8-21-2023 9:45 AM

Data-Driven Exploration of Coarse-Grained Equations: Harnessing Machine Learning

Elham Kianiharchegani, *The University of Western Ontario*

Supervisor: Karttunen, Mikko, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Applied Mathematics

Abstract

In scientific research, understanding and modeling physical systems often involves working with complex equations called Partial Differential Equations (PDEs). These equations are essential for describing the relationships between variables and their derivatives, allowing us to analyze a wide range of phenomena, from fluid dynamics to quantum mechanics. Traditionally, the discovery of PDEs relied on mathematical derivations and expert knowledge. However, the advent of data-driven approaches and machine learning (ML) techniques has transformed this process. By harnessing ML techniques and data analysis methods, data-driven approaches have revolutionized the task of uncovering complex equations that describe physical systems. The primary goal in this thesis is to develop methodologies that can automatically extract simplified equations by training models using available data. ML algorithms have the ability to learn underlying patterns and relationships within the data, making it possible to extract simplified equations that capture the essential behavior of the system. This study considers three distinct learning categories: black-box, gray-box, and white-box learning.

The initial phase of the research focuses on black-box learning, where no prior information about the equations is available. Three different neural network architectures are explored: multi-layer perceptron (MLP), convolutional neural network (CNN), and a hybrid architecture combining CNN and long short-term memory (CNN-LSTM). These neural networks are applied to uncover the non-linear equations of motion associated with phase-field models, which include both non-conserved and conserved order parameters.

The second architecture explored in this study addresses explicit equation discovery in gray-box learning scenarios, where a portion of the equation is unknown. The framework employs eXtended Physics-Informed Neural Networks (X-PINNs) and incorporates domain decomposition in space to uncover a segment of the widely-known Allen-Cahn equation. Specifically, the Laplacian part of the equation is assumed to be known, while the objective is to discover the non-linear component of the equation. Moreover, symbolic regression techniques are applied to deduce the precise mathematical expression for the unknown segment of the

equation.

Furthermore, the final part of the thesis focuses on white-box learning, aiming to uncover equations that offer a detailed understanding of the studied system. Specifically, a coarse parametric ordinary differential equation (ODE) is introduced to accurately capture the spreading radius behavior of Calcium-magnesium-aluminosilicate (CMAS) droplets. Through the utilization of the Physics-Informed Neural Network (PINN) framework, the parameters of this ODE are determined, facilitating precise estimation. The architecture is employed to discover the unknown parameters of the equation, assuming that all terms of the ODE are known. This approach significantly improves our comprehension of the spreading dynamics associated with CMAS droplets.

Keywords: Coarse-graining, Machine learning, Neural Network, Symbolic regression

Summary for Lay Audience

This thesis is centered around the application of machine learning techniques for uncovering hidden patterns and equations in complex physical systems. It showcases the transformative potential of machine learning and data-driven approaches in revolutionizing the process of understanding and describing complex equations. Traditional methods of deriving equations from data often require significant time and expertise. However, with the advent of data-driven approaches and machine learning, we can automate and improve this process. The thesis delves into three distinct learning approaches: black-box, gray-box, and white-box learning. Through these approaches, the thesis explores different ways of learning and extracting insights from data, ranging from scenarios where no prior knowledge of the equations is available to cases where some parts of the equations are known.

In black-box learning, neural network models are developed to uncover non-linear equations governing phase-field models without any prior knowledge of the equations. These models capture the behavior of the systems solely based on the provided data.

In gray-box learning, extended physics-informed neural networks (X-PINNs) are employed to reveal unknown components of an equation. By incorporating domain decomposition and symbolic regression techniques, we can determine the missing components of the equation using the available data.

Finally, in white-box learning, the primary objective is to achieve a comprehensive understanding of a system through the utilization of the physics-informed neural network (PINN) framework. Specifically, this approach focuses on predicting the parameters of a coarse parametric ordinary differential equation (ODE) that accurately characterizes the spreading radius behavior of CMAS droplets.

Acknowledgements

I would like to sincerely express my profound gratitude to my supervisor, Professor Mikko Karttunen, for his expertise and unwavering support throughout my entire PhD journey. Collaborating with him has been an exceptionally rewarding experience. His mentorship has played a crucial role in achieving significant progress in the project, while also leaving a profound impact on my personal academic growth. I am truly thankful for the invaluable opportunities and knowledge gained under his expert supervision.

I would also like to extend my gratitude to Professor George Em Karniadakis and his research group, particularly Professor Khemraj Shukla, for providing me with the invaluable opportunity to engage in research exchange visits at Brown University, USA. Additionally, I would like to express my thanks to Professor Mishra Siddhartha for supporting me during my visit to ETH, Switzerland. The experiences gained during these visits have been incredibly enriching. I would like to extend my heartfelt gratitude to Dr. Mahdi Kooshkbaghi, who has been involved in this project from its inception to its completion. Collaborating with him has been an invaluable experience, and I am genuinely thankful for his guidance in addressing my questions.

I want to extend my heartfelt gratitude to each and every one of my family members for their unwavering and unconditional support throughout this journey. Their encouragement has been the driving force behind my achievements, and I am truly grateful for their constant presence in my life. Throughout this journey, I have been privileged to meet and connect with many wonderful people. Each individual, in their unique way, has left a profound impact on my path. I apologize for not mentioning each of you by name, but I am grateful for the moments we have cherished together.

I gratefully acknowledge the funding support provided by Ontario Graduate Scholarship (OGS), Mitacs Globalink Research Award, Western University's Science International Engagement Fund Award, and Flight 752 Memorial Graduate Scholarship. This financial assistance has been of immense help in conducting my research.

Co-Authorship Statement

This thesis contains the following manuscripts that have been published or submitted:

Chapter 4 has been published in Physical Review E.

E. Kiyani, S. Silber, M. Kooshkbaghi, and M. Karttunen, Machine-learning-based data-driven discovery of nonlinear phase-field dynamics, Physical Review E, 106, 065303, 2022.

Chapter 5 has been published in Computer Methods in Applied Mechanics and Engineering.

E. Kiyani, K. Shukla, G. Em Karniadakis, and M. Karttunen, A framework based on symbolic regression coupled with eXtended Physics-Informed Neural Networks for gray-box learning of equations of motion from data, Computer Methods in Applied Mechanics and Engineering, 415, p.116258, 2023.

Chapter 6 has been submitted to the Journal of Fluid Mechanics.

E. Kiyani, M. Kooshkbaghi, K. Shukla, R. Babu Koneru, Z. Li, L. Bravo, A. Ghoshal, G. Em Karniadakis, and M. Karttunen, Characterization of partial wetting by CMAS droplets using multiphase many-body dissipative particle dynamics and data-driven discovery based on PINNs, Journal of Fluid Mechanics, JFM-23-1205, 2023, Submitted.

As the first author, Elham Kianiharchegani undertook the primary responsibility of drafting the initial versions, performing the analysis, and conceptualizing the problems for each project.

American Physical Society, the publisher of our article, allows for the inclusion of the published article within this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations, Symbols, and Nomenclature

AI	Artificial Intelligence
B-PINNs	Bayesian Physics-Informed Neural Networks
CFL	Courant-Friedrichs-Lewy
CPINN	Conservative Physics-Informed Neural Network
CMAS	Calcium-Magnesium-Aluminosilicate
CNN	Convolutional Neural Network
DPD	Dissipative Particle Dynamics
FNN	Feed-forward Neural Network
GCNs	Graph Convolutional Networks
GRU	Gated Recurrent Unit
HMC	Hamiltonian Monte Carlo
L-BFGS	Limited-Memory Broyden–Fletcher–Goldfarb–Shanno
LSTM	Long Short-Term Memory
MCMC	Markov Chain Monte Carlo
ML	Machine Learning
MLP	Multi-Layer Perceptron
mDPD	Many-body Dissipative Particle Dynamics
MSE	Mean Squared Error
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PINNs	Physics-Informed Neural Networks
POD	Proper Orthogonal Decomposition
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RProp	Resilient Backpropagation
SGD	Stochastic Gradient Descent
SINDy	Sparse Identification of Nonlinear Dynamics
SVD	Singular Value Decomposition
X-PINNs	Xtended Physics-Informed Neural Networks

Chapter 1

Introduction

This thesis explores the benefits, challenges, and potential applications of data-driven approaches in the discovery of partial differential equations (PDEs). Machine learning techniques are employed to unveil the connections between variables and their derivatives. It is important to highlight that the primary objective of this thesis is to leverage machine learning (ML) techniques for the identification of equations using available data. While this thesis provides an overall review of the significance of the chosen equations and the process of data generation, its central focus is on the development and application of ML models and techniques specifically tailored for the discovery of equations.

1.1 Machine learning

Artificial Intelligence (AI) encompasses the ability of a system to effectively comprehend external data, learn from it, and utilize that acquired knowledge to achieve specific goals and tasks. It has garnered significant attention and has been a topic of discussion for many years. However, despite the extensive coverage of AI in recent articles, it is surprisingly challenging to establish a precise definition of what AI entails and what falls outside its scope [13, 14, 15].

By the 1950s, the concept of AI had deeply permeated the intellectual landscape of scientists, mathematicians, and philosophers [13, 16]. Alan Turing stands out as an exemplary fig-

ure who extensively explored the mathematical potential of AI [17, 18, 19]. Turing proposed a thought-provoking idea: if humans can utilize available information and employ reasoning to solve problems and make decisions, why cannot machines do the same? Leveraging his deep engagement with mathematics, Turing delved into the theoretical frontiers of artificial intelligence, yielding remarkable contributions to the field. Alan Turing proposed a definition to determine whether software can be considered intelligent. According to his theory, the intelligence of software can be assessed based on its ability to mimic human cognitive abilities. If a human interacting with the software cannot distinguish it from another human, the software is deemed intelligent. This evaluation method became known as the Turing test.

In 1947, Alan Turing delivered a significant public lecture, which is believed to be the earliest instance of mentioning computer intelligence [20]. In this lecture, he expressed the desire for a machine that could learn from experience, emphasizing the importance of allowing the machine to modify its own instructions. In his 1948 report titled "Intelligent Machinery," Turing introduced key concepts that would become central to the field of AI. These included the hypothesis of intellectual activity as various forms of search, genetic algorithms, and the idea of training artificial neural networks [21]. Several factors prevented Turing from immediately pursuing his ideas. Firstly, the computers of that time lacked a crucial capability for intelligence. They were unable to store commands and could only execute them. In essence, computers were capable of following instructions but could not retain information about their previous actions. Secondly, the cost of computing was exorbitant during the early 1950s.

From 1957 to 1974, the field of AI experienced significant growth. Computers became more powerful, affordable, and accessible, with increased storage capacity. Progress was also made in machine learning algorithms, enabling better problem-solving techniques [22]. In 1970, Marvin Minsky confidently stated that within three to eight years, we would witness the development of a machine possessing the general intelligence of an average human being. However, despite this optimistic outlook, there remained substantial challenges ahead in achieving the ultimate objectives of natural language processing, abstract thinking, and self-

recognition [23].

In the 1980s, AI experienced an expansion of the algorithmic toolkit and increased funding. John Hopfield made notable advancements in the field of neural networks, particularly with the development of the Hopfield network. This network demonstrated the ability to learn and process information in a novel and innovative manner [24, 25, 26]. The Japanese government provided significant funding for AI through its Fifth Generation Computer Project, demonstrating a proactive approach towards advancing the field [14]. During the 1990s and 2000s, significant milestones in artificial intelligence were accomplished, marking the achievement of several key goals in the field [14, 27, 28]. Kohonen (1982) [29] presented a theoretical study and computer simulations showcasing a novel self-organizing process that automatically maps signal representations onto output responses, aligning them topologically with primary events. The process utilizes a network of adaptive physical elements resembling threshold-logic units with short-range lateral feedback, as demonstrated through various computer simulations.

With the growth in computational power and the increasing availability of data, artificial intelligence has witnessed a resurgence of interest, especially in the fields of Machine Learning and Deep Learning. These areas have become prominent focuses within the broader realm of artificial intelligence. The term Machine Learning was introduced by Arthur Samuel in 1959 [30].

The objective of Machine Learning is to develop algorithms that enable computers to learn. It involves designing and implementing techniques that allow computers to automatically analyze and extract patterns or insights from data, and subsequently make predictions, decisions, or take actions based on that learned knowledge [1]. ML utilizes a range of statistical techniques and computational algorithms to uncover relationships and patterns within data. It has made notable progress in diverse fields such as image recognition [31], natural language processing [32], and autonomous vehicles [33]. Its applications encompass automating tasks [33] and facilitating informed decision-making based on patterns that may not be immediately discernible to humans [34]. For example, in visual perception, humans can quickly identify basic

shapes, colors, or objects. In decision-making, individuals can often make judgments based on their immediate observations or experiences while artificial intelligence and automation systems have the potential to uncover complex patterns and insights that may not be immediately apparent to humans [35].

Training ML models is a fundamental aspect of machine learning and presents unique challenges. The process entails providing a dataset to the ML model to facilitate learning and enable accurate predictions or decisions. The dataset comprises input data, commonly referred to as features, which can encompass numerical or categorical variables that capture essential information about the data. The model then examines the patterns, relationships, and dependencies between these features and their corresponding outputs to carry out tasks or make predictions effectively. However, several challenges arise during this training phase. One major challenge is obtaining high-quality and relevant training data. The quality and representativeness of the data greatly impact the performance of the model. Gathering and preprocessing large datasets can be time-consuming and resource-intensive [36]. Another challenge is selecting the appropriate algorithm or model architecture for the specific task [37]. Different algorithms have their strengths and weaknesses, and choosing the right one requires expertise and careful consideration. Additionally, the process of training machine learning models often demands significant computational resources. Training complex models can be computationally intensive and may require specialized hardware or distributed computing setups [34].

This section offers a brief overview of the fundamental types of machine learning, supervised learning, unsupervised learning, and reinforcement learning [1], and their diverse applications in various domains as illustrated in Figure 1.1.

