eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
FACULTY OF SCIENCE AND TECHNOLOGY
UNIVERSITY OF THE BASQUE COUNTRY

Master Thesis

Master in Mathematical Modeling, Research, Statistics, and Computation

On Solving Partial Differential Equations (PDEs) Using Physics-Informed Neural Networks (PINNs)

Jose García Ventura

Advisors
Carlota María Cuesta Romero
Francisco de la Hoz Méndez

June 27, 2025

Abstract

The fusion between AI and physics holds great promise across various scientific disciplines. Physics-informed neural networks (PINNs) have emerged as a versatile and widely applicable concept within the realm of artificial intelligence, impacting various science and engineering domains over the past decade. This work offers a comprehensive overview of the fundamentals of PINNs, tracing their evolution, modifications, and various applications, focusing on the linear, second-order parabolic heat equation, the nonlinear reaction-diffusion Allen–Cahn equation, and the nonlinear, dispersive, third-order Korteweg–de Vries (KdV) equation. The behavior of PINNs under these governing equations is presented while using classical numerical solvers as comparison points, as well as the difficulties encountered while training the networks. Additionally, it identifies current gaps in the research and outlines future directions for the continued development of PINNs. Ultimately, PINNs are positioned as promising tools for a new computational paradigm, enabling significant advancements across diverse scientific and engineering domains.

Keywords: Physics-informed neural networks (PINNs), partial differential equations (PDEs), scientific computing, heat equation, Allen–Cahn equation, Korteweg–de Vries (KdV) equation.

La fusión entre IA y física es sumamente prometedora en diversas disciplinas científicas. Las redes neuronales informadas por la física (PINN) han emergido como un concepto versátil y ampliamente aplicable en el ámbito de la inteligencia artificial, impactando varios dominios de la ciencia y la ingeniería durante la última década. Este trabajo ofrece una visión general de sus fundamentos, rastreando su evolución, modificaciones y diversas aplicaciones, con un enfoque en la ecuación de calor parabólico lineal de segundo orden, la ecuación de Allen-Cahn de reacción-difusión no lineal y la ecuación de Korteweg-de Vries (KdV) no lineal, de tercer orden. Se presenta el comportamiento de las PINN bajo estas ecuaciones, utilizando métodos numéricos clásicos como puntos de comparación, así como las dificultades encontradas durante su entrenamiento. Además, se identifican las lagunas actuales en la investigación y se sugieren futuras direcciones para su desarrollo continuo. Finalmente, las PINN se posicionan como herramientas versátiles para un nuevo paradigma computacional, con la promesa de avances significativos en diversas disciplinas científicas y de ingeniería.

Palabras clave: Redes neuronales informadas por la física (PINNs), ecuaciones en derivadas parciales (EDP), computación científica, ecuación del calor, ecuación de Allen-Cahn, ecuación de Korteweg–de Vries (KdV).

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Recent advances in deep learning are reshaping computational physics and engineering. When trying to solve partial differential equations (PDEs), classical discretisation schemes struggle with noisy or sparse data and with the curse of dimensionality [5]. Physics-informed neural networks (PINNs) address this gap by embedding the physical governing laws of a given PDE directly in the training objective, yielding mesh-free, data-efficient PDE surrogates [6, 7, 1].

1.1 Context and Motivation

This work advocates a synergy between machine learning and classical computational physics that has the potential to enrich both fields and lead to high-impact developments. PINNs provide a new, different way of solving differential equations when carrying out simulations. They are especially useful when we don't know the solution to the differential equation, and where we are trying to estimate the parameters that govern the evolution of an approximated solution [8]. This makes PINNs a powerful tool to investigate because it has the potential to revolutionize physics research workflows by enabling experimental design for inverse problems [9], automated discovery of new governing systems equations [1], and optimizing the design and efficiency of structures exposed to fluid flows, such as bridges, offshore platforms, and wind turbine blades [10]. These are just a few of the many applications of PINNs, which will be discussed further in Chapter 2.

1.1.1 Why Physics-Informed Neural Networks?

Unlike classical numerical methods, such as finite-difference [11], PINNs are mesh-free, differentiable, and highly flexible. This flexibility allows PINNs to handle complex geometries, sparse and noisy observational data, and inverse problems, where the solution to the differential equation is unknown, and parameters governing the system behavior need to be estimated [8, 1].

Moreover, PINNs are particularly useful over purely data-driven approaches that require

extensive datasets, which are often unavailable or impractical in many scientific domains [5]. By integrating physical laws as constraints within the neural network training, PINNs significantly reduce the dependency on large datasets, thereby making them data-efficient. Their ability to encode domain-specific knowledge directly into the learning process ensures physically consistent predictions and robust extrapolation to scenarios beyond training data [6, 12].

Industries such as aerospace and biomedical engineering have already begun getting significant benefits from these advancements. Designers in aerospace engineering can optimize aerodynamic properties with enhanced precision [10], while biomedical researchers leverage PINNs for improved modeling of complex biological processes, including cardiac activation mapping and tissue simulations [13, 14].

These are just a few of the many applications emerging from developments in PINNs, with more to be discussed in Chapter 2.

1.2 Problem Statement

In this thesis, we will specifically address three types of PDEs: the linear second-order parabolic heat equation, the nonlinear reaction-diffusion Allen–Cahn equation, and the nonlinear dispersive third-order Korteweg–de Vries (KdV) equation. Each class presents distinct challenges, from linear diffusion behaviors to complex nonlinear dynamics involving sharp interfaces and wave dispersion that the PINN must be able to learn. Traditional numerical solvers often struggle with these complexities, especially in higher-dimensional or noisy data scenarios as will be shown in Chapter 5. For this reason, a comparison against classical numerical solvers is done, where we present the solutions provided by both methods, and present the gathered conclusions.

1.3 Objectives and Scope

The main objectives of this thesis are to:

- Develop a unified PINN methodology applicable across linear and nonlinear PDEs.
- Evaluate PINN performance against classical finite-difference and spectral numerical methods.
- Investigate the impact of network architecture, loss weighting, and sampling strategies on convergence and accuracy of the obtained PDE solution.
- Identify current limitations and propose improvements that can be tackled with future research projects.

Following this introduction, Chapter 2 provides a detailed literature review covering traditional numerical solvers and recent advances in machine learning-based PDE solvers,

particularly PINNs. Chapter 3 introduces the mathematical formulation of the PDEs addressed in this thesis, detailing their physical contexts and baseline numerical methods. Chapter 4 presents the PINN methodology, describing the neural network architecture, training procedures, and hyperparameter choices. Chapter 5 discusses the results obtained by applying PINNs to each PDE class, comparing performance, and identifying strengths and limitations. Finally, Chapter 6 summarizes the thesis, highlighting key insights, practical challenges, and avenues for future research.

Chapter 2

Literature Review

This chapter reviews both foundational and emerging approaches for solving partial differential equations (PDEs). It first examines classical numerical discretization techniques such as finite-difference methods (FDM), then cover early neural-network PDE solvers and operator-learning frameworks. The core of the chapter focuses on Physics-Informed Neural Networks (PINNs), presenting their formulation, key variants, and diverse applications. The chapter concludes with a comparative analysis of the methods' respective strengths, limitations, and open challenges.

2.1 Classical Numerical Methods

Classical numerical methods, such as finite-difference, finite element, and Runge-Kutta time-integration schemes, have formed the backbone of PDE discretization for decades [11, 15, 16]. These approaches discretize space and time into algebraic updates: spatial derivatives are replaced by stencils or weak forms over a mesh. Simultaneously, time integration is handled via the method of lines (MOL), which reduces a PDE to an ordinary differential equation (ODE) initial-value problem (IVP) requiring a prescribed state $y(t_0) = y_0$ [17, 18, 19]. The simplest time integrator is the first-order forward Euler method:

$$u^{n+1} = u^n + \Delta t\, f(u^n, t^n)\,, \tag{2.1}$$

which has local truncation error $O(\Delta t^2)$ [20]. Higher-order accuracy is achieved via Runge-Kutta methods. Runge-Kutta methods are a class of methods which use the information on the slope at more than one point to extrapolate the solution to the future time step. A commonly used variant is the classical fourth-order scheme (RK4), whose local truncation error is $O(h^5)$ [20]. Given an initial-value problem (IVP)

$$\frac{dy}{dt} = f(y,t), \quad y(t_n) = y_n, \tag{2.2}$$

and a time step h, and the solution y_n at the nth time step, the RK4 update for y_{n+1} reads

$$k_1 = h\, f(y_n, t_n), \tag{2.3}$$
$$k_2 = h\, f\big(y_n + \tfrac{1}{2}k_1,\ t_n + \tfrac{1}{2}h\big), \tag{2.4}$$
$$k_3 = h\, f\big(y_n + \tfrac{1}{2}k_2,\ t_n + \tfrac{1}{2}h\big), \tag{2.5}$$
$$k_4 = h\, f(y_n + k_3,\ t_n + h), \tag{2.6}$$
$$y_{n+1} = y_n + \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}\,. \tag{2.7}$$

As an explicit one-step method, RK4 is only conditionally stable and requires suitable timestep choices in stiff or highly oscillatory settings [21].

2.1.1 Finite-Difference Methods

Finite-Difference Methods (FDM) approximate derivatives using discrete stencils on a structured grid [11, 19]. For a function $u(x)$ sampled at $x_i = x_0 + i\,\Delta x$:

$$\left.\frac{\partial u}{\partial x}\right|_{x_i} \approx \frac{u_{i+1} - u_i}{\Delta x} + O(\Delta x), \tag{2.8}$$
$$\left.\frac{\partial u}{\partial x}\right|_{x_i} \approx \frac{u_{i+1} - u_{i-1}}{2\,\Delta x} + O(\Delta x^2), \tag{2.9}$$
$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2} + O(\Delta x^2). \tag{2.10}$$

Applying these to the 1D heat equation $u_t = \alpha\, u_{xx}$ yields the explicit scheme

$$u_i^{n+1} = u_i^n + \frac{\alpha\,\Delta t}{\Delta x^2}\big(u_{i+1}^n - 2u_i^n + u_{i-1}^n\big). \tag{2.11}$$

Stability is often assessed via Von Neumann (Fourier) analysis [19]. Implicit variants, such as Crank-Nicolson, lift this timestep restriction at the cost of solving linear systems [22]. One of the first systematic finite-difference implementations dates to 1911 [23].

As a specific example, consider the flow of heat in a rod made from some heat-conducting material, subject to some external heat source along its length and some boundary conditions at each end. If we assume that the material properties, the initial temperature distribution, and the source vary only with x, the distance along the length, and not across any cross section, then we expect the temperature distribution at any time to vary only with x and we can model this with a differential equation in one space dimension [19].

Since the solution might vary with time, we let $u(x, t)$ denote the temperature at point x at time t, where $a < x < b$ along some finite length of the rod. The solution is then governed by the heat equation

$$u_t(x,t) \;=\; \big(\kappa(x)\, u_x(x,t)\big)_x + \psi(x,t), \tag{2.12}$$

where $\kappa(x)$ is the coefficient of heat conduction, which may vary with x, and $\psi(x,t)$ is the heat source (or sink, if $\psi < 0$) [19]. The diffusion of heat is just one of the many problems that can be solved applying a finite-difference method.

If the material is homogeneous, then $\kappa(x) \equiv \kappa$ is independent of x and the heat equation 2.12 reduces to

$$u_t(x,t) = \kappa\, u_{xx}(x,t) + \psi(x,t). \tag{2.13}$$

Along with the equation, we need initial conditions,

$$u(x,0) = u^0(x), \tag{2.14}$$

and boundary conditions, for example, the temperature might be specified at each end,

$$u(a,t) = \alpha(t), \quad u(b,t) = \beta(t). \tag{2.15}$$

Such boundary conditions, where the value of the solution itself is specified, are called Dirichlet boundary conditions. Alternatively, one end, or both ends, might be insulated, in which case there is zero heat flux at that end, and so $u_x = 0$ at that point. This boundary condition, which is a condition on the derivative of u rather than on u itself, is called a Neumann boundary condition. To begin, we will consider the Dirichlet problem for 2.13 with boundary conditions 2.15 [19].

In general, we expect the temperature distribution to change with time. However, if $\psi(x,t)$, $\alpha(t)$, and $\beta(t)$ are all time independent, then we might expect the solution to eventually reach a steady-state solution $u(x)$, which then remains essentially unchanged at later times. Usually there will be an initial transient time, as the initial data $u^0(x)$ approach $u(x)$ (unless $u^0(x) \equiv u(x)$), but if we are interested only in computing the steady-state solution itself, then we can set $u_t = 0$ in 2.13 and obtain an ODE in x to solve for $u(x)$:

$$u''(x) = f(x), \tag{2.16}$$

where $f(x) = -\psi(x)/\kappa$. This is a second-order ODE, and from basic theory we expect to need two boundary conditions to specify a unique solution. In our case we have the boundary conditions

$$u(0) = \alpha, \quad u(1) = \beta. \tag{2.17}$$

The problem in 2.16, 2.17 is called a 2-point boundary-value problem (BVP), since one condition is specified at each of the two endpoints of the interval where the solution is desired.

If instead two data values were specified at the same point, say $u(a) = \alpha$, $u'(a) = \sigma$, and we wanted to find the solution for $t \geq a$, then we would have an initial-value problem (IVP) instead [19].

In order to apply a finite-difference method here for the 2-point BVP, we will attempt to compute a grid function consisting of values $\{U_0, U_1, \ldots, U_m, U_{m+1}\}$, where U_j is our approximation to the solution $u(x_j)$. Here $x_j = jh$ and $h = \frac{1}{m+1}$ is the mesh width, the distance between grid points [19]. From the boundary conditions, we know that $U_0 = \alpha$ and $U_{m+1} = \beta$, and so we have m unknown values $U_1, \ldots, U_m$ to compute. If we replace $u''(x)$ in 2.16 by the centered-difference approximation

$$D^2 U_j \;=\; \frac{1}{h^2}\big(U_{j-1} - 2\,U_j + U_{j+1}\big), \tag{2.18}$$

then we obtain a set of algebraic equations

$$\frac{1}{h^2}\big(U_{j-1} - 2\,U_j + U_{j+1}\big) \;=\; f(x_j) \quad \text{for } j = 1, 2, \ldots, m. \tag{2.8}$$

We have a linear system of m equations for the m unknowns, which can be written in the form

$$A\,U = F \tag{2.19}$$

where U is the vector of unknowns

$$U = \begin{bmatrix} U_1 \\ U_2 \\ \vdots \\ U_m \end{bmatrix} \tag{2.20}$$

and

$$A = \frac{1}{h^2}\begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix}, \quad F = \begin{bmatrix} f(x_1) \;-\; \alpha/h^2 \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{m-1}) \\ f(x_m) \;-\; \beta/h^2 \end{bmatrix}. \tag{2.21}$$

This tridiagonal linear system is nonsingular and can be easily solved for U given any right-hand side F. The goal then simply becomes to compute the magnitude of the error vector of the solution, given by $E = U_j - u(x_j)$ [19].

2.1.2 Spectral Methods

The term spectral method generally refers to a numerical method that is capable (under suitable smoothness conditions) of converging at a rate that is faster than polynomial in the mesh width h. Originally the term was more precisely defined. In the classical spectral method, the solution to the differential equation is approximated by a function $U(x)$ that is a linear combination of a finite set of orthogonal basis functions [19].

