Sharda, R., Dursun Delen and Efraim Turban (2021). *Analytics, data science, & artificial intelligence : systems for decision support*. Harflow, Essex, United Kingdom: Pearson Education Limited.

37. Sharma, L., and Pradeep Kumar Garg (2021). *Artificial Intelligence*. CRC Press.

38. Sirignano, J. and Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375, pp.1339–1364.

39. Spyder Team (2018). *Spyder Website*. [online] Spyder-ide.org. Available at: https://www.spyder-ide.org/.

40. Tegmark, M., Aguirre, A., Rees, M.J. and Wilczek, F. (2006). Dimensionless constants, cosmology, and other dark matters. *Physical Review D*, 73(2).

41. Tenenbaum, M. and Pollard, H. (1985). *Ordinary differential equations : an elementary textbook for students of mathematics, engineering, and the sciences*. New York: Dover Publications.

42. Vadyala, S.R., Betgeri, S.N. and Betgeri, N.P. (2022). Physics-informed neural network method for solving one-dimensional advection equation using PyTorch. *Array*, 13(2022), p.100110.

43. Wang, C., Bentivegna, E., Zhou, W., Klein, L.J. and Elmegreen, B. (2020a). Physics-Informed Neural Network Super Resolution for Advection-Diffusion Models. *: Neural Information Processing Systems (NeurIPS 2020) Workshop*. [online] Available at: https://arxiv.org/pdf/2011.02519 [Accessed 30 Dec. 2022].

44. Wang, P., Sun, M., Wang, H., Li, X. and Yang, Y. (2020b). Convolution operators for visual tracking based on spatial–temporal regularization. *Neural Computing and Applications*, 32(10), pp.5339–5351.

45. Wang, S., Wang, H. and Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics informed DeepONets. *Science Advances*, 7(40).

46. Wang, S., Yu, X. and Perdikaris, P. (2022). When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449, p.110768.

47. Wiens, J., and Shenoy, E.S. (2017). Machine Learning for Healthcare: On the Verge of a Major Shift in Healthcare Epidemiology. *Clinical Infectious Diseases*, 66(1), pp.149–153.

48. Yang, L., Zhang, D. and Karniadakis, G.E. (2020). Physics-Informed Generative Adversarial Networks for Stochastic Differential Equations. *SIAM Journal on Scientific Computing*, [online] 42(1), pp.A292–A317. Available at: https://arxiv.org/pdf/1811.02033.pdf [Accessed 13 Jan. 2022].

49. You, H., Zhang, Q., Ross, C.J., Lee, C.-H. and Yu, Y. (2022). Learning deep Implicit Fourier Neural Operators (IFNOs) with applications to heterogeneous material modeling. *Computer Methods in Applied Mechanics and Engineering*, 398, p.115296.

50. Zhao, X., Gong, Z., Zhang, Y., Yao, W. and Chen, X. (2021). Physics-informed Convolutional Neural Networks for Temperature Field Prediction of Heat Source Layout without Labeled Data. *Engineering Applications of Artificial Intelligence*, 117, p.105516.

51. Zhu, Y., Zabaras, N., Koutsourelakis, P.-S. and Perdikaris, P. (2019). Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, [online] 394, pp.56–81. Available at: https://www.sciencedirect.com/science/article/pii/S0021999119303559 [Accessed 14 Aug. 2021].