1.2 Supervised learning

Supervised learning is a ML approach where a model learns to map input data to corresponding output values based on labeled training examples. Labelling itself depends on the nature of the

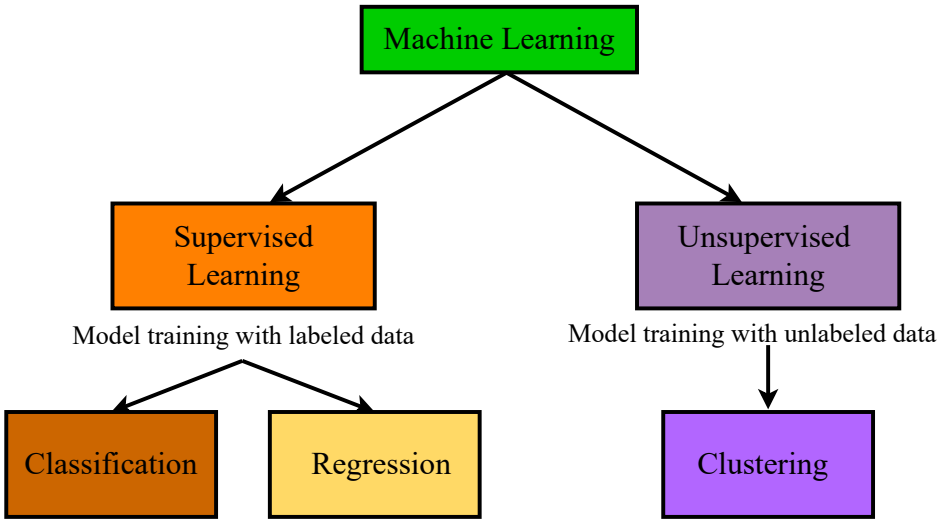


Figure 1.1: Two fundamental types of machine learning [1]. Supervised learning in which ML models are trained on pre-labelled data, consisting of input features and corresponding output labels or target values. Classification and regression are specific types of supervised learning problems, with classification being used for categorical outputs and regression for numerical outputs. Classification involves organizing and categorizing ideas or objects based on their shared characteristics and distinctions while regression models use input data features and associated continuous numeric output values to predict relationships between inputs and desired outputs. Labeled data can vary based on the prediction or classification task. In regression tasks, the labeled data consists of numerical values, while in classification tasks, it involves categorical labels. Unsupervised learning, involves training models on unlabeled data. Clustering is an unsupervised learning technique that groups similar data points together based on their inherent similarities or patterns.

prediction or classification task. In regression tasks, the labeled data includes numerical values, in classification tasks, labeled data can consist of categorical labels. The training data consists of input-output pairs, where the inputs are the features or attributes, and the outputs are the corresponding labels or target values.

The model learns from this labeled data by analyzing the patterns, relationships, and dependencies between the input features and their corresponding outputs. The goal is to learn patterns or relationships from the training dataset and apply them to the test dataset for prediction or classification tasks. Supervised learning can be classified into two types: classification and regression. Each type is designed to address specific problem types based on the characteristics of the output variable [1, 38, 39]. These types are briefly reviewed as follows:

(I) Classification: categorizing and organizing ideas or objects based on their similarities and differences. Classification is the act of recognizing, differentiating, and comprehending various entities. The purpose of classification is to group related facts or entities into distinct classes, allowing for better organization and understanding, see Figure 1.2 (a). It is also a method that brings together similar elements while distinguishing dissimilar ones [1, 40].

(II) Regression: It refers to a class of statistical models that analyze the relationship between a dependent variable and one or more independent variables. The objective of regression analysis is to estimate the relationship between the independent variables, often input data features, and the dependent variable, typically continuous numeric output values. This estimation is used for making predictions or understanding the impact of the independent variables on the dependent variable. Regression models provide a mathematical equation or formula that describes this relationship. These models are used to estimate or forecast values rather than classify them into predefined categories, see Figure 1.2 (b). For instance, regression can be used to predict weather conditions, stock market prices, housing prices, or the demand for a product based on historical data and the relationship between variables [1, 39].

Specific applications of supervised learning include

- Image classification: Supervised learning algorithms can be trained to classify images

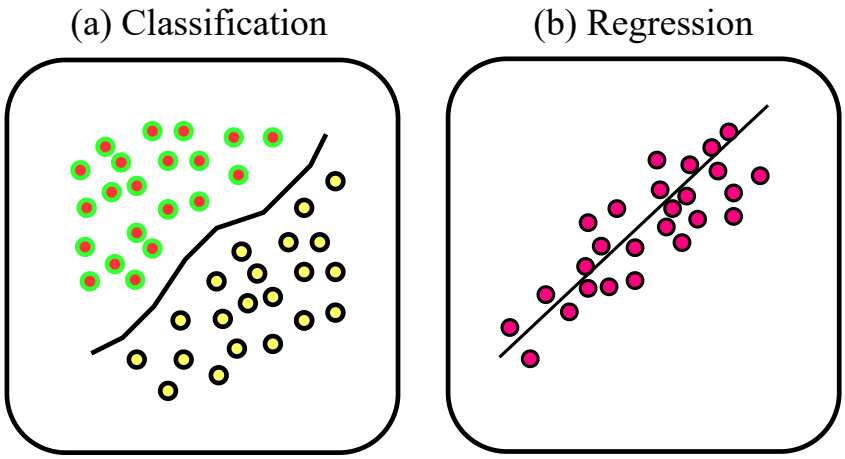


Figure 1.2: Two fundamental techniques in supervised learning, (a) classification and (b) regression. In classification, data is sorted into predetermined classes or labels, allowing for the prediction of future class memberships. Regression, on the other hand, is concerned with predicting continuous numerical values using input variables.

into different categories. This is widely used in applications such as object recognition, facial recognition, and medical image analysis [41].

- Autonomous vehicles: Supervised learning can be used to train models for object detection, lane recognition, and traffic sign classification, contributing to the development of autonomous vehicles [33].

- Sales and demand forecasting: By training on historical sales data along with relevant features such as time, promotions, and competitor information, supervised regression models can predict future sales and demand for products or services for businesses, helping them make informed decisions on inventory management and resource allocation.

1.3 Unsupervised learning

Unsupervised learning is a subset of ML that learns patterns from unlabeled data [42, 43]. The aim to discover hidden structures, clusters, or relationships in the data without explicit knowledge of the ground truth or predefined labels. This approach is particularly useful for organizing unsorted data by identifying similarities, dissimilarities, and hidden patterns in the

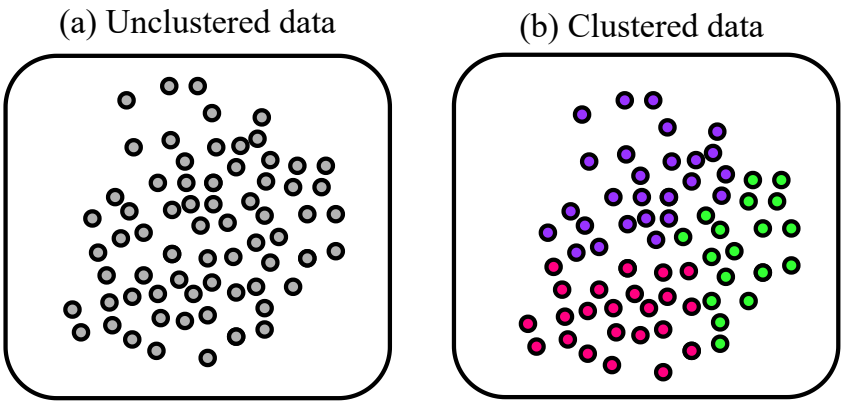


Figure 1.3: Uncluttered data prior to clustering (a) and clustered data after undergoing the clustering process (b).

input data. Unsupervised learning algorithms learn directly from the data without a predefined output variable, allowing them to handle complex tasks. This characteristic makes unsupervised learning particularly useful in scenarios where the data is unlabeled or when we seek to gain insights into the intrinsic patterns and properties of the data itself.

Clustering is often regarded as the most significant unsupervised learning problem. Like other unsupervised learning problems, it revolves around identifying a structure/structres within a set of unlabeled data. A cluster can be defined as a group of objects that exhibit similarity among themselves while being dissimilar to objects belonging to other clusters; clustering is a powerful technique employed to group similar data points together based on their intrinsic similarities or patterns. In the uncluttered data representation (Figure 1.3 (a)), individual data points are scattered without any discernible grouping. However, after applying a clustering algorithm, the data points are grouped into distinct clusters, as depicted in Figure 1.3 (b). In addition to the term data clustering, there are several synonyms used to refer to similar concepts. These synonyms include cluster analysis, automatic classification, numerical taxonomy, botrology, and typological analysis. Clustering algorithms autonomously partition the data into clusters based on various similarity measures or distance metrics. Clustering algorithms are, for example, widely used for the identification of cancerous cells. It divides the cancerous and non-cancerous data sets into different groups [44, 45]. Data

clustering algorithms can be categorized as hierarchical or partitional.

Hierarchical algorithms build clusters in a step-by-step fashion, using previously established clusters [46]. Hierarchical algorithms can be further classified as either agglomerative (bottom-up) or divisive (top-down) approaches. Agglomerative algorithms start by considering each element as a separate cluster [47]. They then iteratively merge these individual clusters to form larger clusters, combining elements based on similarity measures. This process continues until all elements are merged into a single cluster or until a specified termination condition is met. Agglomerative algorithms gradually build clusters from smaller to larger units. In contrast, divisive algorithms begin with the entire dataset as a single cluster. They then proceed to divide this initial cluster into smaller and more distinct clusters through a series of partitioning steps. Divisive algorithms recursively split the dataset into subsets, creating successively smaller clusters until each subset represents a separate cluster or a termination condition is satisfied. Divisive algorithms break down clusters from larger to smaller units.

On the other hand, partitional algorithms determine all clusters at once, without relying on a hierarchical structure [48]. These algorithms assign each data point to a particular cluster based on similarity measures or optimization criteria. Two commonly used heuristic methods for partitioning algorithms are the k-means algorithm [49] and the k-medoids algorithm [50]. The k-means algorithm is designed to divide the data into a pre-defined number of clusters, denoted as k. It begins by randomly selecting k initial cluster centroids from the dataset. Then, in an iterative process, each data point is assigned to the nearest centroid based on a chosen distance metric, typically the Euclidean distance. After all data points are assigned to clusters, the algorithm recalculates the centroids by computing the mean of the data points within each cluster. This new centroid position represents the center of the cluster. The process continues iteratively, with data points being reassigned to the nearest centroids and centroids being recalculated based on the updated cluster assignments. The iterations proceed until convergence is achieved, which happens when the cluster assignments and centroid positions no longer change significantly. The k-medoids algorithm is a variation of the k-means algorithm that addresses

the issue of outliers and robustness. Unlike the k-means algorithm, which uses centroids as representatives of clusters, the k-medoids algorithm employs medoids as representative data points. Medoids are actual data points within the cluster, chosen from the data set. The algorithm starts by randomly selecting k data points as initial medoids. Then, in an iterative process, each data point is reassigned to the nearest medoid based on a chosen distance metric. The medoids are updated by selecting the data point that minimizes the sum of distances to all other points in the same cluster. This process continues until convergence, where the medoid assignments and positions no longer change significantly. The result is a partition of the data into k clusters, with each cluster represented by a medoid. The use of medoids as representatives makes the k-medoids algorithm more robust to outliers and less sensitive to extreme values. By selecting actual data points as medoids, the algorithm can better capture the central tendency and characteristics of each cluster.

Several applications make use of unsupervised learning techniques, including:

- Image and document clustering: Unsupervised learning algorithms can analyze the visual features of images, such as color, texture, shape, or deep learning-based features, to identify similarities and group similar images together. Similarly, they can analyze the textual content of documents, such as words, phrases, or topic distributions, to cluster similar documents together. This enables tasks like organizing large image databases, creating image galleries, or automatically generating tags for images based on their content. Document clustering is particularly useful for tasks like topic modeling, document summarization, content recommendation, and information retrieval systems [51, 52].

1.4 Reinforcement learning

Reinforcement learning enables an agent to learn optimal actions by interacting with an environment and receiving feedback which typically focuses on sequential decision making. It employs a trial-and-error approach, where the agent learns through a feedback-based process.

The agent takes actions in an environment and receives feedback in the form of rewards or penalties based on its actions. The objective of the agent is to maximize the total reward it accumulates over time. By exploring and exploiting the environment, the agent gains knowledge of actions that yield the highest rewards. The agent engages in ongoing interaction with the environment, learning from previous actions and adapting its behavior accordingly. Reinforcement learning encompasses two main types of learning: positive reinforcement learning and negative reinforcement learning [53].

(I) Positive reinforcement learning: It involves rewarding the agent for taking actions that lead to positive outcomes. The agent receives positive reinforcement, typically in the form of rewards, when it makes correct decisions or achieves desired goals. These rewards can be numerical values, such as scores or probabilities, or symbolic values, such as labels or tags. By associating actions with positive rewards, the agent learns to reinforce those actions and increase the likelihood of choosing them in similar situations in the future. Positive reinforcement learning aims to maximize cumulative rewards over time [54].

(II) Negative reinforcement learning: It also known as avoidance learning. In negative reinforcement learning, the agent faces punishment or negative reinforcement for engaging in actions that result in unfavorable outcomes. The punishment can be in the form of penalties, loss of rewards, or negative feedback. The agent learns to avoid actions that result in punishment and instead focuses on actions that lead to positive outcomes or minimize negative consequences. Negative reinforcement learning enables the agent to develop strategies to navigate the environment and avoid undesirable states or outcomes [55].

Both positive and negative reinforcement learning techniques are used in the broader framework of reinforcement learning. The combination of positive reinforcement for desired actions and negative reinforcement for undesired actions helps guide the learning process and shape the agent's behavior towards achieving optimal outcomes.

Reinforcement learning has a wide range of applications across various domains. Here are some notable examples:

- Intelligent robotics: Reinforcement learning plays a significant role in building intelligent robots. By training robots to interact with their environment and receive feedback through rewards or penalties, they can learn to perform complex tasks. This includes tasks such as object manipulation, navigation, grasping, and assembly. Reinforcement learning enables robots to adapt and improve their actions based on the received feedback, leading to more efficient and effective robotic systems [56].

- Personalized treatment plans for patients: In healthcare, reinforcement learning can be utilized to develop personalized treatment plans for patients. By modeling patient responses to different treatments and using reinforcement learning algorithms, medical professionals can optimize treatment decisions. The reinforcement learning agent can learn from patient data, treatment outcomes, and medical guidelines to suggest the most effective treatment options tailored to individual patients' needs [57].

- Autonomous vehicles: Reinforcement learning plays a crucial role in the development of autonomous vehicles. Agents learn to navigate complex traffic scenarios, make decisions about acceleration, braking, and steering, and adapt to changing road conditions. Reinforcement learning enables autonomous vehicles to learn safe and efficient driving behaviors, improving road safety and optimizing transportation efficiency [58].

- Game playing: Reinforcement learning has been successful in developing game-playing agents that can compete with and even surpass human performance in various games. Examples include AlphaGo [59], which defeated world champion Go players, and Dota 2 [60], which achieved high-level gameplay. Reinforcement learning agents learn optimal strategies by playing against themselves or human opponents, honing their skills through continuous learning and exploration.