Spectral methods can approximate the solution by a finite expansion in differentiable basis functions, often Fourier modes for periodic domains or Chebyshev polynomials on tensor-product grids [19, 24]. One writes the trial solution as

$$U(x) \;=\; \sum_{j=0}^{N} c_j \, \phi_j(x)\,, \tag{2.22}$$

and determines the coefficients c_j to minimize an appropriate norm of the residual function by enforcing that the residual

$$\mathcal{R}(x) = \mathcal{L}[U](x) - f(x) \tag{2.23}$$

becomes orthogonal to a chosen set of test functions. Depending on that choice one recovers the classical Galerkin, collocation or tau formulations of spectral methods. Under sufficient smoothness, spectral schemes converge faster than any fixed algebraic order in the mesh size h, in stark contrast to the $O(h^p)$ rates of h-refinement in finite elements or the local-stencil accuracy of finite differences [19]. The classical Galerkin formulation dates to the early 20th century, collocation and tau variants emerged in the 1930s and 1960s, respectively. Modern treatments exploit fast transforms and domain decomposition to handle complex geometries and localized features [24].

2.2 Machine Learning Approaches to PDEs

Physics-informed machine learning (PIML) is a subfield of machine learning where we are trying to build physical models from data via optimization.

Early PIML solvers tackled PDEs by embedding the differential operator into the loss function, training multilayer perceptrons to satisfy both the equation and boundary conditions at collocation points [25]. While they demonstrated the feasibility of mesh-free approximations, convergence and training stability were often problematic.

More recently, operator-learning frameworks such as DeepONets extend neural approaches to learn mappings between function spaces, enabling rapid evaluation of a family of PDEs once trained [26]. An operator is a mapping that takes an entire function as its input and produces another function as its output, like a function-to-function transformation. For example, the differential operator

$$\mathcal{L}[u](x) = u''(x) \tag{2.24}$$

maps the function $u(x)$ to its second derivative $u''(x)$, and the solution operator of a PDE maps a coefficient or source term function to the corresponding solution function. Operator-learning frameworks like DeepONets aim to learn such mappings directly from data. These models leverage the universal approximation theorem for operators [27] but require large datasets of paired input-output functions and careful design to generalize across different parameter regimes.

Despite great progress in simulating multiphysics problems using the numerical discretization of partial differential equations (PDEs), incorporating noisy data into existing algorithms remains challenging. Mesh generation remains complex, and high-dimensional problems governed by parameterized PDEs cannot be tackled [5]. Machine learning has emerged as a promising alternative, but training deep neural networks requires big data, not always available for scientific problems. Instead, such networks can be trained from additional information obtained by enforcing the physical laws (for example, at random points in the continuous space-time domain). Such physics-informed learning integrates (noisy) data and mathematical models and implements them through neural networks [6].

2.3 Physics-Informed Neural Networks (PINNs)

Physics-Informed Neural Networks are a kind of neural network that is trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations [8]. The result is a class of function approximation neural network that allows you to encode an underlying set of physical constraints defined by the PDE. We are then able to use these networks as surrogate models to predict a solution to a PDE while enforcing the physical laws, as these models become fully differentiable with respect to the input coordinates and parameters.

As a result, PINNs have emerged as a key tool in scientific machine learning since their introduction in 2017 [7, 1], enabling the efficient solution of ordinary/partial differential equations using sparse measurements [2].

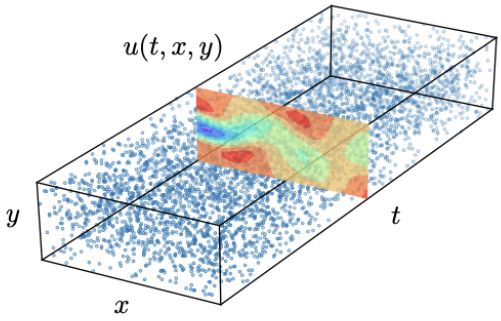
Consider the general two-dimensional PDE in the domain $\Omega \subset \mathbb{R}^d$

$$\mathcal{F}\left(x_1, x_2, u(x_1, x_2), \frac{\partial u(x_1, x_2)}{\partial x_1}, \ldots, \frac{\partial^2 u(x_1, x_2)}{\partial x_1 \partial x_2}, \ldots, \frac{\partial^k u(x_1, x_2)}{\partial x_2^k}; \lambda\right) = 0\,, \tag{2.25}$$

where $\mathcal{F}$ can be a possibly nonlinear operator with partial derivatives up to some order k and parametrization parameter λ [3]. In a PINN, we approximate the solution not only by using a neural network

$$u(x) \approx u_\theta(x), \tag{2.26}$$

Figure 2.1 Collocation points illustrating that denser sampling improves accuracy but slows training (found in [1]).



to learn the trainable parameters θ, but also by adding physics-based constraints to the loss function. To enforce physics, we use automatic differentiation frameworks (e.g. TensorFlow [28] or PyTorch [29]) to compute the network derivatives $\frac{\partial^k u_\theta}{\partial x^k}$ evaluated at a set of specified coordinates normally referred to as collocation points. Commonly used activation functions-such as $\tanh$ or $\sin$ are chosen to be sufficiently smooth [3]. We can think of choosing these collocation points similarly to the h parameter in FDM: the coarser the grid, or the larger the time steps we take as we compute the solution, the less accurate the solution will be. However, also choosing too many collocation points could lead to computational and instability issues as there will be too many points the network will have to validate results against, as shown in Figure 2.1, found in [1].

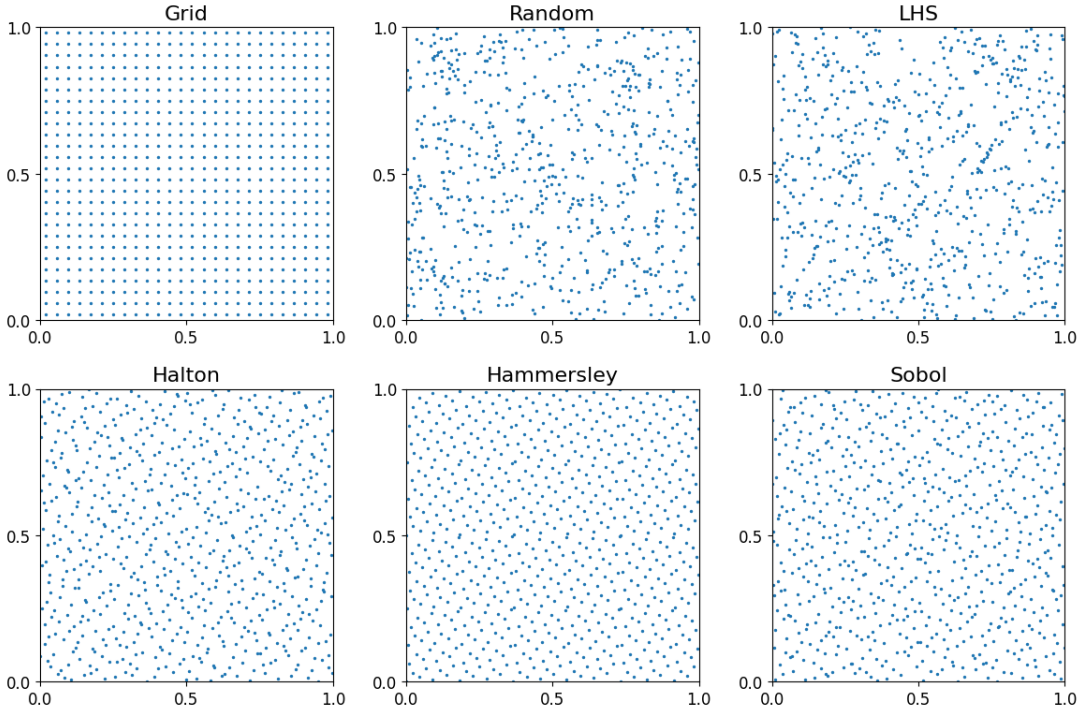
As there is no restriction in the choice of collocation points, they can be randomly sampled from inside the function domain, e.g. using an equispaced uniform grid, uniform random sampling, or a Latin hypercube sampling (LHS) [30]. This gives PINNs a great advantage over classical PDE solvers, such as finite differences, that rely on a cumbersome construction of a computational mesh. Common approaches in PINNs are a fixed set of collocation points, a newly sampled set at each iteration or adaptively chosen points based on the residuals of the PDE [3]. Figure 2.2 shows an overview of the main uniform methods that exist to select the distribution of collocation points that are used during training.

As a common approach in supervised machine learning, the neural network function can be updated with labeled data [3] and we use this to construct a data-based loss function. In a PINN, the shared parameters of the neural networks $h_\theta(t, x)$ and $f_\theta(t, x)$ can be learned by minimizing the mean squared error (MSE) loss [7], typically divided into three parts:

1. Physics (or residual) loss

$$L_r = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \mathcal{F}\left(x_{int}^{(i)},\, u_\theta(x_{int}^{(i)}), \dots\right) \right|^2 \tag{2.27}$$

Figure 2.2 Seven hundred collocation points in $[0, 1]^2$, sampled via different uniform strategies.



Evaluated at the set of *interior* collocation points $x_{int}^{(i)}$. This term forces the network to satisfy the PDE itself at the collocation points. If this part of the loss is not close to 0, it means that the physics are being violated in the system.

2. Data (or supervised) loss

$$L_d = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u_\theta(x_u^{(i)}) - u_{\text{obs}}^{(i)} \right|^2 \tag{2.28}$$

Evaluated at the locations $x_u^{(i)}$ where we have measurements $u_{obs}^{(i)}$, in the cases where the solution is known. In the cases where the solution is unknown, this term of the loss cannot be computed, and we would have to rely solely on the other two terms of the loss.

3. Boundary (and/or initial) loss

$$L_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \left| \mathcal{B}\big(x_b^{(i)},\, u_\theta(x_b^{(i)}), \dots\big) \right|^2 \tag{2.29}$$

Evaluated at boundary points $x_b^{(i)}$. Here $\mathcal{B}$ enforces Dirichlet ($u = \alpha$), Neumann ($\partial_n u = 0$), or other boundary conditions [7].

The data loss term is only defined where we have true function values inside the domain. The boundary term is defined only on the domain's boundary (and/or at $t = 0$ for initial conditions), where we impose known physical constraints at specific points, and not observations [3]. Therefore, the final physics-informed model can be trained by minimizing the following composite loss function

$$L(\theta) \;=\; w_r\, L_r \;+\; w_d\, L_d \;+\; w_b\, L_b. \tag{2.30}$$

Separating the loss terms provides flexibility to balance the weight terms w: how strictly we enforce the PDE versus the data versus the boundary conditions during training.

The L_r term creates a penalty when the neural network is violating known physics. This works very well for systems where we know something about the physics, for example, when we know we are working with a fluid flow, a quantum system, or an electromagnetic system, while still trying to predict our field variables as a function of space and time. So this is a very powerful yet simple idea. If you know the physics of the system, for example, if you know that you are working with a Navier-Stokes fluid flow, even if you only have limited or no observational data, you can still check if the network is "physical", meaning, "Does it satisfy the physics of my system?".

In this *small data regime*, the vast majority of state-of-the art machine learning techniques (e.g., deep/convolutional/recurrent neural networks) are lacking robustness and fail to provide any guarantees of convergence. This is what makes PINNs powerful techniques to solve differential equations: they can excel with limited amounts of data [7].

To enforce governing laws, it is necessary to compute the spatial and temporal derivatives of the approximated solution to construct and penalize PDE residuals. In the original formulation [7], these derivatives are calculated exactly using automatic differentiation (AD). AD takes advantage of the fact that all numerical computations are ultimately compositions of a finite set of elementary operations, for which derivatives are known [31, 32]. However, AD significantly increases computational cost due to the need to calculate and multiply gradients at each layer, which can become inaccurate for higher-order derivatives [33] and infeasible for fractional operators or high-dimensional problems [5]. To address these challenges, several studies have explored alternatives to or enhancements of backpropagation [2].

Some alternative differentiation methods include using the finite differences method to approximate the derivatives, which can greatly decrease computational costs [34]. However, this method relies on a predefined grid, limiting its broader applicability.

2.4 Applications of PINNs

PINNs have been widely applied to different industry problems. To name a few:

1. **Materials Science**: This paper [35] introduces PINNs for three challenging heat transfer problems: 1) recovering full velocity and temperature fields in forced and mixed

convection with unknown thermal boundaries; 2) solving inverse two-phase Stefan problems to infer moving interfaces and material conductivities; and 3) accelerating the solving of heat transfer problems on electronic chips without traditional meshing or discretization. Additionally, the authors show how embedding the governing PDEs via automatic differentiation into a multitask learning framework allows one to infer temperature and flow fields even with unknown boundary conditions and noisy data. Imagine a world where engineers can design composite structures with desired mechanical properties simply by inputting physical constraints into a neural network. The ability to optimize material composition opens new doors in industries such as aerospace engineering, where lightweight yet durable materials are crucial for aircraft design [10].

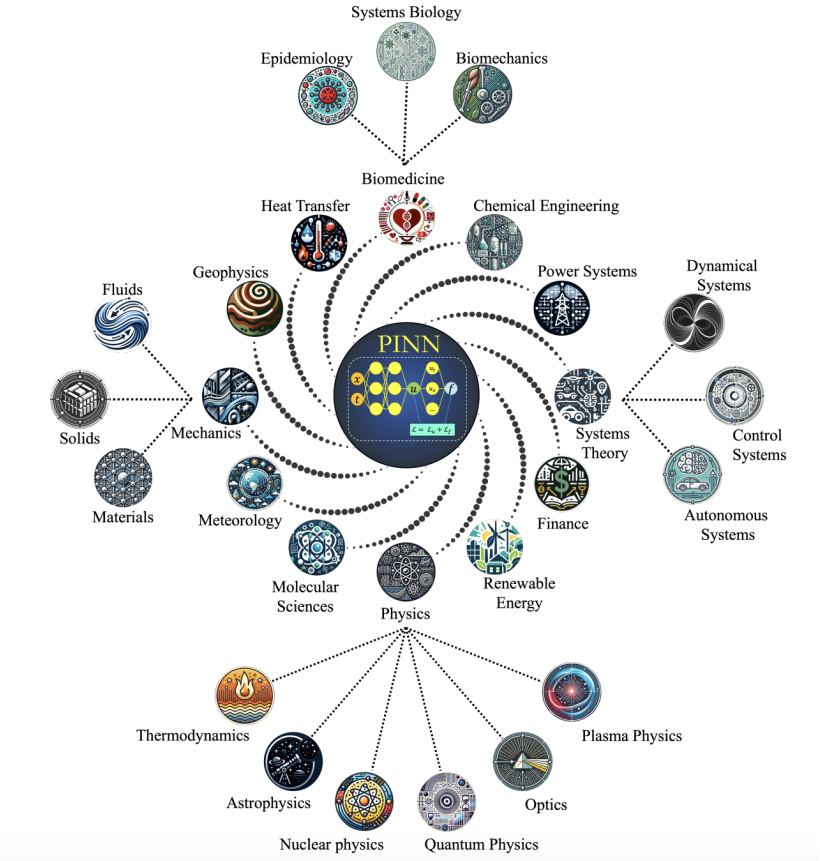
2. **Power Systems**: This 2023 review [36] presents a comprehensive survey of how PINNs can be tailored to power networks by embedding the differential algebraic equations of the grid directly into the neural network training. This reduces the need for large datasets while ensuring physically consistent predictions. It covers key applications in modern power systems: state and parameter estimation, dynamic stability analysis, power flow and optimal power flow (OPF) calculations, as well as anomaly detection and localization. This shows that PINNs can improve modern power grids by enabling real-time monitoring and optimization with minimal data, boosting system reliability and efficiency.