As computers have advanced, generating large volumes of data has become increasingly common in various domains. This exponential growth in data has created a pressing need for models and algorithms capable of effectively handling and analyzing such vast datasets. How-

ever, analyzing complex data often involves a large number of variables, which can lead to challenges such as increased memory and computation requirements. This is particularly true in high-dimensional datasets where the data becomes more sparse, making it harder to extract meaningful insights and reducing the accuracy of predictions. To address these challenges, dimensionality reduction techniques are employed to transform high-dimensional data into a lower-dimensional representation that captures the underlying characteristics of the data [61]. Dimensionality reduction serves multiple purposes, including noise reduction, data visualization, cluster analysis, and supporting other analytical tasks. There are two main approaches in dimensionality reduction, feature selection [62], which identifies the most relevant input variables for a given task, and feature extraction [63], which creates new derived features while reducing the dimensionality of the data. Feature extraction plays a critical role in building effective machine learning models by identifying informative aspects of the data and discarding irrelevant or redundant information, ultimately improving data analysis, model performance, and decision-making based on the extracted features.

High dimensionality of data presents challenges for neural networks, including increased computational complexity, longer optimization steps, and the requirement for larger network architectures [64]. To tackle the challenges posed by high-dimensional data and enhance the efficiency of training neural networks, it is essential to make advancements in dimensional reduction techniques. Dimensional reduction techniques play a critical role in enhancing the performance and effectiveness of neural network models, enabling them to handle high-dimensional data more effectively.

In the following sections of this chapter, we will present a detailed overview of neural network architectures. We will explore essential components such as loss functions, which measure the difference between predicted and actual values, and activation functions, which introduce non-linearity and enable the network to learn intricate data relationships. Grasping these foundational elements is crucial for a deeper understanding of neural network operations and their applications in machine learning.

1.5 Neural networks

ML is a comprehensive concept within the field of AI, encompassing algorithms that have the ability to learn and adapt from data. Neural networks, on the other hand, are a type of ML models that are designed inspired by the connections of neurons in the human brain. They are composed of interconnected nodes known as neurons, organized into layers, which enable them to learn and adapt based on new information. Through the intricate network of neurons, information is processed and transmitted, enabling the neural network to make accurate predictions or decisions.

Training is a crucial phase for neural networks, during which the connections between neurons, referred to as weights, are continually adjusted to optimize the network's performance. Once the training phase is complete and the weights have been optimized, the network enters the prediction phase. In this phase, the weights are typically fixed and no longer adjusted. Through this iterative process, the neural network becomes increasingly proficient at recognizing and generalizing patterns in the input data. The network takes new input data and uses the learned weights to generate predictions or outputs based on the patterns and relationships it has learned during training. The fixed weights enable the network to efficiently and quickly process new data without the need for further training. Inspired by the remarkable parallel structure of the human brain, neural networks have emerged as powerful tools for various applications. Their ability to harness parallelism and adaptability has led to significant advancements in fields such as image recognition [65], natural language processing [66], and predictive modeling [67, 68].

Deep neural networks are a specific type of neural network that consists of multiple hidden layers between the input and output layers. Generally, layers in a neural network that are not directly connected to the input or output are commonly referred to as hidden layers. Each layer of the network learns progressively more abstract and complex features based on the previous layer's output. This allows deep learning models to capture intricate patterns and dependencies in the data. One significant advantage of deep learning is its ability to analyze

and learn from vast quantities of unsupervised data. This makes it a valuable tool for Big Data Analytics, particularly in scenarios where the raw data is predominantly unlabeled and lacks categorization [69]. The rise in popularity of deep learning is attributed to the increasing availability of vast amounts of data and advancements in hardware, providing powerful computing capabilities. Deep neural networks have achieved significant success in various branches of ML, including supervised learning and unsupervised learning [69]. Deep learning finds extensive applications across diverse domains, notably making significant contributions in the field of computer vision. Deep learning models have demonstrated impressive performance image classification [41] and image segmentation [70]. These models can accurately classify objects in images, detect their presence and location, and segment different regions within an image. Additionally, it has emerged as a powerful tool in the field of medical image analysis [71]. Deep learning has been employed to build regression models for predicting time-dependent data, such as road traffic speed forecasting [72]. Additionally, deep learning techniques offer a comprehensive introduction to their application in the field of Natural Language Processing tasks [66].

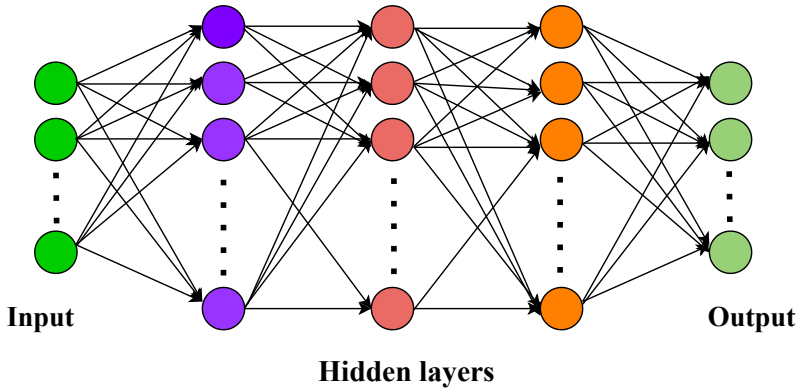


Figure 1.4: A deep neural network architecture which has an input layer, three hidden layer, and an output layer. Each layer consists of multiple neurons. The neurons in each layer are interconnected, with weights assigned to each connection. The input layer receives the input data, and the output layer produces the final output of the network, with the weights between neurons adjusted during training using an optimization algorithm.

To gain a comprehensive understanding of neural networks, it is essential to explore key concepts that significantly impact their functionality. One such concept is hyperparameters in

neural networks [73]. Hyperparameters are parameters that are set prior to the training process and influence the behavior and performance of the network. Hyperparameters in neural networks include the number of hidden layers determines the depth and complexity of the network and the number of neurons in each hidden layer that affects the network's capacity to learn complex patterns. The learning rate, which controls the step size for parameter updates during training, as well as the selection of activation functions, are important hyperparameters to consider in the network's configuration. The process of selecting and tuning hyperparameters plays a crucial role in determining the performance and generalization ability of a model. It typically involves an iterative trial-and-error approach, where different combinations of hyperparameters are tested and evaluated. Techniques like cross-validation are commonly employed to assess the model's performance. Gaining a comprehensive understanding of these fundamental concepts provides valuable insights into the inner workings of neural networks and how they are optimized during the training process.

1.5.1 Activation function

An activation function in a neural network determines whether a neuron should be activated or not based on the input it receives [74]. It plays a crucial role in computing the output of a neuron by applying mathematical operations to the input values. One key requirement is differentiability, as most neural network training algorithms rely on derivatives for weight updates during backpropagation. Activation functions impact the occurrence of the vanishing/exploding gradient problem during training [75]. The vanishing gradient problem arises when gradients become too small, hindering effective learning in deep networks. Conversely, the exploding gradient problem occurs when gradients become excessively large, causing unstable training. The activation function takes the weighted sum of the input values and applies a transformation to produce the output of the neuron. This transformation introduces non-linearity to the network, allowing it to model complex relationships and capture non-linear patterns in the data. By determining the importance of the neuron's input in the prediction

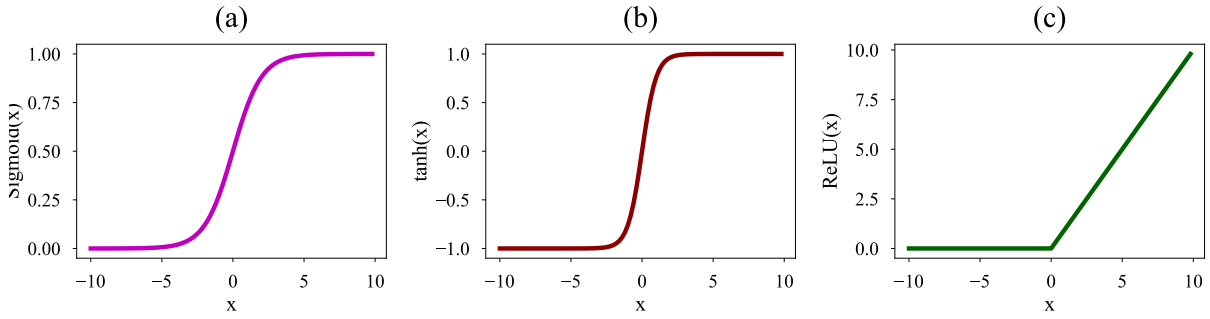


Figure 1.5: Three most frequently employed activation functions in neural networks (a) sigmoid, (b) tanh , and (c)ReLU.

process, the activation function helps the network make more accurate predictions and learn meaningful representations from the data [76].

As mentioned, the choice of activation function has a significant impact on the training performance of a neural network. There are several commonly used activation functions, including sigmoid [77], tangent hyperbolic (tanh) [78], and ReLU (Rectified Linear Unit) [79]. Each of these activation functions has its own characteristics and affects the behavior of the neural network in different ways. They are defined as follows:

- Sigmoid (Figure 1.5 (a)) : The sigmoid activation function, also called the logistic function, has the following mathematical form

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{1.1}$$

  It takes a real value and squashes it between 0 and 1, offering a smooth and continuous output. It is commonly used in binary classification tasks where the goal is to predict a probability or make a decision based on a threshold. The sigmoid function possesses several distinct properties that make it advantageous for use in neural networks. One of the most important property of the sigmoid function is its differentiability. The sigmoid function is continuously differentiable, meaning that its derivative can be easily calculated at any point. This property is essential for the backpropagation algorithm, which is used to train neural networks. The term sigmoid refers to the S-shaped curve, and the

logistic form of the sigmoid function maps the range $(-\infty, \infty)$ onto $(0, 1)$. This property allows the sigmoid function to transform any input, whether positive or negative, into a probability-like output, indicating the likelihood or activation level of a particular event or class. This characteristic is particularly useful in classification tasks where the network needs to assign probabilities to different categories.

- tanh (Figure 1.5 (b)): tanh function maps the input to the range $(-1, 1)$ and exhibits similar properties to the sigmoid function. This non linear function has the following mathematical form

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \tag{1.2}$$

- ReLU (Figure 1.5 (c)): ReLU is a non-linear activation function which has the following mathematical form

$$y = \max(0, x). \tag{1.3}$$

Compared to the sigmoid and tanh functions, it has become popular in recent years due to its computational efficiency and ability to mitigate the vanishing gradient problem. Firstly, ReLU is computationally efficient, making it well-suited for large-scale neural networks and deep learning models. Secondly, ReLU does not suffer from either vanishing or exploding gradients. One significant advantage of the rectifier function ReLU is its ability to produce a true zero output. This distinguishes it from activation functions such as tanh and sigmoid, which approximate a zero output but never precisely reach a true zero value. The ability of ReLU to generate true zero values for negative inputs, is a valuable property in neural networks. With some neurons having zero outputs, the network can focus on learning the important features and disregard irrelevant or noisy information. This helps streamline the learning process and improve efficiency [80].

In addition to the aforementioned commonly used activation functions, a relatively new activation function called Swish was introduced in 2017 [74]. Swish has gained attention for its potential performance improvements compared to other activation functions like ReLU. It is

expressed mathematically as:

$$f(x) = x/(1 + exp(-\beta x)) = x\sigma(\beta x), \tag{1.4}$$

where β is a trainable parameter. By combining the linearity of the input with the non-linear characteristics of the sigmoid function, Swish offers enhanced expressiveness in capturing complex relationships within the data.

1.5.2 Loss function

Loss functions, also known as cost functions, play a vital role in machine learning algorithms, including neural networks. They measure the difference between the predicted output of a model and the actual target output for a given input. The purpose of a loss function is to evaluate how well an algorithm models a dataset. The learning algorithm aims to minimize the loss function value by adjusting the weights and biases of the network through optimization. The loss function aggregates all possible network variables into a single value, indicating how far the system is from achieving perfect performance. Initially, the loss value is high when the network has poor accuracy, but it decreases gradually as the system learns.

The choice of loss function depends on the specific problem being solved. There exists a variety of loss functions, each suitable for different types of tasks. For instance, mean squared error (MSE) is a commonly employed loss function for regression tasks [81]. It can be defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{i=N}(\hat{y}_i - y_i)^2, \tag{1.5}$$

where $\hat{y}$ represents the predicted value, y denotes the target value, and N is the number of samples in the dataset. The MSE is widely used due to its differentiability, which allows for efficient optimization using gradient descent-based algorithms [82]. Moreover, it is straightforward to interpret as it provides a measure of the average deviation between predictions and actual values. One downside of MSE is that it can suffer from slow learning speed or slow convergence

when used in combination with the sigmoid activation function. The sigmoid function has a saturation property where its gradient becomes small for large input values, leading to vanishing gradient problem. This can hinder the learning process and result in slower convergence when optimizing the model using MSE with sigmoid activation [83].

The choice of a loss function is a critical decision in machine learning as it can significantly impact the performance of the network. Using an inappropriate loss function can lead to unintended effects on the network's learning process. The loss function represents the aspects of the problem that the network considers important, and minimizing its value is the goal of the optimization process. In this study, we have selected MSE (1.5) as our loss function for all networks.

1.5.3 Learning rate

The learning rate is a crucial hyperparameter in ML algorithms [84]. A hyperparameter refers to a parameter that is not learned from the training data but is set manually before the learning process begins. Unlike the model's internal parameters that are learned through training, hyperparameters control the behavior and performance of the learning algorithm. The learning rate is a hyperparameter that plays a crucial role in determining the performance and generalization ability of a model. It determines the step size at which the weights and biases of a neural network are updated during the training process. The learning rate is typically selected within the range of 0.0 to 1.0. It determines how quickly or slowly the network adapts to the training data. A learning rate that is too small can lead to slow convergence of the training algorithm, while a learning rate that is too large can cause the algorithm to diverge [85]. Therefore, choosing an appropriate learning rate is essential for achieving good performance and efficient training. A suitable learning rate allows the network to effectively adapt to the training data, achieving optimal performance. However, selecting a learning rate that is too small can result in slow convergence of the training algorithm, while a learning rate that is too large can cause the algorithm to diverge. Finding the right balance is essential for successful training and achieving

accurate predictions. It is important to note that the optimal learning rate can vary depending on the dataset, model architecture, and specific task at hand.

1.5.4 The general architecture and training of deep neural networks

The general architecture of a deep neural network, illustrated in Figure 1.4, is comprised of interconnected neurons arranged in multiple layers. Each neuron in a deep neural network is connected to neurons in the previous layer and the subsequent layer. Typically, the architecture includes an input layer, one or more hidden layers, and an output layer. The input layer receives input data, which flows through the network in a forward direction to the output layer. The output layer delivers the network's final output, which may be classification, regression, or another type of prediction depending on the specific problem being addressed. The selection of architecture and optimization algorithm can significantly affect the network's performance and accuracy.

In a neural network, the weights and biases are parameters that allow the network to learn from input data and make accurate predictions. The weights represent the strength of the connections between neurons in different layers of the network. Each connection has an associated weight, which is multiplied by the input from the previous layer and passed through an activation function to produce the output of the neuron. The weights are initially set to random values and then adjusted during the training process to minimize the error between the predicted output and the actual output.