3. **Fluid Mechanics**: Another body of work that is being impacted by this research is fluid mechanics. This work [37] delves into how a PINN can empower engineers to bypass costly mesh generation and inversion codes by embedding the Navier-Stokes equations into a neural network, enabling rapid reconstruction of complex flows (e.g., 3D wakes, supersonic jets, biomedical flows) from sparse or noisy data and dramatically streamlining experimental-to-simulation workflows.

4. **Additive Manufacturing**: PINNs have also been researched in quality prediction for additive manufacturing (AM), focusing on temperature field prediction, fatigue life prediction, accelerated finite element simulation, and process characteristic forecasting [38]. The paper discusses the pros and cons of traditional data-driven methods and pure physics models and further elaborates on the principles and architecture of the PINN model along with its applications in AM research.

5. **Biomedicine**: PINNs are being used to diagnose and treat atrial fibrillation (rapid irregular beating of parts of the heart). They have been used to create detailed activation maps for diagnosis and estimate cardiac fiber orientation from electroanatomical data, both of which are essential for personalized treatment and procedural planning [13, 14]. Also, in the area of pharmacology, PINNs are being used to predict the assimilation of drugs in the human body, even getting to model glucose-insulin interactions, offering a guide for researchers tackling gray-box identification challenges in complex dynamical systems in biomedicine [39].

Figure 2.3 Applications of PINNs. An illustration of the diverse applications where PINNs have been employed by incorporating the underlying physical laws (found in [2]).



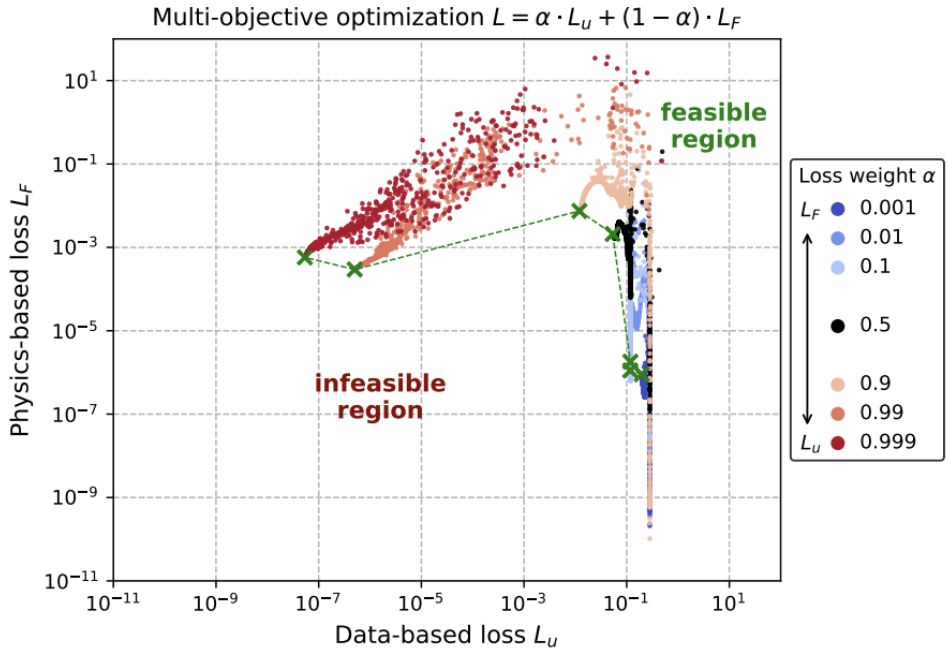
These are just some of the vast areas of applications that are currently taking place. Nonetheless, there are many others, as shown in Figure 2.3 (found in [2]).

Some variants that have recently been explored include variational PINNs (VPINNs), which use a weak form to improve stability and handle higher-order PDEs more naturally [40], and adaptive sampling strategies that concentrate collocation points in regions of high residual error [41]. These works have been published in 2021 and 2025, respectively, so this is a very active area of research.

There has also been work done with PINNs tailored for fractional derivatives (fPINNs), which embed discretized fractional derivatives (via Grünwald-Letnikov approximations) into the training loss function [42]. fPINNs proved to achieve high accuracy and robustness to noise, demonstrating the method's ability to capture nonlocal dynamics and broadening the applicability of PINNs to complex fractional systems. It was particularly observed that this method outperforms FDM when forcing black-box conditions.

PINNs have also been studied on a multiobjective optimization framework, where the

Figure 2.4 Pareto front. How different samples of parameters can shift the Pareto front between the data loss and the physics loss of the problem (found in [3]).



balance in the loss function is explored between the data loss and the physics-informed loss. They investigate this delicate weighting by casting the combined objective as a multi-objective optimization problem and examining its apparent Pareto front, as shown in Figure 2.4 (found in [3]).

They show that physical system parameters such as length and time scales, domain size, and PDE coefficients, act as implicit loss scalings that skew the Pareto front, making some loss-weight choices yield biased or invalid solutions. They explain some common convergence failures and prescribe weight-tuning strategies to rebalance the objectives. One failure mode was described when solving the one-dimensional diffusion equation, where the data and physics-based losses were equally weighted in the network training which led to a large discrepancy between true and predicted system dynamics. This demonstrates that selecting weights on the front's locally convex "bulge" yields solutions closest to the true physical state and confirm that reparametrizing the system can widen the convex region, thereby increasing robustness to weight selection and improving PINN training success across many regimes [3].

Something interesting that is also being actively researched in PINNs is domain decomposition. Finite Basis PINNs (FBPINNs) [43] propose a divide-and-conquer strategy where multiple small neural networks are trained to solve an overlapping subdomain of the PDE and then combine the solutions with input normalization, allowing the effective learning of really complex PDEs. Through this parallel divide-and-conquer training algorithm, FBPINNs

reduce the complexity and nonconvexity of the PINN optimization problem and, in numerical experiments on canonical PDEs such as the Burgers and wave equations, outperform standard PINNs in both accuracy and computational efficiency.

Very recent work has explored multifidelity domain decomposition to address PINNs' spectral bias and multiscale challenges. The term fidelity here refers to the number of collocation points that are used for training: if you train with less points, the fidelity of the resulting PINN would be lower. Heinlein et al. propose stacking a low-fidelity PINN with time-domain-decomposed, high-fidelity finite-basis PINNs (FBPINNs), demonstrating superior accuracy on pendulum dynamics, a two-frequency test, and the Allen-Cahn equation [44]. Building on multigrid ideas, Tsai et al. (2025) introduce a multi-level dataset strategy where PINNs are trained in "V-cycle fashion" on increasingly larger collocation sets so that low collocation point levels eliminate low-frequency errors and larger collocation point levels tackle high-frequency components, yielding 30-60% accuracy improvements on high-frequency ODEs [45]. Applied to PINNs, a V-cycle fashion means you first train a network on fewer collocation points, then transfer its learned parameters to initialize training on a finer set (with more collocation points). This is a very active area of research exploring many ideas in order to solve more complex PDE domains.

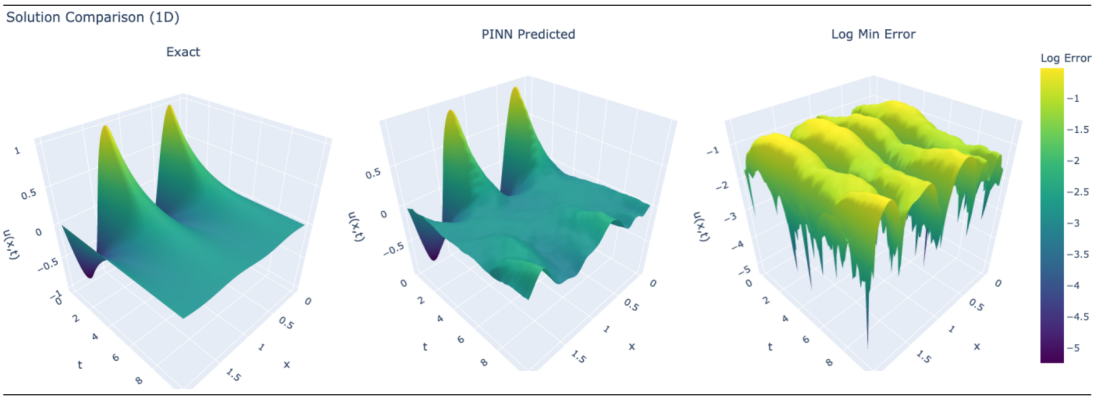
2.5 Comparative Analysis

Classical methods (FDM, spectral) offer rigorous error bounds and efficiency on structured meshes but can struggle with high-dimensional or data-driven contexts as you need to discretize your solution (which requires mesh construction). Early neural PDE solvers introduced PIML but lacked robustness. PINNs bridge these gaps by embedding physics directly into a learning framework, though they often require careful loss balancing and can suffer from spectral bias in network training. Recent reviews highlight open challenges in scalability, convergence guarantees, and integration with large-scale simulation codes [6].

One of the main advantages of Physics-Informed Machine Learning (PIML) methods is their ability to handle high-dimensional problems [32, 46]. They are mesh-free, which is a great advantage since FDM require a delicate construction of a mesh in order to converge successfully. PINNs have been applied to forward and inverse problems, demonstrating flexibility and the ability to handle noisy observations [8]. Also, they are (mostly) unsupervised methods that work well in mixed data regimes where there is only some noisy data, and the goal is to carry out a simulation or an inversion of a problem.

Nevertheless, many challenges come about implementing a PINN to go about solving a PDE. It was found in this study [47] that the convergence properties of PINNs are still not very well understood. The existence of null derivatives in the problem domain can lead to inconsistent solutions where the boundary and/or initial conditions are not met, which highlights the importance of understanding these properties in order to select the PINN architecture and parameters more effectively. An important balance that comes into play here is being able to select the weight parameters in the loss function effective, as giving too much weight to one of the terms and too little to another could lead to poor convergence.

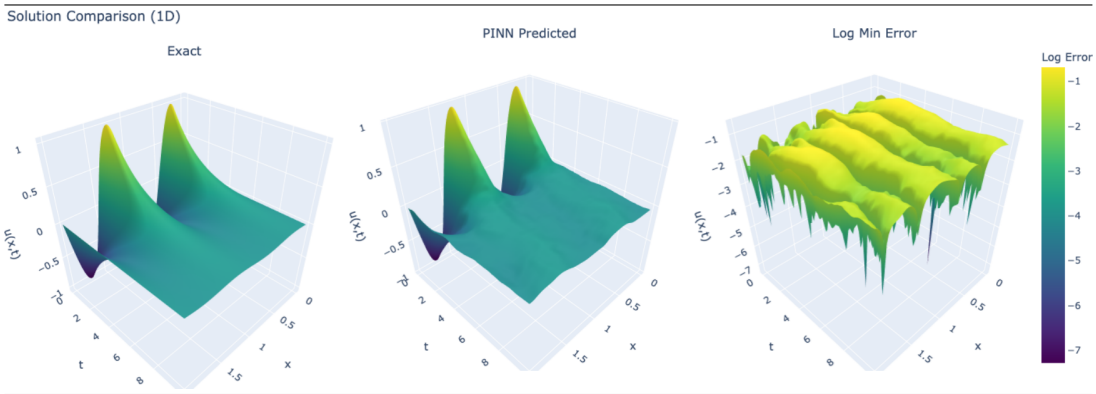
Figure 2.5 Inconsistent PINN compared to the exact solution $u(x,t)$ of the PDE for the heat propagation case. To the left, the exact analytical solution. At the center, The PINN-predicted solution, which fails to correctly adhere to the boundary conditions. To the right, the point-wise error, defined as the logarithm of the minimum between the absolute and relative error: $E = \log_{10}(\min(|u_{exact} - u_{pred}|, \frac{|u_{exact} - u_{pred}|}{|u_{exact}|}))$.



In particular, the vanilla implementation of PINNs is known to be prone to training failures that often lead to inaccurate and nonphysical predictions [48]. Figure 2.5 shows an example where a PINN fails to comply with the boundary conditions of the 1D heat equation problem effectively, due to improper selection of training parameters. The error metric, shown on a log scale, is the point-wise minimum of the absolute and relative errors: $E = \log_{10}(\min(|u_{exact} - u_{pred}|, \frac{|u_{exact} - u_{pred}|}{|u_{exact}|}))$. The PINN was trained by minimizing a composite loss function: $L(\theta) = L_{\text{PDE}} + L_{\text{IC}} + L_{\text{BC}}$. A visual comparison of the exact solution and the PINN-predicted solution shows that the prediction incorrectly deviates from zero at the boundaries. The log min error plot further quantifies this, confirming that the largest errors are concentrated along these boundaries and towards the last time steps of the solution.

In this case, contrary to intuition, initializing the training with too many collocation points (10,000) makes the optimization problem unstable in its later stages. This creates this higher-frequency oscillation behavior in the final solution shown in Figure 2.5, which fails to adhere with the desired behavior shown in the exact solution (to the left). Since at each iteration the network must resample the same number of collocation points to learn the physical dynamics of the PDE, having too many makes the convergence of the solution slower and increases the volatility of the generated surface. This problem could be made smaller by increasing the number of iterations for training, but an improvement is not necessarily guaranteed. Therefore, a prudent number of collocation points is advised, especially when the complexity of the PDE is low. As the complexity of the PDE increases, more collocation points are usually a good idea, but one cannot go overboard with this since this reduces the speed of convergence by increasing computational cost per step. A gradual increase in the selection of the number of collocation points tends to be enough to achieve successful convergence under the PDE at hand, keeping in mind that more collocation points does not always mean better

Figure 2.6 Improved PINN convergence by decreasing the number of collocation points from 10,000 to 8,000.



learning to be achieved. For example, by simply reducing the number of collocation points from 10,000 to 8,000, the oscillatory behavior is largely reduced, as shown in Figure 2.6.

There is a delicate balance to keep in mind here, that must be also weighed in with the complexity of the PDE. For example, for the 1D heat equation case, the boundary conditions are basically trivial to learn. But as more complex PDEs are introduced, the unique weights of the loss terms will also play a role in defining successful convergence. This can introduce significant variance into the gradient estimates, making it difficult for the optimizer to find a stable descent path.

The discussion on training failures in PINNs is diverse, and each problem setup seems to present its own unique challenges for PINN optimization [49]. This diversity makes it difficult to choose the right remedy in the face of certain optimization issues. In general, any improvement in the robustness and generalizability of PINNs requires an understanding of the optimization complexity of the physics loss function [3]. As a rule, its complexity increases as the physical system and governing differential equations become more complex. To cope with complex systems and geometries, domain decomposition or sequence-to-sequence methods have been developed that divide the original problem into smaller subdomains [49]. Each subdomain is then tackled by a separate PINN, resulting in an overall lower optimization complexity and error. It was also shown in [49] that adding soft regularization can actually make the problem harder to optimize, as the regularization leads to less smooth loss landscapes. Regularization is set of techniques used to reduce overfitting tendencies [50], and under a PINN setup it provides a simple starting point for the PINN optimization, which gradually becomes more complex as the PINN is trained. Nevertheless, it was shown in [50] that even when adding regularization to the training of the PINN we could end up without improvements in the results. This emphasizes the challenges that come about scaling PINNs to high-frequency and complex domains. Even with simple systems, the optimization may converge to suboptimal solutions that describe trivial solutions to the physics loss function [51]. In this regard, learning in sinusoidal space [52], methods that respect causality [53], or choosing the right neural network architecture for the problem [49] provide a potential

remedy.

This chapter reviewed both classical numerical methods and recent machine learning approaches to solving PDEs, with a particular focus on Physics-Informed Neural Networks (PINNs). It explored their theoretical formulation, advantages, and limitations compared to traditional solvers. Building on this foundation, Chapter 3 presents the specific PDEs addressed in this thesis: the Heat, Allen-Cahn, and Korteweg–de Vries (KdV) equations. Their mathematical formulations, boundary and initial conditions, and reference numerical solutions are introduced in preparation for implementing and evaluating PINN-based solvers.

Chapter 3

PDE Formulation

In this chapter, the precise mathematical formulations of the physics problems used for the experiments are presented. The various classes of PDEs under study, together with their initial and boundary conditions and the specific parameter values used, are introduced. In addition, the numerical methods employed to generate the reference solutions for these PDEs are described.

As a summary, these are the main properties of the three PDEs studied in this work.

Equation	Application Domains	Complexity
Heat Equation	Heat conduction, diffusion processes [54]	Linear, 2nd-order
Allen-Cahn Equation	Phase transitions, material science, pattern formation [55, 56]	Nonlinear, 2nd-order
Korteweg-de Vries (KdV) Equation	Water waves [57], plasma physics [58], nonlinear optics [59]	Nonlinear, 3rd-order

Table 3.1: Summary of PDEs studied in this work.

3.1 Heat Equation

The heat equation describes the diffusion of thermal energy in a medium, modeling how temperature gradients dissipate over time according to Fourier's law [54]. It is a linear, second-order parabolic PDE, admitting analytic solutions for many canonical cases and serves as a benchmark for numerical and machine-learning solvers.

To present the precise mathematical formulation of the heat equation problems used in our experiments, we cover both the one-dimensional (1D) and two-dimensional (2D) domains.

3.1.1 One-Dimensional Case

The one-dimensional heat equation models the diffusion of thermal energy along a thin rod of length L with no internal heat sources. Over time, temperature gradients smooth out due to conduction.

We solve

$$\frac{\partial u}{\partial t}(x,t) = \alpha \frac{\partial^2 u}{\partial x^2}(x,t), \quad x \in [0,L], t \in [0,T], \tag{3.1}$$

where $u(x,t)$ is the temperature, $\alpha > 0$ the thermal diffusivity, L the rod length, and T the final time.

The initial condition will have a single sinusoidal mode:

$$u(x,0) = u_0(x) = \sin\Big(\frac{2\pi k x}{L}\Big), \quad k \in \mathbb{Z}^+. \tag{3.2}$$

The domain length is set to $L = 2.0$ (defining the size of the spatial interval), the thermal diffusivity to $\alpha = 0.01$ (governing the rate of heat diffusion), and the wave number to $k = 2$ (two complete sinusoidal oscillations over $[0, L]$). Here the k term refers to the number of periods (frequency, or spatial wave number). This choice admits the exact solution below, which was used to benchmark both classical solvers and PINNs.

Periodic boundary conditions enforce continuity at the rod ends:

$$u(0,t) = u(L,t), \quad \frac{\partial u}{\partial x}(0,t) = \frac{\partial u}{\partial x}(L,t), \quad t \in [0,T]. \tag{3.3}$$

For the chosen initial sinusoidal mode, the exact solution is

$$u(x,t) = \exp\big(-\alpha(2\pi k/L)^2 t\big) \sin\Big(\frac{2\pi k x}{L}\Big), \tag{3.4}$$

which decays exponentially in time. This decay factor $e^{-\alpha(2\pi k/L)^2 t}$ represents the exponential damping of the sinusoidal mode's amplitude due to thermal diffusion.

3.1.2 Two-Dimensional Case

Extending diffusion to a square plate, the 2D heat equation captures interactions of temperature gradients in both directions [11], still with an analytic separable solution that decays exponentially.

On the square domain $[0, L]^2$, we solve

$$\frac{\partial u}{\partial t}(x,y,t) = \alpha\Big(\frac{\partial^2 u}{\partial x^2}(x,y,t) + \frac{\partial^2 u}{\partial y^2}(x,y,t)\Big), \quad (x,y) \in [0,L]^2, t \in [0,T]. \tag{3.5}$$

We will have a separable sinusoidal mode for the initial condition

$$u(x,y,0) = u_0(x,y) = \sin\Big(\frac{2\pi k x}{L}\Big) \sin\Big(\frac{2\pi k y}{L}\Big), \quad k \in \mathbb{Z}^+, \tag{3.6}$$

as well as periodic boundary conditions in both directions

$$\begin{aligned} u(0,y,t) &= u(L,y,t), \quad t \in [0,T], \\ \partial_x u(0,y,t) &= \partial_x u(L,y,t), \quad t \in [0,T], \\ u(x,0,t) &= u(x,L,t), \quad t \in [0,T], \\ \partial_y u(x,0,t) &= \partial_y u(x,L,t), \quad t \in [0,T]. \end{aligned} \tag{3.7}$$

The analytic solution for the separable mode is

$$u(x,y,t) = \exp\bigl(-2\alpha(2\pi k/L)^2 t\bigr)\sin\Bigl(\frac{2\pi k x}{L}\Bigr)\sin\Bigl(\frac{2\pi k y}{L}\Bigr). \tag{3.8}$$

Here, the decay factor is expressed the same as in the one-dimensional case, expressing the exponential decay of the temperature, only here we have -2α and the second variable y.

3.2 Allen-Cahn Equation

The Allen-Cahn equation describes phase separation and interface motion in binary alloys, capturing the dynamics of domain coarsening via a cubic reaction term combined with diffusion. It is a nonlinear, second-order parabolic PDE commonly used in material science and pattern-formation studies [60, 55].

On the periodic domain $x \in [-1,1]$ and time interval $t \in [0,T]$, the equation reads

$$\frac{\partial u}{\partial t}(x,t) = \varepsilon \frac{\partial^2 u}{\partial x^2}(x,t) + u(x,t) - u(x,t)^3, \quad x \in [-1,1],\ t \in [0,T], \tag{3.9}$$

where $\varepsilon > 0$ is the interface thickness parameter.

The experiments use

$$\varepsilon = 0.01, \quad T = 1.0, \quad x \in [-1,1],$$

for both the spectral numerical solver and PINN settings.

As for the initial condition, a hyperbolic-tangent profile initializes a diffuse interface centered at $x = 0$:

$$u(x,0) = \tanh\left(\frac{x}{\sqrt{2\varepsilon}}\right), \quad x \in [-1,1]. \tag{3.10}$$

This solves $\varepsilon u_{xx} + u - u^3 = 0$ at equilibrium.

Periodic boundary conditions enforce:

$$u(-1,t) = u(1,t), \quad \frac{\partial u}{\partial x}(-1,t) = \frac{\partial u}{\partial x}(1,t), \quad t \in [0,T], \tag{3.11}$$

ensuring that the solution and its spatial derivative match at the domain endpoints, wrapping the interval $[-1, 1]$ into a continuous loop with no artificial boundary layers.

We test the accuracy of the Allen-Cahn PDE's learned solution via a semi-implicit Euler discretization in time [16] combined with spectral differentiation in space for the diffusion term. On the periodic grid $x_j = -1 + j\Delta x$ with $\Delta x = 2/N_x$ and $\varepsilon = 0.01$ the update of each Fourier mode is

$$\widehat{u}_k^{n+1} = \frac{\widehat{u}_k^n + \Delta t \left[\widehat{u^n}_k - \widehat{(u^n)^3}_k\right]}{1 + \Delta t\, \varepsilon k^2}, \qquad k = -\tfrac{N_x}{2}, \ldots, \tfrac{N_x}{2} - 1.$$

With $N_x = 256$ and $\Delta t = 5 \times 10^{-4}$ we store 80 evenly spaced snapshots between $t = 0$ and $t = 1$. A routine was developed using NumPy's Fast Fourier Transform (FFT) and Inverse FFT functions to compute the derivatives in spectral space and then back to the physical space. The scheme is spectrally accurate in space and first–order in time, providing the high-fidelity baseline for the PINN comparison. Code reference solutions can be found in the Allen-Cahn associated script (see Appendix 6.3).

3.3 Korteweg-de Vries (KdV) Equation

The Korteweg-de Vries (KdV) equation describes the unidirectional propagation of long, weakly nonlinear dispersive waves, such as shallow-water solitons, by balancing quadratic nonlinearity against third-order dispersion [57, 58]. It is a nonlinear, third-order PDE with rich soliton dynamics. Moreover, it is completely integrable via the inverse scattering transform, yielding an infinite hierarchy of conserved quantities that underpin its rich nonlinear evolution [61].

The KdV equation owes its name to the work of Korteweg and De Vries in 1895 [57], who demonstrated that small-amplitude long waves on the free surface of shallow water satisfy a third-order dispersive PDE. The standard form [61] of the PDE is given by

$$u_t + 6\, u\, u_x + u_{xxx} = 0. \tag{3.12}$$

Although now universally known as the KdV equation, an equivalent model was in fact derived earlier by Boussinesq in 1877. Subsequent developments, including inverse-scattering techniques, have revealed its rich soliton dynamics and integrability properties [62].

On the periodic spatial domain $x \in [0, 2\pi]$ and time interval $t \in [0, T]$, the KdV equation reads

$$\frac{\partial u}{\partial t}(x,t) + 6\, u(x,t)\, \frac{\partial u}{\partial x}(x,t) + \frac{\partial^3 u}{\partial x^3}(x,t) = 0, \quad x \in [0, 2\pi],\ t \in [0, T]. \tag{3.13}$$

The experiments use

$$L = 2\pi, \quad T = 0.6, \quad N_x = 256,$$

matching the pseudo-spectral baseline solver and PINN configuration. Here, $L = 2\pi$ is the length of the periodic spatial domain, $T = 0.6$ is the final time up to which the solution is computed, and $N_x = 256$ is the number of spatial grid points used in the pseudo-spectral discretization, which is then solved via the backward-differentiation formula (BDF) [20] for the baseline case.

For the initial condition, a single-mode wave is represented by

$$u(x, 0) = \sin(x), \quad x \in [0, 2\pi], \tag{3.14}$$

which initiates a fundamental Fourier mode and permits comparison against the ODE-based baseline solver.

Periodic boundary conditions enforce

$$u(0, t) = u(2\pi, t), \quad \frac{\partial^k u}{\partial x^k}(0, t) = \frac{\partial^k u}{\partial x^k}(2\pi, t), \quad k = 1, 2, \ t \in [0, T], \tag{3.15}$$

wrapping the domain into a circle with no artificial boundaries.

The reference solution is obtained by combining a pseudo-spectral spatial discretization (via FFT differentiation) with a multi-step backward-differentiation formula (BDF) that acts as a time integrator [16]. First, we turn the Korteweg–de Vries (KdV) into an initial-value problem

$$u_t + 6u\, u_x + u_{xxx} = 0, \qquad u(x, 0) = \sin x, \quad x \in [0, 2\pi),$$

using a pseudo-spectral method in space and a second-order backward differentiation formula (BDF2) in time. This is done with the help of the Numpy library in Python. The spatial discretisation uses a uniform grid (with $N_x = 256$), and the derivatives are evaluated in spectral space, $u_x = \mathcal{F}^{-1}[(ik)\widehat{u}]$ and $u_{xxx} = \mathcal{F}^{-1}[(ik)^3\widehat{u}]$.

In Fourier space, ∂_{xxx} is simply the multiplication by $(ik)^3$, so inverting spectrally, $1 - \frac{2}{3}\Delta t\, \partial_{xxx}$, reduces to a division for each wavenumber. Specifically, the time integrator (BDF2) solves

$$\frac{3u^{n+1} - 4u^n + u^{n-1}}{2\Delta t} = -6u^{n+1}u_x^{n+1} - u_{xxx}^{n+1},$$

at each time step with $\Delta t = 5 \times 10^{-4}$, allowing us to move between physical and spectral space, just as was done with the Allen-Cahn equation. The time stepping is done using Scipy's `solve_ivp` function, which solves a given initial value problem with a selected method. Various solver methods were experimented with (that came with the open source package), such as RK45 (explicit Runge-Kutta method of order 5(4)), RK23 (explicit Runge-Kutta method of order 3(2)), and the BDF method. This BDF method was found to give the best results for the KdV equation, which is described as an implicit multi-step (second-order in this case, but can be higher) method based on a backward differentiation formula (BDF) for the derivative approximation [63]. Along with this derivative approximation method, we used a relative

tolerance of 10^{-6} and an absolute tolerance of 10^{-8}. The pseudo-spectral method yields machine-precision spatial derivatives on the periodic interval, while BDF provides A-stable [20], high-order stepping for the resulting stiff ODE system. Code reference solutions can be found in the KdV associated script (see Appendix 6.3).

This chapter introduced the three partial differential equations studied in this work: the Heat, Allen-Cahn, and Korteweg–de Vries (KdV) equations, detailing their physical interpretations, mathematical formulations, and standard numerical solvers used for validation. With this context established, we will now focus on the design and training of Physics-Informed Neural Networks (PINNs) tailored to each equation in Chapter 4. This includes a detailed explanation of the network architecture, loss formulation, sampling strategy, training procedure, and the reasoning behind key hyperparameter choices.

Chapter 4

PINN Methodology

In this chapter, we detail the physics-informed neural network (PINN) approach used to solve three PDEs of increasing complexity: the one-dimensional Heat equation (a linear second-order parabolic PDE), the Allen-Cahn equation (a nonlinear reaction-diffusion PDE with second-order spatial derivative), and the Korteweg-De Vries (KdV) equation (a nonlinear dispersive wave PDE with third-order spatial derivative). We describe the network architecture employed (fully connected multilayer perceptrons with Fourier feature encodings), the formulation of the PINN loss function (enforcing PDE residuals alongside initial and boundary conditions), the strategy for sampling training points, and the training procedure (optimizer, learning rate, and autodifferentiation aspects). Throughout, we highlight how the PINN methodology is tailored to each equation's characteristics.

4.1 Network Architecture

As we delve into the world of physics-informed neural networks (PINNs), it is essential to understand the underlying principles that govern conventional neural networks. The core of each PINN is a fully connected feedforward neural network (multilayer perceptron, MLP [64]) that takes the space and time coordinates as input (here x and t) and outputs the surrogate solution $u_\theta(x,t)$. Here θ means the network's trainable parameters (weights and biases). We employ a common architecture for all three PDEs, consisting of several hidden layers of fixed width with smooth activation functions. This choice is motivated by prior PINN studies which found that relatively modest MLPs (on the order of 4–6 hidden layers and 50–100 neurons per layer) are often sufficient for a range of PDE problems [65]. In our implementation, each hidden layer uses the hyperbolic tangent ($\tanh$) activation function, a continuously differentiable nonlinearity that promotes a smooth network output suitable for computing higher-order derivatives. Non-differentiable activations like the rectified linear unit (ReLU, $\max\{x,0\}$) are avoided since in our case PDE residuals involve second or third order derivatives. This architectural setup, an $[x,t] \to u$ mapping with 4–5 $\tanh$-activated hidden layers of moderate width (128, 512), is consistent with standard PINN practice [66], providing a universal function approximator for the solution $u(x,t)$.

Figure 4.1 (found in [4]) summarizes the architecture of a PINN, illustrating how physical laws are embedded into the training process through the loss function design. A fully connected neural network u_θ takes space-time coordinates as inputs and outputs an approximation of the solution. The training is driven not only by data from initial and boundary conditions, but also by enforcing the PDE residual at collocation points within the domain Ω. As shown, the total loss is a weighted linear combination of three terms: the PDE residual loss $\hat{J}_{\text{PDE}}$, the boundary condition loss $\hat{J}_{\text{BC}}$, and (optionally) an experimental data loss $\hat{J}_{\text{exp}}$, when observations are available. The network parameters θ are optimized to minimize this composite loss using automatic differentiation to compute the required derivatives for the PDE terms. This unified framework allows the network to learn a solution that is consistent with the governing equation and its constraints, even in the absence of full supervision (the data loss term is optional).