The biases, on the other hand, are added to the weighted sum of the inputs before being passed through the activation function. The biases are also adjusted during the training process to improve the accuracy of the predictions. The bias in a neural network helps the model make predictions more accurately by adjusting the output of each neuron. It acts like a constant value that is added to the overall calculation performed by the neuron. By adjusting the bias, we can shift the activation function used by the neuron towards either the positive or negative side. This shift allows the model to make predictions that are more in line with what we expect. The

general formula for a simple deep neural network with weights and biases can be expressed as

$$y = f(W_m...f(W_3f(W_2f(W_1x + b_1) + b_2) + b_3) + ...b_m),$$

where x is the input to the network, W_i and b_i are the weights and biases of the i-th layer, respectively. Moreover, f is the activation function and y is the output. The number of hidden layers in the network is denoted by m. The output of each layer is calculated by applying the activation function f to the linear combination of the input vector and the corresponding weights and biases of that layer. The output of the final layer gives the prediction or classification of the network.

In the field of neural networks, it is widely acknowledged that backpropagation is the dominant technique used for training neural networks. This algorithm, originally introduced by Frank Rosenblatt in 1962 [86], has become widely adopted in machine learning. Backpropagation is an algorithm that allows the network to adjust its weights and biases by iteratively propagating the errors backward from the output layer to the input layer. This process helps the network learn and improve its performance by minimizing the difference between the predicted and actual outputs.

1.6 Differential equations

Differential equations are mathematical equations that involve derivatives and are used to describe the relationships between variables and their rates of change. They have widespread applications across various fields like physics, engineering, economics, and biology, enabling the modeling and analysis of dynamic systems [87]. In this section, we will focus on discussing two common types of differential equations, ordinary differential equations (ODEs) [88] and partial differential equations (PDEs) [89].

1.6.1 Ordinary differential equations

Ordinary Differential Equation (ODE) [88] is a type of mathematical equation that describes the relationship between a function and its derivatives. In an ODE, the unknown function depends on a single independent variable and its derivatives with respect to that variable. The equation typically involves derivatives of different orders and may include the function itself. The general form of an ODE is given by

$$F\left(x, u, \frac{\partial u}{\partial x}, \frac{\partial^2 u}{\partial x^2}, ..., \frac{\partial^n u}{\partial x^n}\right) = 0, \tag{1.6}$$

where x is the independent variable, u is the dependent variable, and $\frac{\partial u}{\partial x}$ and $\frac{\partial^2 u}{\partial x^2}$ represent the first and second derivative of u with respect to x, and so on, up to the nth derivative $\frac{\partial^n u}{\partial x^n}$. The function F establishes the relationship between the variables x, u, and their derivatives.

ODEs are employed to model a wide range of physical phenomena as they offer a robust mathematical framework to describe the changes in quantities relative to an independent variable, usually time [90]. In addition to their significance in understanding biological phenomena [91], ODEs are also crucial in engineering applications, including motion, population dynamics [92], chemical reactions [93], and electrical circuits [94].

1.6.2 Partial differential equations

Partial Differential Equations (PDEs) are mathematical models that involve partial derivatives of an unknown function with respect to multiple independent variables [89]. They find broad applications in various scientific and engineering disciplines for describing a wide range of phenomena. PDEs are commonly encountered in fields such as fluid dynamics [95], electromagnetism [96], and finance [97], where they play a crucial role in understanding and predicting complex behaviors.

The general form of a partial differential equation is

$$F\left(x_1, x_2, x_3, ..., x_n, u, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, ..., \frac{\partial^2 u}{\partial x_1 x_2}, ..., \frac{\partial^n u}{\partial x_1 x_2, ..., x_n}\right) = 0, \tag{1.7}$$

where $u(x_1, x_2, ..., x_n)$ is the unknown function, and F is some function of the independent variables $x_1, x_2, ..., x_n$ and the partial derivatives of u. The order of a partial differential equation is the order of the highest partial derivative appearing in the equation.

Solving an ODE involves finding a function that satisfies the equation for a given set of boundary conditions, which specify the behavior of the solution at the boundaries of the domain of interest [88]. Both numerical methods and ML methods have their advantages and limitations when it comes to solving ODEs and PDEs. Numerical methods such as separation of variables [98], finite difference methods [99, 100], finite element methods [101], spectral methods [102], and Runge-Kutta methods [103] are widely used for solving ODEs and PDEs. On the other hand, ML methods have gained attention for solving ODEs/PDEs as well. The integration of ODEs/PDEs with ML techniques has ushered in new opportunities for modeling and comprehending intricate physical, chemical, and biological systems. This integration can be broadly classified into two domains, allowing for:

- Solving ODEs/PDEs: ML has shown great potential in solving ODEs/PDEs by offering new approaches to overcome the challenges associated with traditional numerical methods[99, 101, 103, 104, 105]. ML techniques can handle complex geometries, high-dimensional systems, and computationally expensive simulations more effectively. ML techniques, including Gaussian Processes [106, 107], Neural Networks [108, 109, 110, 111, 112, 113], and Physics-Informed Neural Networks [114] have demonstrated their effectiveness in solving PDEs.

- Predicting Unknown ODEs/PDEs: discovery of ODEs/PDEs using ML methods is an exciting area of research that aims to automatically uncover the underlying equations governing a system from available data. ML algorithms can be trained to discover

ODEs/PDEs by learning the functional relationships between the variables in the data. These algorithms can identify patterns, correlations, and dependencies within the data, enabling the extraction of the underlying PDEs. Discovering ODEs/PDEs through ML involves tasks such as identifying the form of the ODEs/PDEs, estimating unknown parameters, and discovering new terms of the equation that improve the fit to the data [115, 116, 117].

The main objective of the studies presented in this thesis is to discover PDEs using ML methods. The focus is on employing ML algorithms to analyze available data and uncover the underlying equations that govern the system of interest. The primary objective of this thesis is to enhance our comprehension of the phenomena governed by the PDEs and make significant contributions to the field of PDE modeling and analysis.

1.7 Data-driven discovery of PDEs

Data-driven discovery of PDEs extracts mathematical models from observational or experimental data to describe complex systems [114, 117, 118, 119]. It aims to identify the underlying PDEs and reveal their physics without prior knowledge of the equations. This approach enables exploration and understanding of complex phenomena solely based on available data, opening up new possibilities for modeling diverse systems. It has shown promising outcomes in fields such as fluid dynamics, materials science, and neuroscience, leading to accelerated discoveries and improved understanding. By extracting meaningful insights, data-driven PDE discovery contributes to advancements and comprehension of complex phenomena across disciplines. The ability to predict PDEs from data opens up new possibilities for modeling and analyzing complex systems. It enables researchers to gain insights into the underlying physical processes and improve our understanding of the system's behavior.

The process of discovering PDEs from data involves training an ML model to identify a

function F, which satisfies the PDE

$$u_t = F(u, u_x, u_{xx}, ...), \qquad x \in \Omega \subset \mathbb{R}^n, \tag{1.8}$$

where t belongs to the time interval $[0, T]$, u is a series of measurements of certain physical quantities on the domain $\Omega \subset \mathbb{R}^n$, u_x and u_{xx} represent the first- and second-order spatial derivatives of u, and u_t denotes the time derivative. During the training process, the ML model is fed with labeled data, where the input data is known, and the corresponding output data is provided. In certain models, the input data includes spatial and temporal variables, and the output data represents the solution of the PDEs. The model captures patterns and relationships between the input data and the corresponding output to discover the function F that appears on the right-hand side of Equation (1.8). However, in some models, the input data includes spatial derivatives of the solution, such as u_x and u_{xx} and the corresponding output represents the time derivative, u_t. In these cases, the model learns the relationships between the derivatives to uncover the correct form of the function F in Equation (1.8). The training process involves feeding the model with labeled data, where the input data is known, and the corresponding output data is provided. In some models, the input data includes spatial and temporal variables, and the corresponding output data represents the solution of the PDEs. The model learns from this labeled data by adjusting its internal parameters. However, in some models, the input data includes spatial derivatives of the solution and corresponding output is time derivative. The model learns patterns between derivatives to discover right hand side of the Equation (1.8).

Once the model is trained, it can be used to identify the underlying PDEs that govern the system's dynamics. The ML model can capture the spatial and temporal dependencies present in the data, allowing it to infer the underlying PDEs based on the learned patterns. This approach is particularly useful when the explicit form of the PDEs is unknown or difficult to derive analytically.

Various methods have been proposed to tackle the task of predicting PDEs from data. PDE-

Net [120, 121], in which the primary goal is to learn the form of the non-linear function F such that

$$u_t = F(x, u, \nabla u, \nabla^2 u, ...),$$

where $x \in \Omega \subset \mathbb{R}^2$ and $t \in [0, T]$, and u_t is derivative of u with respect to time t. The PDE-Net approach offers a notable advantage by reducing the reliance on extensive prior knowledge regarding the specific form of the non-linear response function F, making it applicable to a wide range of complex PDE problems. This simplifies the modeling process and eliminates the need for specific operator approximations. However, PDE-Net is computationally demanding, especially for large-scale or high-dimensional PDE problems. Training and inference times may increase significantly, requiring adequate computational resources for efficient implementation. Additionally, the performance of PDE-Net is highly dependent on the availability of abundant and representative training data. In cases where data is limited or of poor quality, the network may encounter challenges in accurately learning the underlying dynamics of the PDE.

Neural networks [122] and Gaussian processes [117] are two other often used methods for discovering equations. Neural networks offer flexibility in discovering equations by approximating complex relationships, while Gaussian processes provide a probabilistic approach with uncertainty quantification. Sparse Identification of Non-linear Dynamics (SINDy) [123, 124] is a data-driven technique that uncovers the governing equations and dynamics of a system from noisy and high-dimensional data. The SINDy algorithm employs sparse regression to estimate coefficients in governing equations. It constructs a library of candidate functions, such as polynomials and trigonometric functions, and identifies the most relevant terms and their coefficients. SINDy has found success in various scientific domains, however, the choice of candidate functions is critical, requiring expertise specific to the problem at hand. Incorrect choices may lead to inaccurate equation discovery [123].

Physics-Informed Neural Networks (PINNs), on the other hand, have emerged as a powerful technique that combines the flexibility of neural networks with the incorporation of physical constraints. By integrating known physics principles into the learning process, PINNs enhance

the accuracy and generalizability of the predicted PDEs [114, 125]. By integrating physics-based constraints into the loss function, PINNs enable the direct learning of the underlying equations governing a system from available data.

Conservative Physics-Informed Neural Network (cPINN) [126] and eXtended Physics-Informed Neural Networks (X-PINNs) [127, 128] are valuable approaches for equation discovery, with their specific advantages and considerations. cPINN is a spatial domain decomposition approach within the PINN framework specifically designed for conservation laws. It focuses on enforcing conservation principles in the solution of PDEs. cPINN decomposes the computational domain into smaller subdomains and trains separate neural networks for each subdomain. The solutions from individual subdomains are then combined to ensure overall conservation. X-PINNs, on the other hand, generalized space-time domain decomposition approach for PINNs framework. It enables the deployment of multiple neural networks in smaller subdomains, enhancing representation and parallelization capabilities. Unlike cPINN, X-PINN is applicable to any type of PDE and allows arbitrary decomposition in space and time.

The utilization of the residual network (ResNet) aims to reveal hidden parameterized dynamical systems through the utilization of observational data concerning state variables in [129]. Time-series or spatio-temporal datasets can be employed to identify accurate governing systems using utilizing neural networks for ordinary and partial differential equations (NeuPDE), as discussed in [130]. This model is parameterized using both shallow multilayer perceptrons and nonlinear differential terms, allowing it to capture relevant correlations among spatio-temporal samples. A network-based approximation to an ODE/PDE was constructed, considering both the interconnections among components (via a dictionary of monomials) and the differential characteristics of spatial terms (through finite difference kernels). A framework named DLGA-PDE, which merges deep learning and genetic algorithms, has been introduced for the discovery of PDEs, as discussed in [131].

1.8 Coarse-graining: bridging the gap between microscopic and macroscopic properties

This section introduces the basic idea of coarse-graining, and the concepts of microscopic and macroscopic properties. In this thesis, the focus is not on the simulation methods, but simulations are used simply as the source of data. This data is then used for discovering the underlying PDEs or ODEs from the data using ML methods. Figure 1.6 shows a sketch of different common simulation methods (the boxes), typical time scales, and the typical entities or/and particle numbers (ovals and spheres) used in them. In this thesis, phase-field models and coarse-grained MD were used for data generation and the specific methods used are described in Chapter 3.

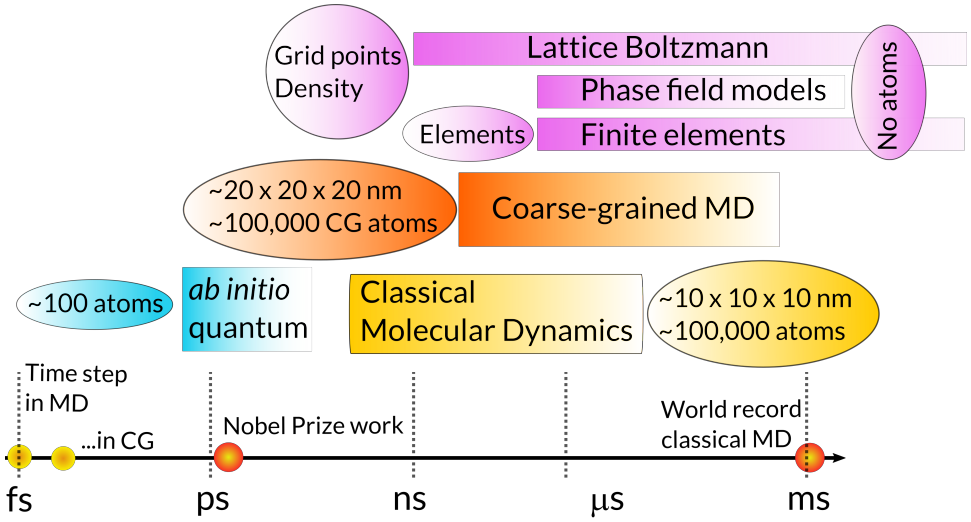


Figure 1.6: Typical time scales and simulation modelling methods, fs=femtosecond (10^{-15} s), ps=picosecond (10^{-12} s), ns=nanosecond (10^{-9} s), μs=microsecond (10^{-6} s), and ms=millisecond (10^{-3} s). MD stands for molecular dynamics and CG for coarse-grained. The rectangular boxes list the methods while the ovals and spheres provide typical systems sizes and basic entities (that is, whether the method uses atoms, elements or such) used by the method. The current longest MD simulations can reach about a millisecond on specialized hardware [2]. The work that lead to the 2013 Nobel Prize in Chemistry for multiscale modeling is discussed in detail in a pre-Nobel Prize article by Karplus [3]. The relations between the different methods are discussed in detail by Murtola et al. [4].

Microscopic properties refer to the characteristics and behaviors of a material at the atomic

or molecular level. These properties encompass various attributes, including the positions and velocities of particles, their energies, and the interactions between them. Examples include atomic arrangements, chemical bonding, intermolecular forces, electronic structure, and vibrational modes. An understanding of these microscopic properties is crucial for predicting and explaining the macroscopic behavior and properties of the material [132].

On the other hand, the macroscopic properties refer to the observable behaviors and characteristics of a material at a larger scale. These properties can include mechanical behavior (e.g., elasticity, strength), thermal behavior (e.g., temperature, conductivity, expansion), electrical behavior (e.g., conductivity, resistivity), and other relevant properties that are measurable at a macroscopic level. They are influenced by the underlying microscopic properties and interactions within the material but are generally expressed in terms of averaged or bulk properties that are relevant to practical applications and experimental measurements [132, 133].

Coarse-graining techniques can be used to bridge the gap between microscopic and macroscopic by providing simplified descriptions that capture the essential behavior at larger scales. Coarse-graining plays a crucial role in reducing the computational complexity of simulations in modeling physical systems, enabling studies of larger systems or longer timescales than would be possible with microscopic simulations alone. It simplifies complex systems by reducing the number of degrees of freedom. This involves averaging or integrating over the small-scale details such as the positions and velocities of individual atoms, to derive a simplified description of its behavior [4, 134, 135, 136, 137, 138, 139, 140]. Multiscale modeling was recognized with the 2013 Nobel Prize in Chemistry.

Three main categories commonly used to classify different levels of scale are macroscale, mesoscale, and atomistic. Macroscale refers to time and length scales observable by the naked eye consisting typical times $> 0.1\,\mathrm{s}$ and typical lengths $> 1\,\mathrm{mm}$. At the macroscopic level, the main simulation methods commonly employed include phase-field models [141], finite element method [101], and Monte Carlo simulations [142]. The mesoscale represents intermediate scales not directly observable by the naked eye but accessible through various exper-

imental techniques, typical time scales range from 10^{-7} s up to 10^{-1} s, while lengths range from micrometers to millimeters. At the mesoscale, commonly used simulation methods include phase-field models, lattice Boltzman [143], coarse-grained molecular dynamics [144], and Monte Carlo simulations.

The atomistic scale refers to the molecular scale, where phenomena involve the interactions and behaviors of individual atoms or molecules. The typical time scales range from picoseconds (10^{-12} s) to a few hundreds of nanoseconds (10^{-7} s), while lengths range from Ångströms to a few tens of nanometers (10^{-10} - 10^{-8} m). This scale focuses on phenomena involving microscopic mechanisms and interactions, such as hydrogen bonding. Commonly used simulation methods at the atomistic scale include classical molecular dynamics [3] and Monte Carlo simulations.

1.9 Thesis outline

In this dissertation, we focus on ML-based approaches for discovering coarse-level equations from data. We aim to extract meaningful relationships and patterns from the given data to derive simplified equations that capture the essential dynamics of the system. Our efforts are primarily dedicated to three key objectives: discovering equations through the utilization of black-box learning [145], gray-box learning [146], and white-box [147] learning approaches. Black-box learning involves training models without any prior knowledge about the underlying system. Gray-box learning, on the other hand, combines limited prior knowledge with data-driven learning.

In the case of black-box models, our focus is on learning the relationships between the inputs and the outputs without explicitly representing the equations symbolically. We investigate two specific scenarios to tackle this challenge. In the first scenario, we encounter an unknown relation between field evolution and its spatial derivatives. To address this, we employ two different approaches. The first involves utilizing a multi-layer perceptron (MLP) to discover

the underlying equation. In the second approach, we combine a convolutional neural network (CNN) with long short-term memory (LSTM) to uncover the equation. The CNN-LSTM architecture proves beneficial in modeling the temporal dependencies and capturing the complex relationships between the field evolution and its spatial derivatives. In the second scenario, we encounter a situation where the spatial derivatives, their orders, and combinations are unknown. To address this challenge, we employ a CNN-based approach. The CNN proved effective in capturing the complex patterns and dependencies present in the data, allowing us to uncover the underlying equation without prior knowledge of the derivatives and their combinations. This approach proved especially valuable when dealing with complex and non-linear systems, where deriving the underlying equations may be difficult or even unknown.

Explicit equation discovery approaches have previously been explored in addition to black-box learning. Techniques such as PINNs [114] and X-PINN [127, 128] have been developed to explicitly uncover the underlying equations governing a physical system. In this thesis, we have developed a framework that combines X-PINNs with data-driven methods to uncover the non-linear term in a partially known PDE, incorporating a Laplacian term as a diffusion operator. This approach provides an effective solution for gray-box learning of equations using X-PINNs, where equations are discovered with limited prior knowledge. Gray-box learning involves uncovering equations when only partial knowledge is available [148, 149].

Moreover, to discover an equation using white-box learning, we utilized PINNs to investigate the behavior of calcium-magnesium-aluminosilicate (CMAS) droplet spreading dynamics. By leveraging the parameter discovery capability of PINNs, we uncovered the unknown parameters of the proposed ordinary differential equation (ODE) that effectively captures the droplet spreading behavior.

The remainder of this thesis is organized as follows:

Chapter 2 provides an overview of the neural network methods employed in this study. Section 2.1 covers the feed-forward neural network and multilayer perceptron (MLP). Section 2.2 discusses the Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM). Sec-

tion 2.3 focuses on Convolutional Neural Networks (CNNs) and their architecture. The application of PINNs and X-PINNs is explored in Section 2.4. Finally, Section 2.5 delves into the use of symbolic regression.

Chapter 4, machine learning based data-driven discovery of non-linear phase-field dynamics, which includes an introduction to the subject in Section 4.1. A brief summary of the phase-field approach and data preparation is presented in Section 4.2. In Section 4.3, an overview of MLP and CNN-LSTM networks. Finally, in Section 4.4, we introduce a CNN network that learns PDEs without any assumption regarding spatial derivatives.

Chapter 5, A Framework Based on Symbolic Regression Coupled with eXtended Physics-Informed Neural Networks for Gray-Box Learning of Equations of Motion from Data, which includes an introduction to the subject in Section 5.1. Section 5.2 provides an overview of the phase-field approach and data preparation. Section 5.3 gives a brief summary of the PINNs and X-PINNs. Section 5.4 presents the symbolic regression results. The performance of the framework for noisy data is discussed in Section 5.5. In Section 5.6, we examine the framework's performance for different sizes of training data sets to investigate the amount of optimal data required for training. Finally, a summary of the chapter is provided in Section 5.7.

Chapter 6, Characterization of partial wetting by multiphase many-body dissipative particle dynamics and data-driven discovery based on Physics-Informed Neural Networks, which includes an introduction to the subject in Section 6.1. In Section 6.2, we provide an overview of the mDPD simulation parameters and system setup. The simulation results and the process of data preparation are presented in Section 6.3. Section 6.4 gives a brief introduction to the PINNs architecture, followed by a presentation of the results of PINNs and parameter discovery. The symbolic regression results are outlined in Section 6.5. Section 6.6 covers the discussion on B-PINNs. Finally, we conclude with a summary of the chapter in Section 6.7.

Chapter 7 summarizes the study's major conclusions, highlights key findings, and provides a comprehensive discussion on potential future research directions.

Chapter 2

Neural network models: An overview

In this chapter, we provide a comprehensive review of the deep learning methods employed in this thesis. It is important to highlight that all the methods utilized in the thesis research are supervised learning models, which involve training the models using labeled data. Throughout the thesis, the power of deep learning techniques for the discovery of PDEs is harnessed. To accomplish this, we employed various deep learning models, including multi-layer perceptrons (MLPs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs).

Using neural networks for data-driven discovery of PDEs has become a powerful and effective approach [114, 117]. The ability of neural networks to learn complex patterns and relationships within the data enables the discovery of hidden physical laws and its governing equations. This approach offers several advantages, including the potential to uncover novel insights, accelerate the discovery process, and enhance the accuracy of the physical PDE models.

In contemporary neural network design, one of the critical challenges is determining the most suitable network size for a particular application. The concept of network size encompasses multiple factors within layered neural network architectures, such as the number of layers, neurons per layer, and connections. Fundamentally, a neural network operates as a nonlinear mapping denoted by $y = F(x)$, where the mapping function F is established during a

training phase. This training phase enables the network to effectively associate input patterns x with their corresponding output y. Through iterative adjustments to its parameters, the neural network learns to capture and represent complex relationships between the input and output, ultimately achieving the desired mapping. Given a collection of training examples (x, y), there exists a multitude of network sizes that could potentially learn to map input patterns x to output patterns y. However, determining which network size is the most appropriate for a given problem is often far from straightforward. To date, there is no definitive answer to this question [150].

2.1 Feed-forward neural network

The feed-forward neural network (FNN), recognized as the multilayer perceptron (MLP), stands as a foundational architecture within the domain of artificial neural networks. It is the first, simplest and most common type of neural network used in various machine learning applications [69, 150, 151]. An FNN consists of an input layer, one or more hidden layers, and an output layer. Each layer is composed of interconnected neurons. Information flows through the network in forward direction, from the input layer through the hidden layers to the output layer. A fundamental form of a feed-forward neural network is the linear network, which consists of input and output nodes organized in layers. In this simple architecture, the inputs are directly connected to the outputs through a set of weights. The information flows straightforwardly through the network without intricate transformations or hidden layers. The node outputs are determined by calculating the sum of the products between the weights and their corresponding inputs.

The advancement of neural network research has led to the development of multi-layer feed-forward neural networks, commonly known as MLPs. MLPs are comprised of multiple layers of perceptrons and are fully connected, meaning that each neuron in a layer is connected to every neuron in the subsequent layer. Perceptron represents a single neuron in the network.

It receives inputs, applies weights to these inputs, and computes an output using an activation function [152].

The concept of MLPs was first introduced by Alexey Ivakhnenko and Valentin Lapa in 1965 [153]. MLPs are often informally referred to as vanilla neural networks, particularly when they consist of a single hidden layer. In an MLP, each neuron establishes connections with neurons in both the previous and following layers. The computation within a neuron involves weighing the inputs received from the preceding layer, applying an activation function to the resulting sum, and transmitting the output to the subsequent layer. During the training process, MLPs iteratively adjust the weights and biases of the neurons in order to minimize the discrepancy between the predicted output and the actual output. This iterative optimization process allows MLPs to learn complex patterns and relationships within the input data, making them suitable for a wide range of tasks across various domains.

In the domain of deep learning, backpropagation is the most commonly used technique for training MLPs [75]. The primary objective of backpropagation is to analyze the relationship between the network's error and its weights and biases. By computing this relationship, the backpropagation algorithm enables the optimization of the network's weights and biases, leading to a reduction in the overall error and an improvement in performance. By iteratively adjusting the weights and biases based on the calculated gradients of the loss function with respect to the weights and biases, backpropagation aids the network in learning the appropriate transformations and representations necessary to make accurate predictions or classifications. This iterative process gradually improves the network's ability to generalize and make correct mappings, enhancing its overall learning capability. The algorithm involves computing the gradient of the error function with respect to the weights and biases and using it to update their values. This iterative process continues until the network is able to accurately predict the output for new input data [154, 155, 156].

Figure 2.1 shows a basic feed-forward neural network architecture having an input layer with two nodes. The hidden layer consists of three nodes, and the output layer comprises a

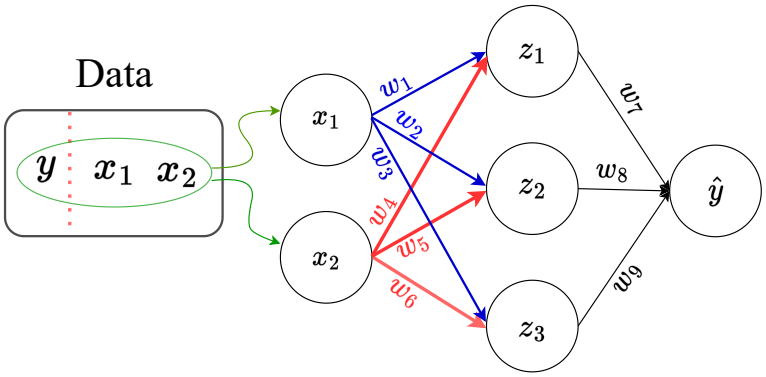


Figure 2.1: A basic feed-forward neural network is constructed with a single input layer comprising of two nodes, $\boldsymbol{X} = (x_1, x_2)$, followed by a hidden layer consisting of three nodes shown by z_1, z_2, z_3, and culminating in an output layer containing one node, denoted as $\boldsymbol{Y} = y$. The connections between the input layer and the hidden and output layers are represented by weights w_i for $i \in 1, 2, 9$.

single node. To illustrate a specific instance, let's consider an input vector $\boldsymbol{X} = (x_1, x_2)$ and the desired output $\boldsymbol{Y} = y$. Initially, the weights and biases connecting the input layer to the hidden and output layers are randomly or near-zero initialized. The network utilizes the weights and biases, in conjunction with the activation functions, to compute the outputs of the hidden layer nodes as well as the final output of the network. The output of each node in the hidden layer can be calculated by taking the weighted sum of the inputs for that particular node,

$$z_1 = L(w_1 \cdot x_1 + w_4 \cdot x_2 + b_1),$$
$$z_2 = L(w_2 \cdot x_1 + w_5 \cdot x_2 + b_2),$$
$$z_3 = L(w_3 \cdot x_1 + w_6 \cdot x_2 + b_3),$$

where L represents the activation function.

In a neural network, the output of each layer becomes the input for the next layer, establishing a sequential flow of information through the network. This iterative process of passing information from one layer to the next is repeated for each neuron in the output layer. It ensures that the information is effectively transmitted and propagated throughout the entire network,

culminating in the computation of the final output

$$\hat{y} = L(w_7 \cdot z_1 + w_8 \cdot z_2 + w_9 \cdot z_3 + b_4).$$

The process of training a neural network involves several interconnected steps. First, the error for each output neuron is computed by comparing the predicted output $\hat{y}$ with the actual output y using a chosen loss function E. This error serves as a measure of the network's performance. Next, the backpropagation algorithm is employed to calculate the gradients of the error with respect to the trainable parameters of a neural network, weights and biases, denoted by $\boldsymbol{\theta}$. During the training process, the trainable parameters of the neural are updated according to

$$\theta_t = \theta_{t-1} - \epsilon \frac{\partial E}{\partial \boldsymbol{\theta}}. \tag{2.1}$$

This process efficiently propagates the error from the output layer back to the hidden layers, allowing for adjustments to be made at each layer. In the optimization process, the loss function E is often chosen to be the squared-error function. The learning rate ϵ plays a significant role as it determines the step size for updating the weights and biases. It controls the extent to which the parameters are adjusted based on the calculated gradients. Selecting an appropriate learning rate is crucial because a value that is too small may result in slow convergence, while a value that is too large can cause the optimization process to diverge.