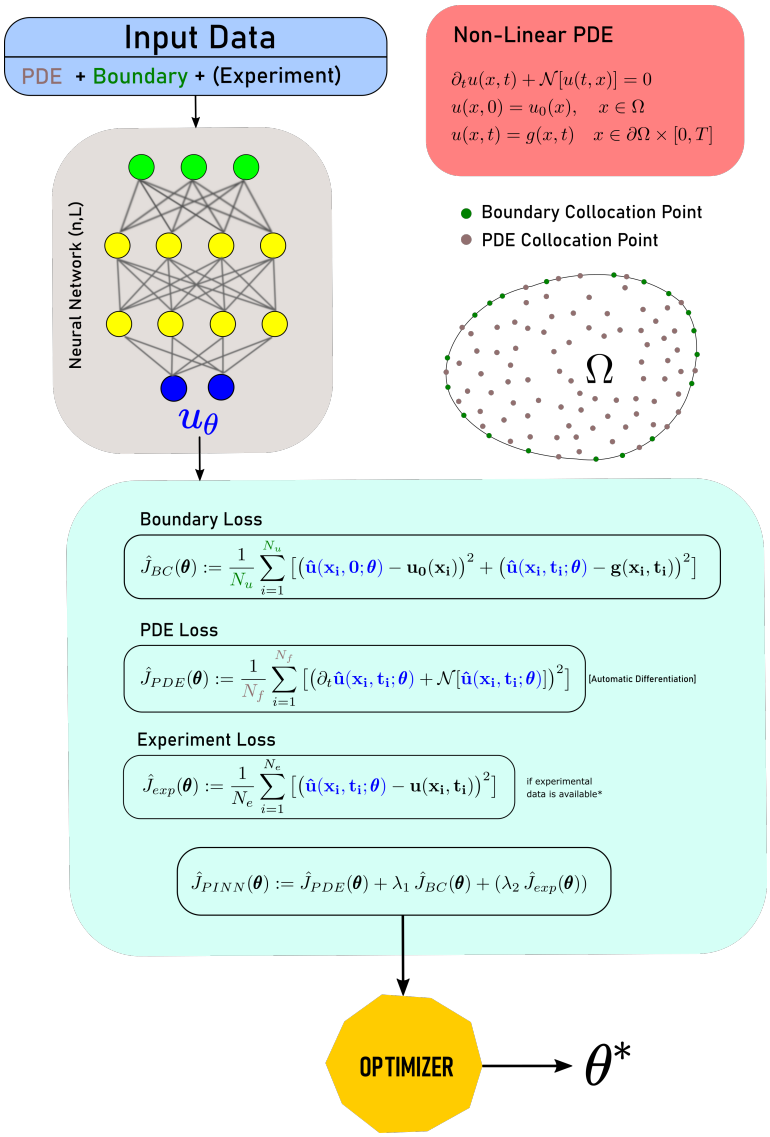
We augment the input with Fourier feature encodings to enhance the network's ability to represent complex solution behavior, especially for the higher-frequency content. A simple Fourier feature mapping enables a multilayer perceptron (MLP) to learn high-frequency functions in low dimensional problem domains, transforming the original inputs via sinusoidal functions of various frequencies [67]. We apply a randomized Fourier feature embedding $\gamma(x,t)$ defined as:

$$\gamma(x,t) = [\cos(2\pi \mathbf{B}[x,t]^T),\ \sin(2\pi \mathbf{B}[x,t]^T)]\ , \tag{4.1}$$

where $\mathbf{B}$ is a matrix of chosen frequencies sampled from a distribution in the domain of the problem. The concatenated feature vector $\gamma(x,t)$ is fed into the MLP in place of (x,t), allowing the network to more easily learn oscillatory components. This technique addresses the spectral bias of MLPs, wherein standard networks tend to struggle with high-frequency functions [67]. It was found that adding Fourier components to the KdV PINN made the training unstable, therefore it was not added in this PINN. The bandwidth and number of Fourier features were selected based on the expected solution complexity. For example, the Allen-Cahn solution requires a richer frequency spectrum, so we employ a larger embedding dimension of scale π, while the heat PINN uses a Fourier scale of 1.

While the overall network structure remains a fully connected MLP across all three cases, certain aspects are adapted to each of the PDE requirements. For the heat equation, the architecture is relatively shallow (2 hidden layers of 128 neurons) since the solution $u(x,t)$ is expected to be smooth and slowly varying. For the Allen–Cahn and KdV equations, which include nonlinear reaction terms and can produce sharper transitions, it was found beneficial to use a slightly deeper/wider network to capture the more complex patterns. For Allen–Cahn, the presence of the u^3 term means the network must accurately approximate both the function and its nonlinear transformation. The KdV equation poses the greatest challenge: it is third order in space and considered on a periodic domain. The MLP used for the KdV equation adopts Xavier initialization for all linear layers to improve training stability and ensure a balanced signal propagation through the network [68]. To ensure the network output is sufficiently smooth, $\tanh$ activation functions were used, enabling stable computation of up

Figure 4.1 General network architecture for a PINN (found in [4]).



to third-order derivatives via automatic differentiation. Aside from these adjustments, the PINN architecture does not fundamentally change between the three equations.

We summarize the forward pass computation of the PINN for each equation in pseudocode form. Algorithms 4.1, 4.2, and 4.3 illustrate the network evaluation and residual calculation for the Heat, Allen-Cahn, and KdV equations, respectively. Each algorithm outlines the initialization of the MLP, the incorporation of Fourier features, and the computation of the physics-informed residual corresponding to the given PDE. In all three cases the Adam optimizer was used for training [69], with $\eta = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ as recommended by the authors. Adam is widely used in PINN implementations for its robust

performance on non-convex loss functions and ability to handle noisy gradients from random sampling.

Input: $\alpha = 0.01$, $L = 2.0$, $T = 10.0$, $k = 2$
Initialize NN parameters θ (2 hidden layers, 128 units, `Tanh`), Fourier features B ($m = 100$)
for epoch = 1 **to** 5000 **do**
  Sample $N_{\text{PDE}} = 2000$ collocation points $(x, t) \in [0, L] \times [0, T]$
  Sample $N_{\text{IC}} = 200$ initial condition points, $N_{\text{BC}} = 200$ boundary times
  Compute total weighted loss:

$$\mathcal{L} = 10 \cdot \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{IC}} + 10 \cdot \mathcal{L}_{\text{BC}}$$

  Update θ using Adam optimizer
end for
return trained model $u_\theta(x, t)$

Algorithm 4.1: PINN for 1D Heat Equation with periodic BCs

Input: $\varepsilon = 0.01$, $x \in [-1, 1]$, $t \in [0, 1]$
Initialize NN parameters θ (2 hidden layers, 128 units, `Tanh`), Fourier features ($m = 100$, scaled by π)
for epoch = 1 **to** 5000 **do**
  Sample $N_{\text{PDE}} = 3000$ collocation points (x, t)
  Sample $N_{\text{IC}} = 500$ initial condition points, $N_{\text{BC}} = 500$ boundary times
  Compute total loss:

$$\mathcal{L} = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{IC}} + \mathcal{L}_{\text{BC}}$$

  Update θ using Adam optimizer
end for
return trained model $u_\theta(x, t)$

Algorithm 4.2: PINN for 1D Allen-Cahn Equation with periodic BCs

You can see how we are increasingly updating the number of collocation points that are used for training as the complexity of the PDE gets larger. Note how the PINN architecture itself does not hard code any specific equation. Rather, the differences across the three cases manifest in the computation of the residual $f(x, t)$ and in the enforcement of appropriate boundary and initial conditions. In summary, the network architecture for our PINNs comprises fully connected layers with smooth activations and Fourier feature inputs, with minor modifications (network depth or feature count, Fourier components, initialization step) to address the increasing complexity from the Heat equation to Allen-Cahn to KdV.

Input: $x \in [0, 2\pi]$, $t \in [0, 0.6]$
Initialize (with Xavier method) NN parameters θ (4 hidden layers, 512 units, Tanh)
for epoch = 1 **to** 9000 **do**
  Sample $N_{\text{PDE}} = 5000$ collocation points (x, t)
  Sample $N_{\text{IC}} = 500$ initial condition points, $N_{\text{BC}} = 500$ boundary times
  Compute weighted loss:

$$\mathcal{L} = 45 \cdot \mathcal{L}_{\text{PDE}} + 50 \cdot \mathcal{L}_{\text{IC}} + 40 \cdot \mathcal{L}_{\text{BC}}$$

  Update θ using Adam optimizer
end for
return trained model $u_\theta(x, t)$

Algorithm 4.3: PINN for KdV Equation with periodic BCs

4.2 Loss Function

For the case of both the Heat and Allen-Cahn equations, the PINN approximates $u(x, t) \approx u_\theta(x, t)$ via a Fourier-feature MLP, minimizing the composite loss

$$L(\theta) = L_{\text{PDE}} + L_{\text{IC}} + L_{\text{BC}}, \tag{4.2}$$

where for the case of the Allen-Cahn PDE, the PDE MSE term would look like

$$L_{\text{PDE}} = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| u_t - (\varepsilon\, u_{xx} + u - u^3) \right|^2,$$

with analogous MSE terms enforcing the initial condition and periodic boundary constraints. In our setup, we applied periodic boundary conditions for Allen–Cahn on a finite interval, as shown in [8], where they assumed $u(t, -1) = u(t, 1)$ and $u_x(t, -1) = u_x(t, 1)$). Thus L_{BC} for Allen–Cahn includes terms to enforce $u_\theta(-1, t) = u_\theta(1, t)$ and $u_{x,\theta}(-1, t) = u_{x,\theta}(1, t)$ for sampled times, ensuring the solution and its first derivative are continuous at the boundaries.

For the case of the heat equation, there are no nonlinear terms. Thus, the network primarily must learn a smooth diffusive behavior. The periodic boundary conditions dictate a repeating temperature at both ends, which are enforced via L_{BC} as described above. An additional weight was added to the PDE and boundary loss terms as described in Algorithm 4.1, to give a stronger emphasis on the minimization of these MSE terms.

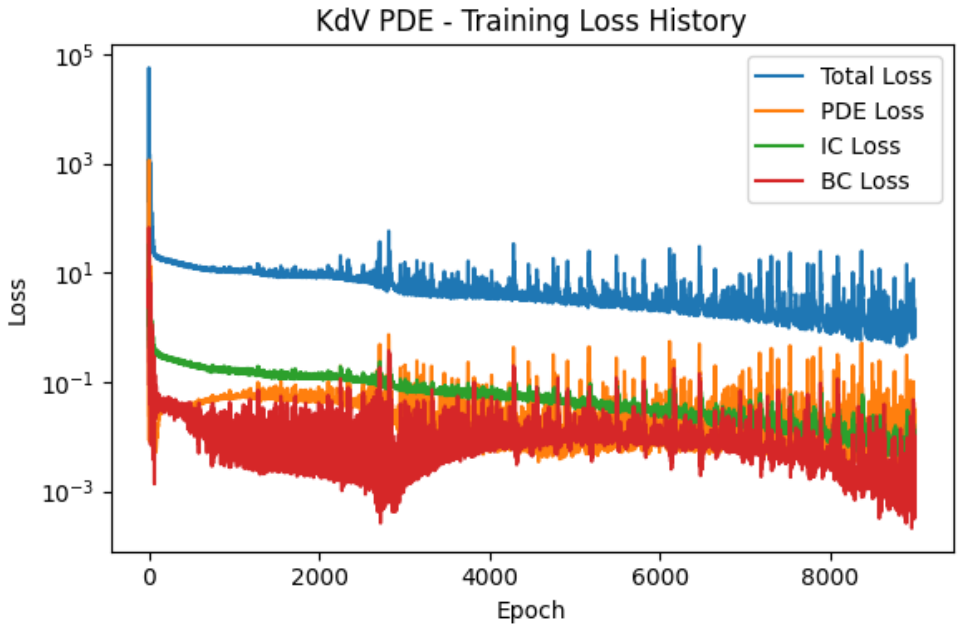
For the case of the KdV equation, the PINN approximates $u(x, t) \approx u_\theta(x, t)$ with a fully connected MLP. It minimizes the weighted composite loss function

$$L(\theta) = w_{pde}\, L_{pde} + w_{ic}\, L_{ic} + w_{bc}\, L_{bc}, \tag{4.3}$$

where

$$L_{pde} = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| u_t + 6u\, u_x + u_{xxx} \right|^2, \quad L_{ic} = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u_\theta(x_i, 0) - \sin(x_i) \right|^2,$$

Figure 4.2 KdV Equation: Training history components.



and

$$L_{bc} = \frac{1}{N_b}\sum_{i=1}^{N_b}\left|u_\theta(0,t_i) - u_\theta(2\pi,t_i)\right|^2.$$

These are the minimization terms that the network is attempting to bring to zero.

The KdV experiment served as a starting point to understand how increasing the weight of one loss term biases the total loss, while ensures that other terms in the loss composition remain low. In Figure 4.2 it is shown how having $w_{pde} = 45$, $w_{ic} = 50$, and $w_{bc} = 40$, really adds a consistent decay tendency to the initial condition loss (the green line). However, the sum of all three loss terms (the blue line) is shown to be inflated due to this artificial weighting of the loss terms. This indicates that when we are adding weights in the loss function, in order to accurately measure the progress of the training we need to focus on each individual loss term, and not the sum of all available terms.

Nevertheless, having a weighted loss function is important since if we leave them all equally weighted, we might end up with imbalanced loss terms which can impede training if, for example, the PDE residual loss is much smaller in magnitude than the boundary loss (or vice versa). Weighting the loss function gives the PINN the ability to target a specific MSE, which enables more efficient training. The selected weight parameters for the KdV equation are specified in Algorithm 4.3.

4.3 Sampling Strategy

A critical aspect of implementing PINNs is choosing the set of collocation points and the points for initial/boundary condition enforcement. In this work, we use a simple uniformly random sampling strategy to select training points in the space-time domain for each PDE. For each training epoch, we sample points (x_i^f, t_i^f) independently from a uniform distribution over the domain ($x \in [a, b]$, $t \in [0, T]$) to evaluate the PDE residual. Likewise, we sample initial condition points x_j^0 uniformly over the spatial domain (at $t = 0$) and boundary condition points (x_{bc}, t_k^b) uniformly over the temporal extent of the boundary. We drew a new random batch of collocation points each optimization iteration [30].

For the KdV equation which develops fast oscillations, a larger number of collocation points (and training epochs) was used to compensate for the random sampling's potential misses. From this dense uniform sampling, even the small-scale features have a good chance of being sampled. The only problem with this is the increased computational cost of sampling a larger number of points at every optimization iteration.

An alternative strategy could have been to select the collocation points with an adaptive sampling scheme, where the same accuracy could be achieved with fewer points by concentrating effort where there is a larger residual error by dynamically updating collocation points during training [41]. However, these are very advanced sampling techniques that are beyond the scope of this thesis, which also require very careful posing of the optimization problem, as training could easily become unstable for complex PDEs.

Therefore, we used uniformly random collocation and condition points as a baseline strategy in this thesis. This choice was guided by ease of implementation and the moderate complexity of our test problems. However, we recognize that for more challenging PDEs or higher-dimensional problems, adaptive sampling could substantially improve the PINN's convergence and accuracy by alleviating the issue of imbalanced error distribution across the domain.

4.4 Training Procedure

The training of all PINN models was carried out using the Adam optimizer with a fixed learning rate of $\eta = 10^{-3}$, without early stopping or learning rate decay. Some implementations [8] employ L-BFGS (a quasi-Newton second-order optimizer [70]) at the end of training to "polish" the solution after an initial Adam phase, but this algorithm can be memory-intensive for large networks and requires full-batch gradient computation. It was also found that the loss function failed to converge due to the introduced complexity in the optimization procedure (at the end of training the loss function remained very large), so we primarily report results using Adam only. The number of iterations used was increased as the complexity of the PDE increased, from 5000 to 9000. For all problems, collocation points were sampled uniformly at random for the PDE residual, initial condition, and boundary condition terms. Network architectures were kept consistent across problems, with two hidden layers of 128 neurons each and `Tanh` activations, except for the KdV model, which used a wider MLP with 512 hidden

units per layer. Fourier feature embeddings were used to enrich the input representation in the heat and Allen-Cahn equations, with mapping sizes and scales adjusted slightly for each PDE. Fourier features were not used in the case of the KdV equation given that it was found the optimization problem became ill-posed in this case, given the third-order complexity inherent in this PDE. Loss weights were kept uniform only in the Allen-Cahn PDE, which was sufficient to reach successful convergence, but for the KdV and heat PDE cases distinct weights were applied to each loss term to balance their contributions during optimization. For the heat equation PDE, both the PDE residual loss and the boundary condition loss terms were augmented by 10 additional points (both were multiplied by 10), and as for the KdV case, the used loss weights for training are: $w_{pde} = 45$, $w_{ic} = 50$, and $w_{bc} = 40$. This greatly enhanced the convergence of the network, given that both the PDE and initial condition loss terms in this case were rather computationally challenging to train (approximately 69 minutes).

In conclusion, the PINN methodology applied in this thesis was designed to be as straightforward as possible while still effective for the Heat, Allen-Cahn, and KdV equations. The results demonstrate that with a sufficiently deep choice of architecture (fully connected MLP), input encoding (Fourier features for oscillatory solutions), loss formulation (including all physical constraints), and optimization strategy (Adam only, for simplicity), PINNs can successfully solve PDEs ranging from simple linear diffusion to nonlinear, higher-order dynamics.

This Chapter has detailed the reasoning behind each component of the methodology, supported by established findings in the PINN literature, and sets the stage for the subsequent chapter, where we present and discuss the results obtained for each of the three example equations, highlighting the performance of the models and analyzing their effectiveness across different equation types. Each implementation serves as a case study in how PINNs cope with different types of PDEs, illustrating both the versatility of the approach and the practical considerations that guide its application.

Chapter 5

Results

This chapter presents the results obtained from implementing the PINN models described in the previous section. Each PDE case is evaluated individually, with attention to the accuracy of the learned solutions, training behavior, and comparison against reference solvers.

5.1 Comparison with Baselines

All experiments were executed using Python 3.13.2 and PyTorch version 2.6.0 on a MacBook Pro (M4 chip) with 24 GB of unified memory. The operating system was macOS Sequoia 15.4.1. All training was performed on the integrated GPU using Apple's Metal Performance Shaders (MPS) backend.

Table 5.1 summarizes the training performance for each of the three PDEs considered. The error norms reported for the PDE, initial condition (IC), and boundary condition (BC) terms correspond to empirical L^2 losses, computed over randomly sampled collocation points. Additionally, the total training time (in minutes) is included for each case, as measured on the specified hardware. It is made evident that the training time increases as the complexity of the PDE is increased.

Table 5.1: Training performance of PINN models for different PDEs

PDE	Epochs	PDE Loss	IC Loss	BC Loss	Time (min)
1D Heat Equation	5000	0.0001072	0.00000235	0.0001287	1.3
2D Heat Equation	5000	0.001609	0.0001239	0.002017	2.5
Allen-Cahn	5000	0.00202	0.00538	0.00110	10.3
KdV Equation	9000	0.00707	0.00759	0.00144	69.3

5.1.1 Heat Equation

Figure 5.1 shows the PINN's 2D temperature map for the sinusoidal initial condition, and Figure 5.2 plots the corresponding 3D surface. The network reproduces the exact exponential

Figure 5.1 1D Heat Equation: 2D representation.

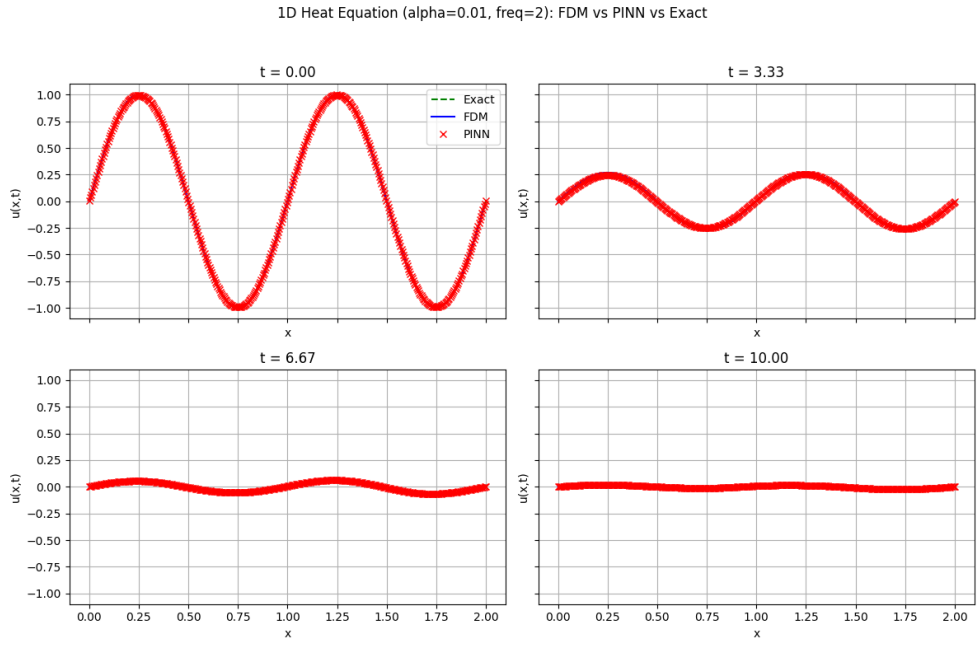
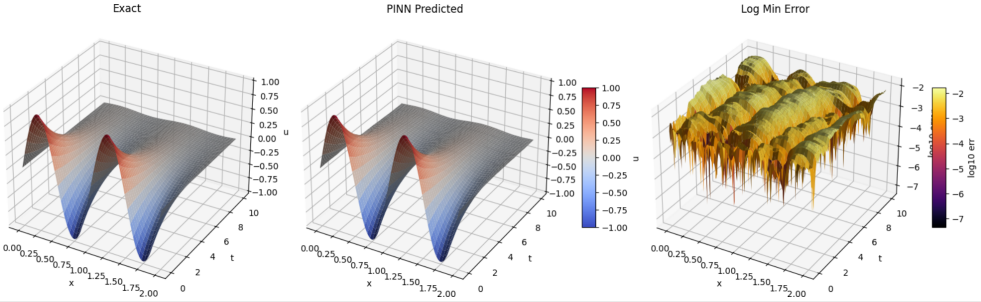


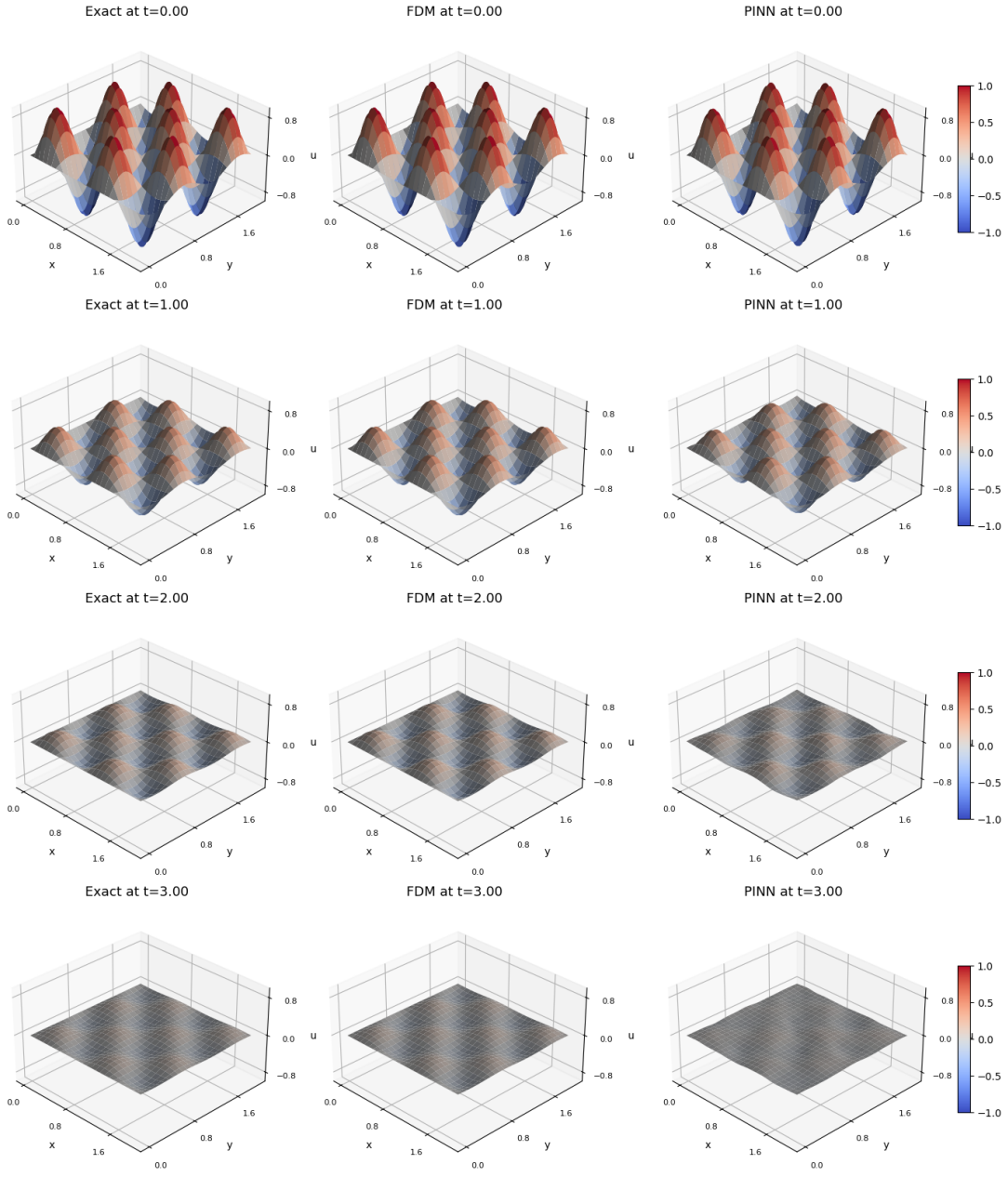
Figure 5.2 1D Heat Equation: 3D representation.



decay of the single Fourier mode with an empirical L^2 error of $\approx 10^{-4}$ (Table 5.1), matching the analytic solution without any explicit mesh. Training converged in just 1.3 min for the 1D case, demonstrating that a modest MLP can learn smooth, parabolic behavior as accurately as classical finite-difference solvers but in a fully mesh-free fashion.

Extending to two dimensions, the PINN captures the separable sine-sine mode on a square plate (Figure 5.3). Despite the added complexity of enforcing periodicity in both x and y, the network maintains the same order of accuracy as in 1D, with negligible amplitude error. This confirms that PINNs scale naturally to higher dimensions, offering similar precision to finite-difference methods on simple geometries.

Figure 5.3 2D Heat Equation: 3D representation.



5.1.2 Allen-Cahn Equation

Figure 5.4 presents the PINN's 3D reconstruction of the diffuse tanh front, while Figure 5.5 overlays the learned profile against the high-order spectral reference. The network faithfully captures the sharp interface and its slow coarsening, with point-wise errors well below 10^{-2}. Although training took about 10 min and required careful tuning of the boundary-loss

Figure 5.4 Allen-Cahn Equation: 3D representation.

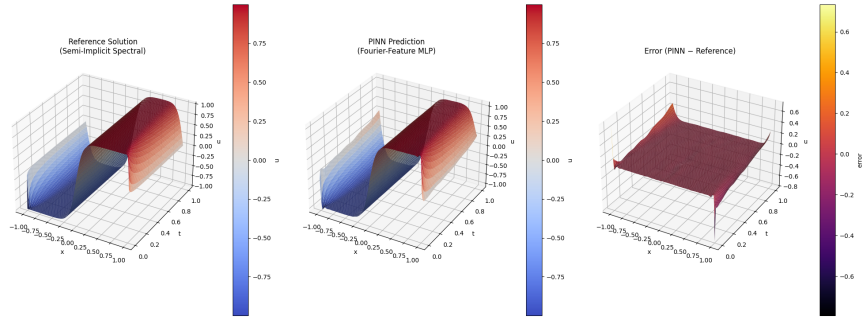
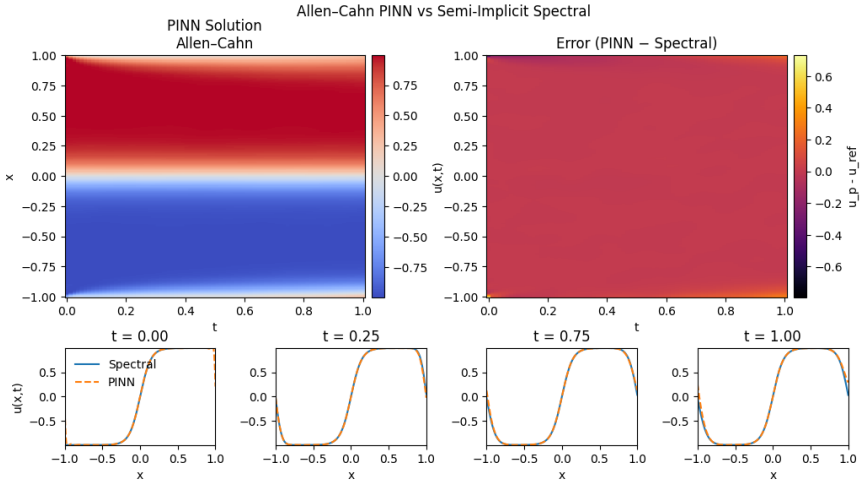


Figure 5.5 Allen-Cahn Comparison with spectral numerical solver.



weight, the PINN achieves near spectral-solver accuracy on this nonlinear reaction-diffusion problem, demonstrating its ability to handle cubic reaction terms and moving fronts. It can be appreciated on the loss plot that the PINN approximation makes an almost consistently correct generalization across different time steps, making the error plot look very flat, and the line comparison in Figure 5.5 almost identical to the spectral reference.

5.1.3 KdV Equation

Figures 5.6, 5.7, and 5.8 illustrate the PINN's approximation of the wave from different viewpoints: as a heatmap, and in direct time-step comparison with the pseudo-spectral + BDF reference. Even with third-order spatial derivatives and nonlinear components, the network reproduces both amplitude and phase of the PDE with L^2 errors on the order of 10^{-2}. This required 9000 epochs and around 69 min of training, reflecting the greater challenge of balancing PDE, IC, and BC losses. Nevertheless, the result showcases the PINN's flexibility in capturing dispersive wave dynamics without constructing a mesh or explicit time integrator, ideal for computational simulation tasks.

Figure 5.6 KdV Equation: 3D representation from different angles.