This update rule guides the optimization process, gradually minimizing the loss function and improving the network's performance. Optimization is a crucial process in ML, where the model is iteratively trained to find the optimal values for the model's parameters. It plays a vital role in achieving better results and improving the performance. By optimizing the model, we aim to find the optimal set of parameters that minimize the loss function, thereby improving the accuracy and effectiveness of the model's predictions.

By combining these steps, neural networks can learn from data, adjust their parameters, and optimize their performance. This iterative process of computing errors, backpropagating

gradients, and updating parameters allows the network to converge towards a more accurate representation of the desired output, making it capable of solving complex problems and making reliable predictions.

2.2 Recurrent neural network

A Recurrent Neural Network (RNN) is a type of artificial neural network designed to handle sequential data, such as time series or natural language text [157, 158]. It incorporates the concept of recurrence into its architecture, allowing for the processing of sequential information. Similar to feed-forward neural networks, a basic RNN consists of three essential components: the input layer, the hidden layer, and the output layer. However, RNNs utilize feedback connections that enable the flow of information in both the forward direction and in a recurrent manner. In each time step, the input layer receives sequential input data, serving as the initial stage for processing the sequential information within the network.

The hidden layer in a RNN plays a crucial role by incorporating recurrent connections, allowing the network to maintain a memory of past inputs. In each time step, the hidden state is updated by combining the current input with the previous hidden state. This hidden state serves as a memory bank, capturing important information from previous time steps. By retaining this memory, the network can learn and capture temporal dependencies within the data, enabling it to effectively analyze sequential patterns.

RNNs have been successfully applied to a wide range of tasks, such as speech recognition [159], machine translation [160], and image captioning [161]. However, training RNNs can pose challenges, primarily because of the issue of vanishing or exploding gradients [75]. These gradient-related problems can impede the network's ability to effectively learn long-term dependencies within the input sequence. The vanishing gradient problem occurs when the gradients calculated during backpropagation diminish exponentially as they propagate through the layers of the network. Consequently, the network struggles to capture and learn dependencies

that span across many time steps in the sequence. The exploding gradient problem arises when the gradients grow exponentially during backpropagation, leading to unstable and ineffective learning. Both the vanishing and exploding gradient problems hinder the RNN's capacity to retain and utilize information from earlier time steps, impairing its ability to model and learn long-term dependencies accurately. Addressing these challenges often requires careful initialization of the network's parameters, regularization techniques, or the utilization of specialized RNN architectures such as Long Short-Term Memory (LSTM) [162, 163], and Gated Recurrent Unit (GRU) [164].

2.2.1 Long short-term memory

Long Short-Term Memory (LSTM) is a specific type of recurrent neural network (RNN) that has demonstrated remarkable capabilities in learning sequential problems [162, 163]. LSTM networks excel particularly in scenarios where past inputs play a crucial role in predicting the current output. The LSTM model was introduced in 1997 by German scientists Hochreiter and Schmidhuber [165]. LSTM is specifically designed to retain information over long sequences. The architecture of LSTM enables it to effectively capture and retain dependencies and patterns that occur at different time steps in the input sequence. This distinctive capability to preserve information over long periods sets LSTM apart from other recurrent neural networks. It makes LSTM particularly suitable for tasks that require modeling long-term dependencies, including language modeling, speech recognition, and time series analysis.

As shown in Figure 2.2, the LSTM architecture comprises three primary states: the cell state (C_{t-1}, C_t), the input state (X_t, h_{t-1}), and the output state (h_t). The term 'state' refers to the internal memory components that store and carry information throughout the network's sequential processing. These states play a crucial role in retaining relevant information from previous inputs and propagating it to subsequent time steps. The cell state (C_{t-1}, C_t) is a fundamental component of the LSTM architecture, often referred to as the memory unit. It passes information along the LSTM units within the network, allowing for the transfer and modifica-

tion of information. This information is altered using gate layers, which play a significant role in controlling the flow of information within the LSTM. In addition, each LSTM unit comprises four essential components, the forget gate (f_t), the input gate (i_t), the new memory gate (C'_t), and the output gate (o_t). These gates enable the network to selectively retain or discard information based on the context of the input sequence. They are defined as follows:

- Forget gate: Determines what information should be forgotten or discarded from the previous cell state. It takes as input the previous cell state and the current input, applies a σ activation function, and produces a forget gate vector. This vector determines the portion of the previous cell state that should be retained and passed to the next time step.

- Input gate: A crucial component in determining the new information to be stored in the current cell state. It accomplishes this by taking the previous cell state and the current input and passing them through a σ activation function. The σ function squashes the values between 0 and 1, allowing the LSTM cell to selectively control the flow of information. Simultaneously, the current input is processed separately using a tanh activation function, resulting in the creation of a candidate vector. The candidate vector represents potential new information that could be added to the current cell state. It is an intermediate vector generated by applying the tanh activation function to the current input. The tanh activation function in an LSTM assists in capturing non-linear relationships, handling positive and negative values, and evaluating the importance of the current input for memory state. The candidate vector, along with the input gate, is then utilized to determine which parts of the candidate vector should be integrated into the cell state. By combining the input gate and candidate vector, the LSTM unit can effectively control and regulate the flow of new information into the current cell state, allowing for selective memory retention and integration of relevant information.

- Output gate: Takes both the previous cell state and the current input, applies a σ activation function, and generates an output gate vector. Subsequently, the current cell state

undergoes a tanh activation function, resulting in a transformed cell state. This transformed cell state is then element-wise multiplied with the output gate vector, producing the output of the LSTM cell. This final output encapsulates the relevant information from the cell state that is deemed important for further processing or prediction.

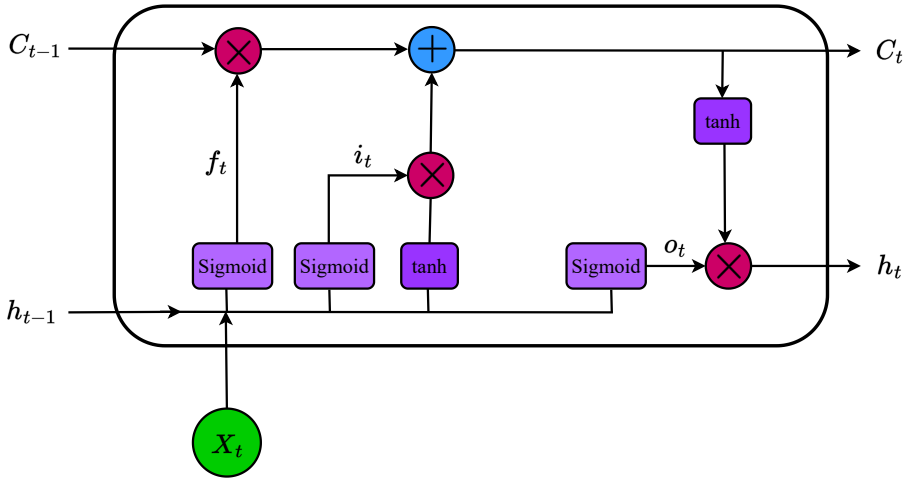


Figure 2.2: The LSTM architecture diagram illustrates the presence of three inputs: X_t (current input), h_{t-1} (previous hidden state), and C_{t-1} (previous cell state). It also showcases the existence of three gates, input gate, forget gate, and output gate. Furthermore, the diagram includes two outputs, h_t (current hidden state) and C_t (current cell state). The symbol $\times$ represents element-wise multiplication and the symbol + represents element-wise addition.

Each gate has a σ activation function that outputs values between zero and one, controlling the flow of information. Additionally, the LSTM unit includes a memory cell that utilizes the tanh activation function to regulate the cell state. Terminology:

- h_{t-1} is the output of the LSTM at the previous time step, that is, $t-1$.
- X_t is the input to the LSTM at time step t
- C_{t-1} is the cell state at the previous time step $(t-1)$.
- $i(t)$, $f(t)$, $o(t)$, and $g(t)$ are the values of the input gate, forget gate, output gate, and cell update vectors at time step t, respectively.

Each LSTM unit receives three inputs: X_t, h_{t-1}, and C_{t-1}, and generates one output h_t (output at time step t) and a new cell state C_t. The input h_{t-1}, which comes from the previous LSTM unit, plays a role in controlling the information flow. If the current unit is the first unit of the LSTM, there is no previous input available. In such cases, a randomly generated value is assigned to h_{t-1}. Once these inputs are processed through the internal gates, they are used to update the cell state from C_{t-1} to C_t and contribute to predicting the output h_t of the current LSTM unit.

The forget gate in an LSTM determines what information should be retained or forgotten from the previous cell state (C_{t-1}). It uses a σ layer to make this decision. The forget gate takes inputs from the input state (X_t) and the previous output state (h_{t-1}) and produces a value between 0 and 1 for each element in the cell state C_{t-1}. The purpose of training a network within the forget gate is to produce outputs close to 0 for irrelevant input components and closer to 1 for relevant ones. If the output is 1, the corresponding information is retained completely, while if it is 0, the information is entirely removed from the cell state C_{t-1}. The mathematical formula for the forget gate is given by

$$f_t = \sigma(W_f[h_{t-1}, X_t] + b_f), \tag{2.2}$$

where W_f represents the weight matrix, which is a square matrix of dimension $(n \times n)$, n is the number of features, and b_f represents the bias term associated with the forget gate.

In the first process, the inputs X_t and h_{t-1} are passed through a σ function to generate the input gate (i_t). The input gate decides which values should be passed on to the next step. The mathematical formula for the input gate is

$$i_t = \sigma(W_i[h_{t-1}, X_t] + b_i). \tag{2.3}$$

In the second step, a tanh function is employed to process the inputs (X_t) and h_{t-1}, generating

values for $C_t^{'}$ in the range of $[-1, 1]$,

$$\hat{C}_t = \tanh(W_c[h_{t-1}, X_t] + b_c). \tag{2.4}$$

The terms W_c and b_c represent the weight matrix and bias vector. A new cell state, C_t, is generated by multiplying the previous cell state, C_{t-1}, with the forget gate, f_t, this helps the LSTM unit to selectively retain relevant information and discard irrelevant or outdated information. Then it is added to the element-wise multiplication of the input gate i_t, and the values of $\hat{C}_t$. Therefore, the LSTM unit can selectively update the cell state with new information. The mathematical formula for computing C_t is given by

$$C_t = f_t C_{t-1} + i_t \hat{C}_t. \tag{2.5}$$

The newly generated cell state, C_t, contains crucial information that is passed on to the next LSTM unit in the network. This information is used by the subsequent unit to predict its output.

In the final step, the output of the LSTM unit is determined. This step involves two sub-steps. In the first sub-step, the information from the current input, X_t, and the previous output, h_{t-1}, are fed into a σ function. The σ function determines which parts of the cell state contribute to the output. Mathematically, this is represented by

$$o_t = \sigma(W_o[h_{t-1}, X_t] + b_o). \tag{2.6}$$

In the second sub-step, the values of the cell state, C_t, are passed through a tanh function to scale the values between -1 and 1. The output of the tanh function is then multiplied by the output of the σ function to produce the final output, h_t. This can be represented by the following formula,

$$h_t = o_t \tanh(C_t), \tag{2.7}$$

where o_t represents the output gate, C_t represents the cell state, and tanh represents the hyper-

bolic tangent function. The multiplication of the output gate with the scaled cell state allows the LSTM unit to selectively include relevant information from the cell state in the final output.

2.3 Convolutional neural network

Convolutional Neural Networks (CNNs) are a specialized type of deep learning algorithm primarily utilized for image and video processing tasks [166]. CNNs have found extensive utility in object and face recognition, and image classification [167, 65]. At the heart of a CNN lies the convolutional layer, which employs filters or kernels to extract important visual features like edges, shapes, and patterns from the input image.

As illustrated in Figure 2.3, the typical architecture of CNN consists of multiple hidden layers that facilitate the extraction of information from images. The three key layers in a CNN are:

1. Convolutional layer
2. Pooling layer
3. Fully connected layer

A CNN model typically operates in two main steps, feature extraction and classification. In the feature extraction step, the CNN learns to automatically extract relevant and meaningful features from, for example, the input images. This is achieved through the use of convolutional layers. The convolution layer is an initial step in extracting important features from an image. It consists of multiple filters that perform the convolution operation. Each filter convolves over the image, performing element-wise multiplication and summing up the results to produce a feature map. The filters capture different patterns, such as edges, corners, and textures, at various spatial locations within the image. The values of the subsequent feature map are calculated as

$$G[m,n] = (f * h)[m,n] = \sum_i \sum_j h[i,j] f[m-i, n-j], \tag{2.8}$$

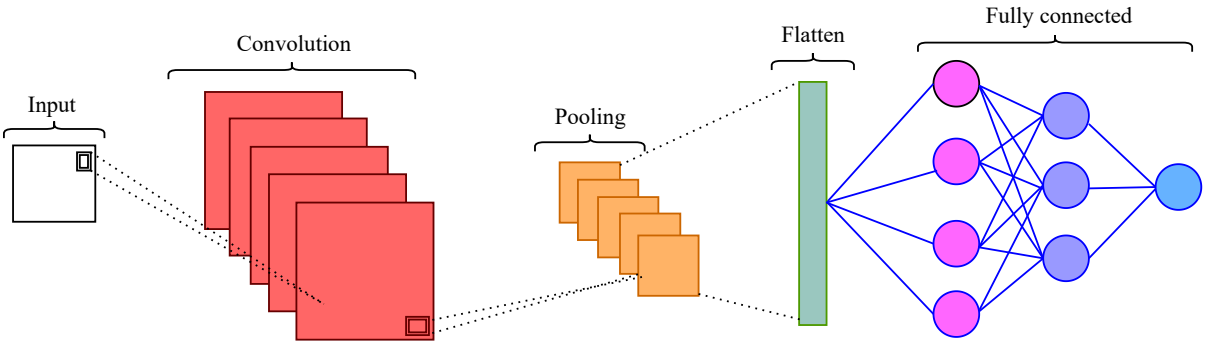


Figure 2.3: A basic architecture of CNNs comprises multiple convolutional layers, followed by pooling layers, which reduce the dimensionality of the feature maps to extract the most relevant features. After the pooling layers, a flattening layer is applied to convert the multi-dimensional feature maps into a one-dimensional vector. This prepares the data for further processing by fully connected layers. The fully connected layers are responsible for making predictions or performing classification tasks based on the extracted features.

where the input image is represented by f and the kernel by h. The indices of rows and columns in the resulting matrix are indicated by m and n, respectively.