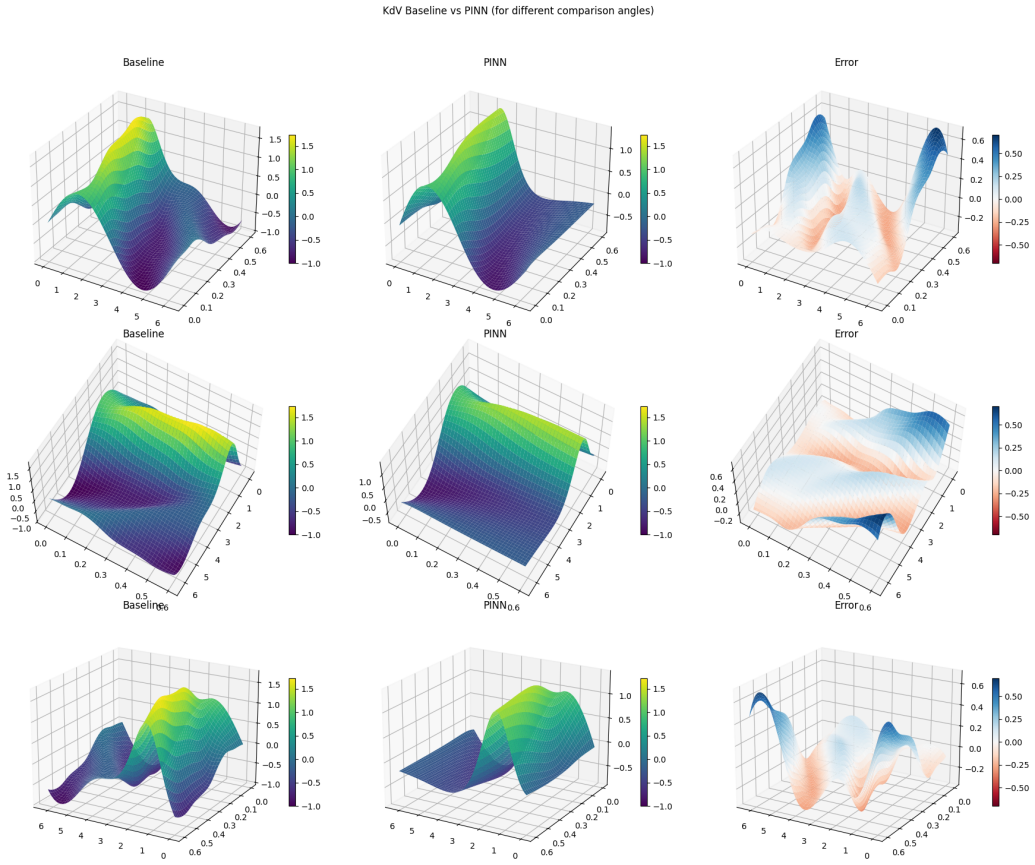


Figure 5.7 KdV Equation: Heat map.

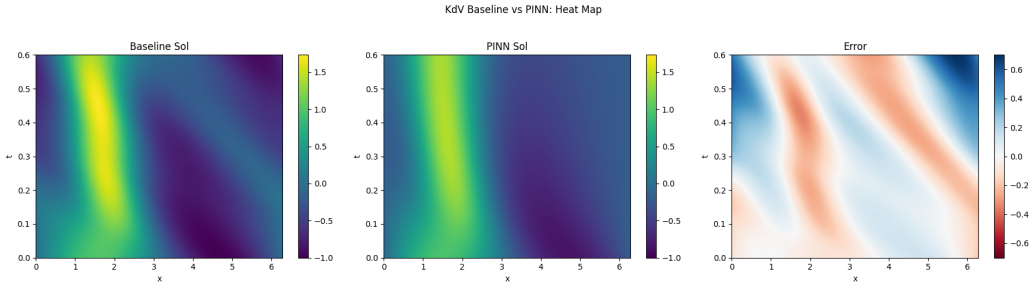
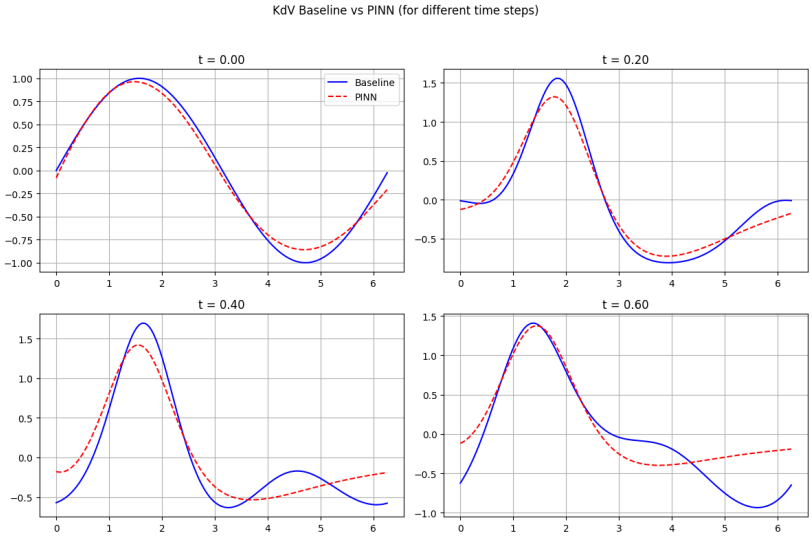


Figure 5.7 shows the pointwise absolute error of the PINN's solution as a heat map, highlighting small regions, primarily around steep gradients, where the network underestimates the wave amplitude by up to a few percent. Nevertheless, these discrepancies do not undermine the overall usefulness of the PINN approximation: the waveform, phase speed, and conserved quantities (mass and energy integrals) are captured with high fidelity. The PINN solution can be used to quickly generate continuous spatio-temporal predictions or

Figure 5.8 KdV Equation: Time step comparison.



extract soliton trajectories without the need for costly grid-based remeshing. Moreover, its differentiable nature makes it ideal for sensitivity analysis [31], inverse parameter estimation [8], and automated discovery of underlying PDE operators from data [1], even where small pointwise errors exist because the dominant dynamics remain accurately represented across the domain.

5.2 Discussion

It was found that one of the major challenges in training the PINN until successful convergence was the selection of the weights that govern the loss function. Sometimes, despite the successful learning in the initial conditions of the problem, the PINN would have trouble learning the boundary conditions across the time domain. It is here where it becomes useful to assign a higher value to the boundary condition loss term, as the training would then assign a higher weight to this particular part of the loss function. This has the effect of being able to get different error accuracies of the reconstructed fields. However, when the PDE is very simple in nature, assigning high values to the loss term made the training very low to converge, sometimes even making the network unable to do so.

The ability of the network to converge also proved to be highly dependent on the number of collocation data points it needed to train on. It could be reasoned that the more collocation points the network must validate the physics against, the better the reconstructed error should be. However, it was found that as we make the density of points higher, the more unstable the training would become. The best results were achieved when trying to be considerate of the number of points the network would have to validate its physics results against, and that this number should be picked with the number of training iterations that were being done to train the whole network. The simpler the PDE, the less overkill the problem setup should be,

as trying to estimate really dense reconstruction fields highly increases the instability during training.

The KdV equation has a shock that develops at the point $t = 0.40$. This shock is where the largest errors occur for PINNs trained using each sampling method. However, our proposed sampling method achieves the lowest errors in this scenario, with the distribution of sampling points concentrated around the shock location. We hypothesize that this concentration ensures that the model trained using this distribution of collocation points better captures the steep gradients associated with the shock. In addition, the ability of our method to maintain accuracy away from the shock highlights its capacity to balance local accuracy and global consistency, addressing challenges presented by the nonlinear dynamics of the KdV equation. It can be seen how the numerical approximation has some trouble adapting to the PDE dynamics, whereas the PINN adaptation provides a more stable solution.

Moreover, despite localized unstable regions near steep gradients, the PINN demonstrates strong overall generalization: it accurately reconstructs the full solution profile even in far-field regions where no shocks occur. This robustness makes PINNs not only effective for forward simulations of heat conduction, wave propagation, and phase-field dynamics, but also exceptionally well-suited for inverse problems, such as parameter estimation in reactive flows, and for automated equation discovery from data [8, 1].

Across all three cases, PINNs deliver accuracy on par with classical FDM and spectral solvers, often within an order of magnitude of the analytic or high-fidelity reference, while operating in a mesh-free, fully differentiable framework. The main trade-off is computational cost: training times grow with PDE complexity, from 1.3 min for linear heat diffusion to over an hour for nonlinear, dispersive waves (KdV PDE). Still, PINNs demonstrate a unified approach to solving diverse PDEs, overlooking the step of mesh generation and offering a flexible alternative to traditional numerical methods.

In this chapter, we demonstrated that PINNs achieve accuracy comparable to classical finite-difference and spectral methods, often within an order of magnitude of high-fidelity references, across heat diffusion, nonlinear phase-field, and dispersive wave problems. This nonetheless comes with a cost of longer training times for more complex PDEs. These results underscore the viability of a mesh-free, fully differentiable framework for solving diverse PDEs, while also revealing the computational challenges inherent in scaling to higher complexity. In Chapter 6, we will summarize found challenges and present possible future research directions to accelerate convergence, manage computational cost, and further bridge the gap between PINNs and traditional numerical solvers.

Chapter 6

Conclusions

In this thesis, we have systematically investigated the application of Physics-Informed Neural Networks (PINNs) to three classes of partial differential equations: the linear, second-order parabolic heat equation, the nonlinear reaction-diffusion Allen-Cahn equation, and the nonlinear, dispersive, third-order Korteweg-de Vries (KdV) equation.

6.1 Contributions

A unified PINN framework is proposed, where we designed a single, modular PINN methodology comprising a fully connected multilayer perceptron with smooth (tanh) activations, elective Fourier input embeddings, and a composite loss enforcing PDE residuals, boundary/initial conditions, and an optional data supervision that can be applied without modification across diverse PDE types [25, 7]. Our PINN implementation only relied on the first three loss terms in the building of the loss function. Across all three PDEs, our PINNs achieved relative L^2 errors within one order of magnitude of classical finite-difference and spectral solvers, validating the mesh-free, differentiable approach against high-fidelity reference solutions [19]. We quantified the trade-off between solver complexity and training time. It was found that the biggest challenge in implementing PINNs is that as the complexity and magnitude of the optimization problem increases, the more complicated it is to converge to a desired solution. This difficulty also gets amplified if we have a higher order, highly nonlinear PDE loss function ingrained in the PINN optimization problem. A weight-tuning study is also presented, providing insights into the importance of correctly posing the optimization problem at the beginning of training.

The work shows the critical role of loss-term weights and uniform random sampling in balancing convergence of physics, boundary, and initial condition residuals, and discussed how residual-driven adaptive schemes could further improve robustness [30, 41]. We analyzed usual PINN failure modes, boundary condition violations, spectral bias in high-frequency regimes, and convergence stalls, and identified these as key optimization bottlenecks that make the loss landscape very hard to optimize [48, 47]. These contributions establish PINNs as a flexible, mesh-free alternative to classical solvers, capable of addressing both linear and nonlinear PDEs within a differentiable programming paradigm.

Despite all these difficulties in training these types of networks, PINNs unlock a playground where physics and machine learning unite to create simulation models that generalize beyond their training data. Their mesh-free nature eliminates the need for grid generation around complex geometries or moving boundaries, paving the way for real-time simulations in materials science [35], biomedicine [13, 14], pharmacology [39], power systems [36], and manufacturing systems [38]. By combining PINNs with classical solvers in hybrid schemes, researchers are achieving fast predictions that rigorously respect physical laws within their research domains [12]. Looking forward, PINNs are poised to revolutionize physics research workflows by enabling experimental design for inverse problems [9], automated discovery of new governing systems equations [1], and optimizing the design and efficiency of structures exposed to fluid flows, such as bridges, offshore platforms, and wind turbine blades [10].

Relying on the function approximation capabilities of the feedforward neural network provides accurate and differentiable solutions in an analytic form. The success of the method can be attributed to two factors. The first one is the employment of neural networks that are excellent function approximators and the second is the form of the trial solution that satisfies by construction the BCs and therefore the constrained optimization problem becomes a substantially simpler unconstrained one [25].

While PINNs have primarily focused on integrating partial differential equations (PDEs) as input constraints, there is a vast universe of physical knowledge that can be tapped into, some of which may not even be known yet as new discoveries remain to be found. By expanding our understanding and incorporating additional laws governing different phenomena, we can enhance the capabilities of AI models even further. Imagine a world where PINNs can seamlessly incorporate principles from quantum mechanics or electromagnetism to tackle complex problems in chemistry or electronic device design. By capturing the essence of these fundamental laws within neural networks, we could unlock new frontiers in scientific research and technological innovation. From predicting molecular interactions to optimizing energy-efficient electronic circuits, there are only more good things on the horizon [10].

6.2 Strengths and Limitations

Physics-informed neural networks combine data and governing equations in a single, differentiable model, eliminating the need for intricate mesh generation and enabling seamless gradient-based inverse modeling and control applications [1, 25]. Moreover, a single network architecture can handle linear, nonlinear and higher-order problems alike, demonstrating true methodological unification across diverse PDE classes.

- Mesh-free differentiability: No grid generation—direct evaluation at arbitrary points in space–time.
- Small-data robustness: Accurate recovery from limited or noisy measurements, even achieving robust performance even with sparse or noisy observations, often outperforming purely data-driven surrogates in low-data regimes

- Unified framework: One PINN base design (excluding Fourier features which are problem-specific) scales from heat diffusion to reaction–diffusion to dispersive waves.

Despite these advantages, PINNs face several practical difficulties: training can be computationally expensive for stiff or high-frequency PDEs, and convergence often requires careful loss-weight tuning and initialization. Spectral bias and ill-conditioned composite losses can stall optimization or produce nonphysical solutions without bespoke adjustments. Extending PINNs to high-dimensional domains, complex geometries or coupled systems remains an open challenge, and rigorous error estimates and convergence proofs are still under development [48].

- High computational cost: Long training times limit real-time or large-scale use.
- Optimization fragility: Sensitive to weight settings, network initialization and sampling strategy.
- Scalability: PINNs are nontrivial to apply in many dimensions or intricate geometries. As the complexity of the governing PDE is higher, the higher the training compute time required to reach a decent-enough solution.
- Theoretical gaps: Formal error bounds and convergence guarantees are not yet established with rigorous foundations.

With each technical problem that is overcome, PINNs open new doors for innovation across science and engineering.

6.3 Future Work

It goes without saying, there is a lot of work to be done in this field. Building on the insights and limitations identified, several promising directions are outlined for advancing the PINN paradigm.

- Reinforcement learning for adaptive collocation point selection: Recent studies (2025) have suggested that formulating the selection of collocation points as a Markov Decision Process, where they replace gradient-dependent residual metrics with a computationally efficient function variation as the reward signal can outperform existing residual-driven adaptive methods in accuracy [71]. This could serve as a logical extension to existing work on multi-fidelity sampling, which propose hierarchical collocation schemes (such as V-cycle strategies) to concentrate training effort in high-error regions and accelerate convergence for stiff or multiscale PDEs [44]. The combination of these two ideas is something that has not yet been explored.

- Incorporating numerical methods as derivative solvers directly into the PINN loss function: Another recent study has delved into the introduction of an RK4 scheme directly into the MLP loss function, which shows that PINN training becomes far more stable and predictable [72]. This is a promising research direction to explore given the wide availability of numerical methods that could be experimented with. This also has the potential to evolve into a sort of hybrid method between classical numerical solvers and PINNS, which would allow for a structured discretization with the flexibility of data-driven neural approximators.

- More research in inverse problems and uncertainty quantification: There has been some research done on this domain (parameter identification [39] and Bayesian PINNs [73]), quantifying solution uncertainty under noisy observations and model discrepancies. However more research is promising in this direction given the inherent difficulties that come about inverse modeling, which could be greatly enhanced with the correct PINN approach with applications even in the field of geophysics and biomedical imaging.

- Domain decomposition: As was briefly mentioned in Chapter 2, domain decomposition leverages finite-basis and overlapping-subdomain PINNs to partition large or complex geometries into smaller, tractable subproblems, enabling scalable training and improved optimization landscapes [43]. FBPINNs have the potential to reduce the complexity and nonconvexity of PINN optimization problems, and developing these techniques to more robust standards could be beneficial for other PINN practitioners who may have problems making their PINNs converge to desirable solutions.

- Explore the convergence of different neural network architectures and neural activation functions: Despite the highly custom set of hyperparameters that are needed to make a PINN converge as the complexity of the governing PDE evolves, a sort of categorization could be done as to what works under what conditions. New architectural components such as SIREN [74], residual nets [75], and autoencoder [76] mechanisms could be explored to see if they allow for better convergence in these kinds of optimization problems. Other ideas in this domain include adaptive loss weights [77, 78], which is something that was shortly attempted in this work but did not work due to improper adaptation to the PDE at hand. The initial weight parameters that are set in the initial stages of training have a huge influence on whether the PINN converges to a decent solution. This could be a promising research direction: basically, to define the conditions under which the optimization problem becomes ill-conditioned or unstable.

By addressing these open challenges, future research can further bridge the gap between PINNs and conventional solvers, unlocking their full potential for scientific computing and engineering applications.

As we stand at the intersection of physics and machine learning, PINNs show us that the boundaries between data-driven insight and first-principles understanding are not walls but bridges. Neural networks can indeed carry the hat of classical solvers, creating new pathways in science and engineering. Yet, this is just the prelude. Each challenge, be it spectral

bias or high-dimensional complexity, calls for a deeper collaboration between practitioners. Imagine PINNs that not only solve equations but propose the governing laws themselves, that adapt in real time to streaming experimental data, or that co-pilot the design of next-generation materials and devices. In embracing these possibilities, we do more than accelerate computations; we reshape the very methods by which discoveries are made.

The road ahead will demand creativity, rigor, and humility. But the potential rewards, for both our theories and our technologies, are boundless. The grand experiment of fusing AI with the wisdom of physics lies before us, and the world awaits the next leap.

Appendix

All of the code used for experimentation in this project is publicly available at GitHub. The goal of the repository is define a baseline that could be scaled to further apply the project to many different types of PDEs, with the possibility of implementing many different types of novel techniques such as different neural architectures, adaptive selection of collocation points, and adaptive weight terms, to name a few of the possible directions of the project. The modular and interactive definition of the repository also facilitated the experimentation of different techniques across the development of the initial project, and the logging of the different parameters that were used under each of the saved outputs.

https://github.com/josegarciav/PINNs-RL-PDE/tree/main/src/numerical_solvers

The top-level structure of the repository is organized as follows:

1. src/numerical solvers/: Definition of the scripts needed to implement numerical solver (RK4, FDM, etc.) vs. PINN implementations to particular types of PDEs. This is the most relevant folder to the results presented in this work.
2. src/neural networks/: Definitions of the PINN architectures (architecture types).
3. src/pdes/: High-level definition of the PDE loss terms based on the predefined boundary/initial condition types, exact solutions, and ad hoc particularities of each of the PDEs.
4. interactive trainer: This is the main file from where the interactive trainer is launched. This launches a GUI-type application that brings all the elements of the project together, allowing for the dynamic selection of the PDE type, architecture to be tested, and parameter tuning. This launches a training session that saves the obtained results for later comparison, along with all the parameters that triggered the training session.

For more details, see the README file in the repository.

Bibliography

[1] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations. *CoRR*, abs/1711.10566, 2017. See pages v, 1, 10, 11, 40, 41, and 44.

[2] Juan D. Toscano, Vivek Oommen, Alan J. Varghese, Zongren Zou, Nazanin A. Daryakenari, Chenxi Wu, and George E. Karniadakis. From PINNs to PIKANs: Recent advances in physics-informed machine learning. *Machine Learning for Computational Science and Engineering*, 1, 2025. See pages v, 10, 13, and 15.

[3] Franz M Rohrhofer, Stefan Posch, Clemens Gößnitzer, and Bernhard C Geiger. Data vs. physics: The apparent pareto front of physics-informed neural networks. *IEEE Access*, 11, 2023. See pages v, 10, 11, 13, 16, and 19.

[4] Prateek Bhustali. Physics-Informed-Neural-Networks (PINNs), 2021. GitHub repository, MIT License. See pages v, 28, and 29.

[5] Zheyuan Hu, Khemraj Shukla, George Em Karniadakis, and Kenji Kawaguchi. Tackling the curse of dimensionality with physics-informed neural networks. *Neural Networks*, 176, 2024. See pages 1, 2, 10, and 13.

[6] George E. Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3, 2021. See pages 1, 2, 10, and 17.

[7] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations. *CoRR*, abs/1711.10561, 2017. See pages 1, 10, 11, 12, 13, and 43.

[8] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 2019. See pages 1, 10, 17, 31, 33, 40, and 41.

[9] Apivich Hemachandra, Gregory Kang Ruey Lau, See-Kiong Ng, and Bryan Kian Hsiang Low. PIED: Physics-informed experimental design for inverse problems. In *Proceedings of the International Conference on Learning Representations (ICLR) 2025 Workshop on AI for Science*. International Conference on Learning Representations (ICLR), 2025. See pages 1, 44.

[10] Mirko Peters. Revolutionizing AI: The Role of Physics-Informed Neural Networks. *Medium* post, 2025. See pages 1, 2, 14, and 44.

[11] J. C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2nd edition, 2004. See pages 1, 5, 6, and 22.

[12] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific Machine Learning Through Physics–Informed Neural Networks: Where We Are and What's Next. *Journal of Scientific Computing*, 92, 2022. See pages 2, 44.

[13] Francisco S. Costabal, Yibo Yang, Paris Perdikaris, Daniel E. Hurtado, and Ellen Kuhl. Physics-informed neural networks for cardiac activation mapping. *ELSP: Advanced Manufacturing*, 2020. See pages 2, 14, and 44.

[14] Carlos Ruiz Herrera, Thomas Grandits, Gernot Plank, Paris Perdikaris, Francisco Sahli Costabal, and Simone Pezzuto. Physics-informed neural networks to learn cardiac fiber orientation from multiple electroanatomical maps. *Engineering with Computers*, 38(5), 2022. See pages 2, 14, and 44.

[15] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, 6th edition, 2005. See page 5.

[16] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2nd edition, 2008. See pages 5, 24, and 25.

[17] L. Euler. *Institutiones Calculi Differentialis*. Imperial Academy of Sciences, 1755. See page 5.

[18] A. L. Cauchy. *Cours d'Analyse de l'École Royale Polytechnique*. Imprimerie Royale, 1821. See page 5.

[19] R. J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, 2007. See pages 5, 6, 7, 8, 9, and 43.

[20] G. Dahlquist and Å. Björck. *Numerical Methods*. Prentice Hall, 1974. See pages 5, 25, and 26.

[21] Michael Zeltkevic. Numerical Solution of Initial Value Problems: Runge-Kutta Methods. *MIT*, 1998. See page 6.

[22] J. Crank and P. Nicolson. A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type. *Proc. Cambridge Philosophical Soc.*, 43, 1947. See page 6.

[23] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Phil. Trans. R. Soc. A*, 210, 1911. See page 6.

[24] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang. *Spectral Methods: Fundamentals in single domains*. Springer, 2nd edition, 2006. See page 9.

[25] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 1997. See pages 9, 43, and 44.

[26] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3), 2020. See page 9.

[27] Boris Hanin. Universal function approximation by deep neural nets with bounded width and relu activations. *Mathematics*, 7(10), 2019. See page 10.

[28] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden,

Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016. See page 11.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. *Neural Information Processing Systems*, 2017. See page 11.

[30] Chenxi Wu, Min Zhu, Qinyang Tan, Yadhu Kartha, and Lu Lu. A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 403, 2023. See pages 11, 33, and 43.

[31] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. *CoRR*, abs/1502.05767, 2018. See pages 13, 40.

[32] Maziar Raissi. Forward–backward stochastic neural networks: deep learning of high-dimensional partial differential equations. In *Peter Carr Gedenkschrift: Research Advances in Mathematical Finance (Chapter 18)*. World Scientific, 2023. See pages 13, 17.

[33] Sifan Wang, Bowen Li, Yuhan Chen, and Paris Perdikaris. PirateNets: Physics-informed deep learning with residual adaptive networks. *Journal of Machine Learning Research*, 25(402), 2024. See page 13.

[34] Kart Leong Lim, Rahul Dutta, and Mihai Rotaru. Physics informed neural network using finite difference method. *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2022. See page 13.

[35] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks for heat transfer problems. *The American Society of Mechanical Engineers*, 2021. See pages 13, 44.

[36] Bin Huang and Jianhui Wang. Applications of physics-informed neural networks in power systems - A review. *IEEE Transactions on Power Systems*, 2023. See pages 14, 44.

[37] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, 37(12), 2021. See page 14.

[38] Shoulan Yang, Shitong Peng, Jianan Guo, and Fengtao Wang. A review on physics-informed machine learning for monitoring metal additive manufacturing process. *ELSP: Advanced Manufacturing*, 2024. See pages 14, 44.

[39] Nazanin Ahmadi Daryakenari, Mario De Florio, Khemraj Shukla, and George Em Karniadakis. AI-Aristotle: A physics-informed framework for systems biology gray-box identification. *PLOS Computational Biology*, 20(3), 2024. See pages 14, 44, and 46.

[40] Ehsan Kharazmi, Zhongqiang Zhang, and George E. Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374, 2021. See page 15.

[41] Adrian Celaya, David Fuentes, and Beatrice Riviere. An Adaptive Collocation Point Strategy For Physics Informed Neural Networks via the QR Discrete Empirical Interpolation Method. *arXiv preprint arXiv:2501.07700*, 2025. See pages 15, 33, and 43.

[42] Guofei Pang, Lu Lu, and George Em Karniadakis. fPINNs: Fractional Physics-Informed Neural Networks. *SIAM Journal on Scientific Computing*, 41(4), 2019. See page 15.

[43] Ben Moseley, Andrew Markham, and Tarje Nissen-Meyer. Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *Advances in Computational Mathematics*, 49(4), 2023. See pages 16, 46.

[44] Alexander Heinlein, Amanda A Howard, and Damien Beecroft. Multifidelity domain decomposition-based physics-informed neural networks and operators for time-dependent problems. *Mathematical Optimization for Machine Learning: Proceedings of the MATH+ Thematic Einstein Semester*, 79, 2023. See pages 17, 45.

[45] Yao-Hsuan Tsai, Hsiao-Tung Juan, Pao-Hsiung Chiu, and Chao-An Lin. Multi-level datasets training method in physics-informed neural networks. *arXiv preprint arXiv:2504.21328v1*, 2025. See page 17.

[46] Chuwei Wang, Shanda Li, Di He, and Liwei Wang. Is L2 physics informed loss always suitable for training physics informed neural network? *Advances in Neural Information Processing Systems*, 35, 2022. See page 17.

[47] Nathan Doumèche, Gérard Biau, and Claire Boyer. Convergence and error analysis of PINNs. *arXiv preprint arXiv:2305.01240*, 2023. See pages 17, 43.

[48] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in neural information processing systems*, 34, 2021. See pages 18, 43, and 45.

[49] Francisco Eiras, Adel Bibi, Rudy Bunel, Krishnamurthy Dj Dvijotham, Philip H. S. Torr, and M. Pawan Kumar. Efficient error certification for physics-informed neural networks. In *International Conference on Machine Learning*, 2023. See page 19.

[50] Olga Fuks and Hamdi A Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, 2020. See page 19.

[51] Franz M. Rohrhofer, Stefan Posch, Clemens Gößnitzer, and Bernhard C. Geiger. On the role of fixed points of dynamical systems in training physics-informed neural networks. *Transactions on Machine Learning Research*, 2023. See page 19.

[52] Jian Cheng Wong, Chinchun Ooi, Abhishek Gupta, and Yew-Soon Ong. Learning in sinusoidal spaces with physics-informed neural networks. *IEEE Transactions on Artificial Intelligence*, 2022. See page 19.

[53] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality for training physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 421, 2024. See page 19.

[54] H. Carslaw and J. Jaeger. *Conduction of Heat in Solids*. Oxford Science Publications, 2nd edition, 1986. See page 21.

[55] Nikolas Provatas and Ken Elder. *Phase-Field Methods in Materials Science and Engineering*. Wiley-VCH, 2010. See pages 21, 23.

[56] Colby L. Wight and Jia Zhao. Solving Allen-Cahn and Cahn-Hilliard Equations using the adaptive physics informed neural networks. *arXiv preprint arXiv:2007.04542*, 2020. See page 21.

[57] D. J. Korteweg and G. De Vries. On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves. *Philosophical Magazine Series*, 39(5), 1895. See pages 21, 24.

[58] N. J. Zabusky and M. D. Kruskal. Interaction of solitons in a collisionless plasma and the recurrence of initial states. *Physical Review Letters*, 15(6), 1965. See pages 21, 24.

[59] Andrey I Maimistov. Completely integrable models of non-linear optics. *Pramana*, 57, 2001. See page 21.

[60] Samuel M. Allen and John W. Cahn. A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. *Acta Metallurgica*, 27(6), 1979. See page 23.

[61] Mark J. Ablowitz and Harvey Segur. *Solitons and the Inverse Scattering Transform*. Society for Industrial and Applied Mathematics (SIAM), 1981. See page 24.

[62] Roger Grimshaw. Korteweg–de Vries Equation. In *Nonlinear Waves in Fluids: Recent Advances and Modern Applications*, volume 483 of *CISM International Centre for Mechanical Sciences*. Springer, 2007. See page 24.

[63] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3), 2020. See page 25.

[64] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Internal Representations by Error Propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. MIT Press, 1986. See page 27.

[65] Stefano Markidis. The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers? *Frontiers in Big Data*, 2021. See page 27.

[66] Shaghayegh Fazliani, Zachary Frangella, and Madeleine Udell. Enhancing physics-informed neural networks through feature engineering. *arXiv preprint arXiv:2502.07209*, 2025. See page 27.

[67] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33, 2020. See page 28.

[68] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010. See page 28.

[69] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2017. See page 29.

[70] Dong C. Liu and Jorge Nocedal. On the limited memory BFGS Method for large scale optimization. *Springer Mathematical Programming*, 45, 1989. See page 33.

[71] Zhenao Song. RL-PINNs: Reinforcement Learning-Driven Adaptive Sampling for Efficient Training of PINNs. *arXiv preprint arXiv:2504.12949*, 2025. See page 45.

[72] Jiaming Zhang, David Dalton, Hao Gao, and Dirk Husmeier. Physics-informed deep learning based on the finite difference method for efficient and accurate numerical solution of partial differential equations. In *Proceedings of the International Conference on Statistics: Theory and Applications (ICSTA '24)*. Avestia Publishing, 2024. See page 46.

[73] Liu Yang, Xuhui Meng, and George Em Karniadakis. B-PINNs: Bayesian Physics-Informed Neural Networks for Forward and Inverse PDE Problems with Noisy Data. *Journal of Computational Physics*, 425, 2020. See page 46.

[74] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit Neural Representations with Periodic Activation Functions. *CoRR*, abs/2006.09661, 2020. See page 46.

[75] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385, 2015. See page 46.

[76] Hervé Bourlard and Yves Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59, 1988. See page 46.

[77] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*. PMLR, 2018. See page 46.

[78] Zixue Xiang, Wei Peng, Xiaohu Zheng, Xiaoyu Zhao, and Wen Yao. Self-adaptive loss balanced physics-informed neural networks for the incompressible navier-stokes equations. *Neurocomputing*, 496, 2021. See page 46.