Pooling is an operation used for down-sampling in order to reduce the dimensionality of the feature map. After the rectified feature map is obtained, it is passed through a pooling layer to produce a pooled feature map. Pooling layers identify different parts of the image, such as edges, by downsampling the feature map. By reducing the spatial resolution of the features, pooling layers highlight important patterns and structures in the image. The pooling operation helps to capture the presence of edges by effectively summarizing the information in a local neighborhood and retaining the most prominent features. This allows the network to focus on important visual cues, making it more robust to variations in position and scale.

Figure 2.4 shows an example scenario where a 6×6 image is represented as a matrix of pixel values, where each pixel can have a value of either 0 or 1. Additionally, a filter matrix of size 3×3 is used. The process of obtaining the convolved feature matrix involves sliding the filter matrix over the image and computing the dot product at each position. This operation captures important information from the image and produces the convolved feature matrix, resulting in a matrix of size 4×4. Furthermore, a max pooling layer with 2×2 filters is applied to reduce the dimensionality of the feature map. A max pooling layer is a type of pooling layer

that performs down-sampling by dividing the input into non-overlapping regions and selecting the maximum value from each region [168]. The process of flattening in a CNN is used to transform the pooled feature maps, which are initially represented as 2-dimensional arrays, into a 1-dimensional array. This involves rearranging the elements of the pooled feature maps in a sequential manner, forming a vector. This vector representation is often referred to as a single, long, continuous linear vector. It allows for a more straightforward input into the fully connected layer, which is responsible for classifying the image based on the extracted features.

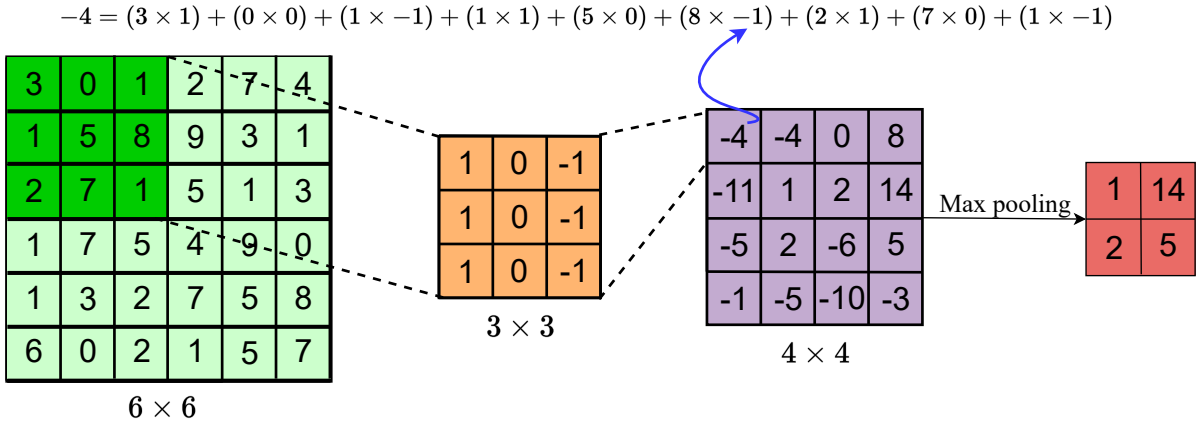


Figure 2.4: Example of a convolution operation. It involves taking a 3×3 matrix of numerical values known as the kernel or the filter, and applying it to a 6×6 input image. During this process, the kernel is slid over the image, and element-wise multiplication is performed between the kernel and the corresponding pixel values in the image. The results of these multiplications are then summed up to generate an output using Equation (2.8). This operation allows the network to extract relevant features from the image that are essential for subsequent analysis and processing.

After the data has been flattened into a 1D array, it is forwarded to the fully connected layer. As we use CNNs for regression purposes in this thesis, the output layer consists of a single node without an activation function. The predicted value is directly obtained from this node, representing the estimated numerical value [168]. Extensive research and literature have shown that CNNs generally follow a common architecture, which involves stacking convolutional layers and pooling layers in a repeated manner before forwarding the output to

fully-connected layers [169, 170]. This structured approach has proven to be successful in various CNN applications.

One advantage of CNNs over recurrent-type networks is their convolutional structure, which leads to a smaller number of trainable weights. This characteristic makes CNNs more efficient during the training and prediction processes. By leveraging shared weights and local receptive fields, CNNs can effectively capture spatial and hierarchical patterns in data, while reducing the overall number of parameters [171].

2.4 Physics-Informed Neural Networks

Deep neural networks have garnered significant attention in the field of ML due to their impressive universal approximation properties. Through the training process, the models can learn to solve and uncover the equations that govern the observed phenomena. However, training deep neural network models typically demands a substantial quantity of labeled data, which is frequently scarce in many scientific applications. A physics-informed approach, even with relatively small amounts of labeled data, can be used to solve and discover equations. Physics-Informed Neural Networks (PINNs) are a machine learning approach proposed by Raissi et al. in 2019 [114] that combines the power of neural networks with the principles of physics to solve and discover partial differential equations (PDEs). This approach differs from the standard supervised learning as it leverages the physical properties of the PDE to guide the training process: It incorporates known physical laws and governing equations as constraints during the learning process. The algorithm incorporates the governing equation of a problem into the neural network structure, and enhances the loss function by introducing a residual term derived from the equation itself. The PDE residual is included as a regularization term in the loss function of fully-connected neural networks.

This residual term serves as a penalty, restricting the search space to feasible solutions. By doing so, the task of inferring solutions for PDEs is reformulated as an optimization problem,

where the goal is to minimize the loss function.

PINNs have been applied to two main problems in the field of PDEs:

- Data-driven solutions: PINNs can be utilized to solve PDEs by incorporating boundary and initial conditions into the loss function. The network is trained to minimize the loss, ultimately yielding a solution. This approach is particularly useful when analytical solutions are not available or when enhancing the accuracy of numerical solutions is desired. By learning from available data and integrating the known physical constraints, PINNs provide efficient and accurate approximations to PDE solutions [172].

- Discovering PDEs: PINNs enable the discovery of PDE models from data by training the network to fit the observed data and identifying the PDE that describes the system's dynamics. This approach is useful in cases where the underlying physics is unclear or when deriving the PDE analytically is challenging. By combining ML with data-driven insights, PINNs provide a powerful tool for exploring and understanding complex systems, offering a direct and effective way to discover PDEs from observed data [172, 173].

In PINNs, both initial and boundary conditions can be integrated into the loss function during the training process. This is achieved to ensure that the neural network not only fits the given data but also respects the physical behaviors dictated by these conditions. Initial conditions are conditions specified at the starting point of a system's evolution. In PINNs, they are incorporated by evaluating the neural network's predictions at the initial time step and then comparing them to the provided initial values. The discrepancy between the predicted and actual initial values contributes to the loss function. Boundary conditions, on the other hand, are constraints set at the boundaries of the problem domain. In most cases, essential boundary conditions can be enforced by introducing penalization terms to the loss function. These penalization terms penalize the deviation of the neural network's predictions from the desired boundary values. These terms are multiplied by constant weighting factors that control their influence on the

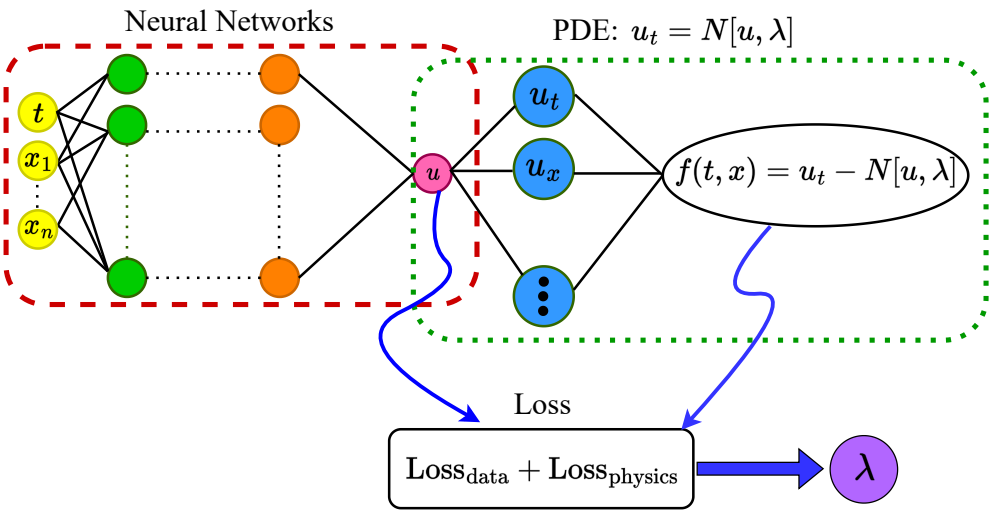


Figure 2.5: The schematic of the PINNs methodology is utilized to uncover the unknown parameters of a PDE. An MLP is trained using independent variables to predict the value of u. The predicted u is then utilized in the physics-informed portion of the methodology. The loss function consists of two components, namely $\text{Loss}_{\text{data}}$ and $\text{Loss}_{\text{physics}}$, which is minimized to determine the unknown parameters λ.

overall loss.

PINNs can be trained with relatively small datasets, especially when the underlying physics of the problem is known and incorporated into the loss function. In such cases, the focus is on capturing the physics-based constraints rather than relying solely on the quantity of training data. However, in situations where more complex or diverse patterns need to be learned, a larger dataset may be beneficial to improve the generalization and accuracy of the PINN model. Two main challenges in using PINNs are the substantial computational resources required for training and the complexity of designing appropriate loss functions and constraints. The high computational cost arises from the iterative nature of PINN training, involving forward and backward passes, weight updates, and loss function minimization. This intensity increases when dealing with large datasets or complex network architectures. On the other hand, formulating accurate loss functions and constraints that effectively capture the underlying physics can be a difficult and intricate task. It requires a deep understanding of the problem domain and expertise in translating the physics into mathematical formulations [127, 174].

The general form of a non-linear PDE is given as

$$u_t - N[u, \lambda] = 0 \quad x \in \Omega \subset \mathbb{R}^n, \tag{2.9}$$

where u is a function of t and x represents the solution of the equation, u_t is its time derivative, and $N[u, \lambda]$ denotes a non-linear operator that depends on the parameter λ. In the context of data-driven discovery of PDEs, the goal is to determine the values of the model parameters λ that best describe the observed data. These parameters represent the unknown quantities or characteristics of the underlying system that we want to estimate or infer based on the available data.

To encapsulate the left-hand side of Equation (2.9), we define the residual $f(t, x)$ as

$$f(t, x) = u_t - N[u, \lambda]. \tag{2.10}$$

First, as shown in Figure 2.5, a neural network is trained using the available data, which includes initial and boundary values, to learn $u(t, x)$, $f(t, x)$, and parameters λ. Predicted $\hat{u}(t, x)$ is subsequently utilized to satisfy Equation (2.10). The loss function plays a crucial role in guiding the training process. The loss function in PINNs consists of two main components, $\text{Loss}_{\text{data}}$ and $\text{Loss}_{\text{physics}}$,

$$\text{Loss} = \text{Loss}_{\text{data}} + \text{Loss}_{\text{physics}}. \tag{2.11}$$

$\text{Loss}_{\text{data}}$ measures the discrepancy between the predicted values of the neural network and the observed data. It quantifies how well the network fits the given data points. For example, a commonly used loss function for regression problems is the mean squared error (MSE), which compares the predicted values with the actual data points via

$$\text{Loss}_{\text{data}} = \frac{1}{N_u} \sum_{i=1}^{i=N_u} |u(t_i, x_i) - \hat{u}(t_i, x_i)|^2, \tag{2.12}$$

where N_u is the number of training points, and $u(t_i, x_i)$ and $\hat{u}(t_i, x_i)$ denote the observed and

predicted initial and boundary values at the training points.

$\text{Loss}_{\text{physics}}$ incorporates the governing equations or physical constraints into the training process. It ensures that neural network solutions satisfy the underlying laws of physics. This is achieved by evaluating the residual of the differential equation, which represents the difference between the left-hand side and the right-hand side of Equation (2.10). $\text{Loss}_{\text{physics}}$ penalizes the deviation from the governing equation and encourages the network to learn solutions that satisfy the physics constraints,

$$\text{Loss}_{\text{physics}} = \frac{1}{N_u} \sum_{j=1}^{j=N_f} \left| f(t_j, x_j) \right|^2 , \tag{2.13}$$

where N_f is the number of residual points, and $f(t_j, x_j)$ is the value of the function $f(t, x)$ at the residual points.

During training, the network parameters are updated iteratively by computing the gradients of the loss function with respect to these parameters. In addition to updating the network parameters, the optimization process also involves iteratively adjusting the values of λ. This allows for a search in the parameter space to find the combination of values that yields the best fit between the model predictions and the observed data. The overall loss in PINNs is a combination of $\text{Loss}_{\text{data}}$ and $\text{Loss}_{\text{physics}}$, which are typically weighted to reflect their relative importance. The relative weights assigned to the losses can vary depending on the specific problem and the desired balance between accurately fitting the data and enforcing the physics constraints. By jointly optimizing both components of the loss, PINNs aim to find solutions that not only match the observed data, but also adhere to the underlying physical laws governing the system.

2.4.1 eXtended Physics-Informed Neural Networks

Since the introduction of PINNs, several extensions have been developed to enhance their performance and to expand their applicability to various problem domains. These

extensions include Physics-Informed Attention-Based Neural Networks (PIANNs) [175], Generative Adversarial Physics-Informed Neural Networks (GA-PINNs) [176], Graph Convolutional Networks (GCNs) [177], Conditional Physics-Informed Neural Networks (CPINN) [126], eXtended Physics-Informed Neural Networks (X-PINNs) [127, 128], and Bayesian Physics-Informed Neural Networks (B-PINNs) [178].

One of the primary drawbacks of PINNs is their significant training cost, which can have a detrimental effect on their performance, especially in real-life applications where real-time execution is necessary. The computational expense of training a PINN model can be substantial, making it challenging to achieve rapid convergence and efficient deployment of the model in time-critical scenarios [126, 127, 128]. Therefore, there is a critical need to address this issue and develop techniques that can accelerate the convergence of PINNs without compromising their performance. One of the methods is to partition the computational domain into several subdomains to train each subdomain separately.

X-PINNs are a generalized approach for solving and discovering PDEs in complex-geometry domains. They involve generalized space-time domain decomposition in order to provide computationally efficient solutions to PDEs across large spatial and temporal domains. In this approach, the domain is first split into smaller subdomains. Then, the PDEs are solved in each subdomain using PINNs and at the interfaces with certain continuity conditions imposed as soft-constraints in the loss function. This allows X-PINNs to use large neural networks without the common problem of overfitting. X-PINNs also reduce the computational cost associated with training due to their implicit concurrent implementation. X-PINN is designed to leverage parallel processing capabilities, particularly through the implicit concurrent implementation, which allows for simultaneous computations and exploits parallelism. This can help reduce the computational cost associated with training by utilizing multi-core processors or specialized hardware like GPUs.

The original studies in this thesis employ both PINNs and X-PINNs to enable the exploration of white-box and gray-box learning respectively, for discovering PDEs based on avail-

able data.

2.5 Symbolic regression

Symbolic regression is the process of discovering an optimal mathematical expression that accurately predicts a continuous target variable [179]. It has found significant applications in the field of physical systems, particularly in deriving natural laws from observational data [180]. The primary objective of symbolic regression is to discover a model that accurately represents the underlying relationship between the input variables and the target variable. This model should also be interpretable and easily understandable.

Given a set of features represented by $x \in \mathbb{R}^n$ and a corresponding target variable y for which a ground-truth solution $y(x) = f(x)$ exists, symbolic regression aims to learn a mapping $y(x) = f^*(x)$. The optimal function f^* can be composed of any combination of features from x and mathematical operators that transform or combine these features. It involves searching for the optimal combination of base functions, such as addition, trigonometric functions, and exponentials, that can effectively capture the underlying relationships in the data [181, 181, 182, 183]. Symbolic regression is a fundamental concept that underlies various fields of research, including engineering [184], psychology [185], economics [186], physics [187], and chemistry [188]. Various optimization techniques, such as genetic programming [189], evolutionary algorithms, or Bayesian approaches, are commonly employed to guide the search and evolution process towards finding the best-fitting model.

2.5.1 Genetic programming approaches

Genetic programming is widely used to learn symbolic regression models [190, 191]. By randomly generating a set of candidate expressions and then gradually improving the candidates through series of mutations, crossovers, reproduction, and selection operations, until the best candidate model satisfactorily fits the designated target variable. In these cases, the symbolic

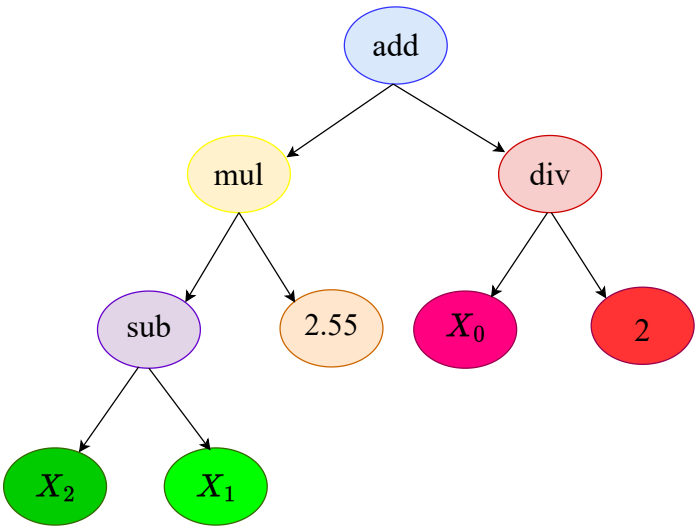


Figure 2.6: Expression tree representing a symbolic regression model, with three inputs X_0, X_1, and X_2, and the equation $2.55(X_2 - X_1) + \frac{X_0}{2}$.

regression model is represented as an expression tree, in which the internal nodes represent mathematical operators, while the terminal nodes represent features or constants, which collapse into a single mathematical expression via recursive tree traversal. Figure 2.6 shows an expression tree for presenting symbolic regression. The example consists of three inputs X_0, X_1, and X_2, which generate a mathematical presentation $2.55\,(X_2 - X_1) + \frac{X_0}{2}$.

In traditional genetic programming approaches, the initial population of candidate expression trees is created by randomly combining features, mathematical operators, and constants. These expression trees represent the individuals in the population, and they consist of hierarchical compositions of primitive functions and terminals that are suitable for the specific problem domain. The primitive functions used in genetic programming typically include basic arithmetic operations such as addition, subtraction, multiplication, and division. They also encompass mathematical functions such as logarithms, exponentiation, and trigonometric functions. These primitive functions provide the building blocks for creating complex expressions. The terminals, on the other hand, represent inputs to the expressions and can include variables, constants, or other data relevant to the problem domain. Numeric constants, which are fixed values used in calculations, can also be included as terminals. By combining these primitive functions

and terminals in various hierarchical compositions, genetic programming aims to evolve individuals that can effectively solve the given problem. The algorithm applies genetic variation and selection techniques to iteratively improve the population. From this population, a subset of trees is selected for the next generation based on their fitness, which is determined by their prediction error or correlation with the target variable. Through processes like crossover and mutation [192], the algorithm generates new individuals with combinations of functions and terminals from the fittest individuals in the current population.

Crossover and mutation are two fundamental genetic operators used to modify the genetic material of the trees, introducing variations and combinations of features, mathematical operators, and constants. The purpose of these genetic operations is to explore the different combinations of genetic material in the population, and potentially improve the fitness of the individuals in subsequent generations. After the selected trees have undergone genetic operations, they progress to the next generation, and the fitness evaluation is conducted once more. This iterative process continues until a stopping criterion is met, which can be based on a predictive performance evaluation or reaching a predefined maximum number of generations [181, 190, 192].

The challenge in symbolic regression lies in searching the space of mathematical expressions to find a model that optimally balances accuracy and simplicity. This involves exploring different combinations of mathematical operations, functions, and variables to construct candidate expressions, and then evaluating their performance against the dataset. In symbolic regression, it is possible to have multiple symbolic expressions that represent the same underlying function. This occurs due to the various ways in which elementary functions, variables, and constants can be combined to portray a specific mathematical relationship.

Additionally, while the complexity of generic functions can pose significant challenges for symbolic regression, making their discovery nearly impossible in certain cases, it is worth noting that many practical functions of interest often possess simplifying properties. These properties, such as symmetries, separability, and compositionality, can be leveraged to generate

accurate multidimensional expressions [193].

Chapter 3

Data generation: An overview

Generating appropriate datasets for training deep learning models is a critical step. This chapter aims to provide a review the concepts of phase-fields and droplet spreading, which serve as the primary datasets for our deep learning methods in this thesis. Additionally, we discuss the methodologies employed to generate the required data for training and evaluating our machine learning models. It is important to note that the central focus of this thesis is the development and application of ML models and techniques specifically designed for the discovery of equations. These datasets were used in the subsequent chapters for developing and applying deep learning models for equation discovery.

3.1 Phase-field modeling

The phase-field method is an important technique for investigating non-equilibrium interface phenomena [141]. Non-equilibrium conditions refer to situations where a system is not in a state of thermodynamic equilibrium. In thermodynamics, equilibrium is a state where all forces, flows, and energy transfers within a system are balanced, resulting in a stable and unchanging state. In contrast, non-equilibrium conditions occur when a system is subjected to external influences or experiences changes that prevent it from reaching or maintaining a state of equilibrium. These conditions can arise due to various factors, such as temperature

gradients, pressure differences, chemical reactions, mechanical stresses, or external forces. The main objective of phase-field modeling is to study the dynamics and structural changes that occur during these processes. It finds applications in various fields, ranging from dendrite growth in materials science to reaction-diffusion systems, damage and fracture models, and even biological systems [141].

Phase-field modeling is a coarse-grained approach (see Figure 1.6) used to study the behavior of materials, particularly in processes involving slow variables such as concentrations. Phase-field modeling technique employs continuum fields, which are continuous representations of physical quantities, to describe the system's behavior and evolution. This method utilizes a continuous variable, the order parameter, that depends on both position and time to describe the state of the material, differentiating, for example, between different phases. The interfaces separating these phases are characterized by smooth transitions in the phase-field variable, with rapid changes occurring within a narrow region.

This modeling technique involves describing the dynamics of one or more fields through a set of coupled dynamical equations given as PDEs. By solving these PDEs, the system can be time-evolved, allowing for, for example, studies of complex phenomena and the prediction of microstructural changes over time. In the construction of phase-field models, one typically proposes a phenomenological free energy functional known as the Ginzburg-Landau free energy or Lyapunov functional [141]. This functional is developed by expanding the order parameter using a gradient expansion and subsequently taking a functional derivative. The terms included in this expansion must satisfy the symmetries associated with the specific system being modeled. The free energy functional, denoted as F, plays a fundamental role in phase-field modeling. It is typically expressed as an integral over space and consists of two main terms: the gradient energy term $|\vec{\nabla}U|^2$ (in a simple case) and the bulk free energy term $f(U)$. The gradient energy term accounts for the spatial variations or gradients of the order parameter $U \equiv U(\vec{x}, t)$. This term captures the energy associated with changes or variations in the order parameter across space.

Using only the term $|\vec{\nabla}U|^2$, the general form of the free energy functional is given as

$$F = \int d\vec{x} \left[|\vec{\nabla}U|^2 + f(U) \right]. \tag{3.1}$$

One of the fundamental aspects of phase-field modeling is the order parameter. Order parameters can be categorized as either conserved or non-conserved, depending on the nature of the properties and dynamics being examined. Conserved order parameters correspond to quantities that remain constant during the evolution of the system. On the other hand, non-conserved order parameters do not possess such constraints and can freely change over time.

In this thesis, the focus is on systems that can be described using a single order parameter. The behavior of this order parameter is governed by different equations of motion depending on whether it is conserved or non-conserved. For a non-conserved order parameter, the equation of motion is given by

$$\frac{\partial U}{\partial t} = -\Gamma \frac{\delta F}{\delta U}, \tag{3.2}$$

and for a conserved order parameter, the equation of motion takes the form

$$\frac{\partial U}{\partial t} = \Gamma \nabla^2 \frac{\delta F}{\delta U}. \tag{3.3}$$

These equations describe the rate of change of the order parameter U with respect to time, $\delta F/\delta U$ denotes the functional derivative of F with respect to U, and Γ is a constant representing the generalized mobility. The functional derivative captures how the free energy changes as the order parameter varies.

The bulk free energy term, denoted as $f(U)$ in Equation 3.1, represents the energy associated with the bulk properties of the system. In this particular work, the bulk free energy is modeled using a double well potential,

$$f(U) = \frac{a_4}{4}U^4 + \frac{a_2}{2}U^2. \tag{3.4}$$

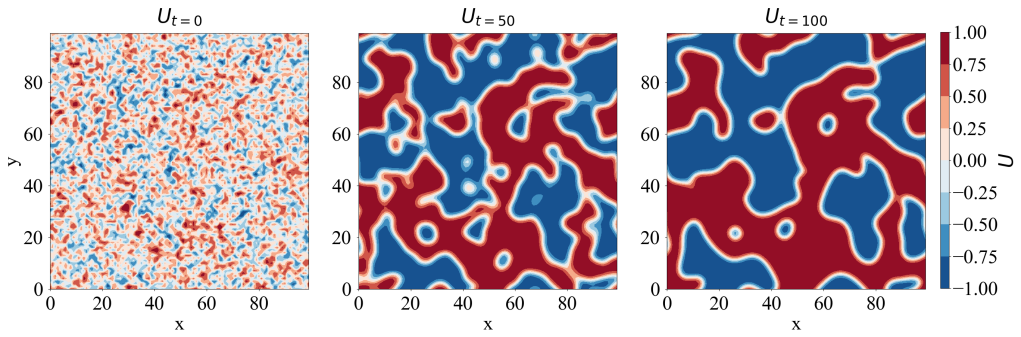


Figure 3.1: Snapshots of the field solutions for the Allen-Cahn model (Eq. (3.5)) at three distinct time points: $t = 0$, $t = 50$, and $t = 100$. The simulations (in dimensionless units) were conducted on a uniformly discretized two-dimensional grid with dimensions of 100×100 and a spacing of $\Delta x = \Delta y = 1$. The time step used in the simulations was $\Delta t = 0.1$, and the simulation duration extended up to $t = 20$.

Here, a_4 and a_2 are constants that determine the shape and properties of the potential.

By incorporating both the gradient energy term and the bulk free energy term into the free energy functional, one can capture the interplay between spatial variations and the overall energetics of the system. This enables the simulation and study of phase transitions, pattern formation, and other complex phenomena in materials and systems. In this thesis, we employed three single order parameter phase-field models, the Allen–Cahn [194], the Cahn–Hilliard [195], and the phase-field crystal (PFC) model [5].

- The Allen-Cahn model, originally proposed by Allen and Cahn in 1972 [194], is a dynamical model used to describe the process of solidification. It is characterized by a single non-conserved order parameter. In this model, the equation of motion is given by

$$\frac{\partial U}{\partial t} = -M \left(\nabla^2 U + a_2 U - a_4 U^3 \right). \tag{3.5}$$

  Here, the constant Γ in Equation (3.2) is set to M in Equation (3.5), where M represents a constant related to the chemical mobility. Figure 3.1 shows snapshots from a simulation of the Allen-Cahn model at three specific time steps: $t = 0$, $t = 50$, and $t = 100$.

- The Cahn-Hilliard model, by Cahn and Hilliard in 1958 [195], is used to describe the

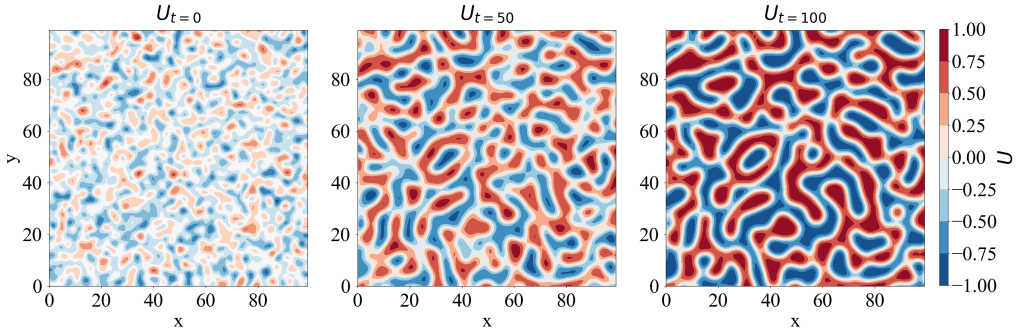


Figure 3.2: Snapshots of the field solutions for the Cahn-Hilliard model (Equation (3.6)) at three distinct time points: $t = 0$, $t = 50$, and $t = 100$. The simulations were conducted on a uniformly discretized two-dimensional grid with dimensions of 100×100 and a spacing of $\Delta x = \Delta y = 1$. The time step used in the simulations was $\Delta t = 0.01$, and the simulation duration extended up to $t = 20$.

phenomenon of spinodal decomposition. It is a conserved order parameter model, corresponding to Equation (3.3). By applying the free energy density given by Equation (3.1), the Cahn-Hilliard model can be expressed as

$$\frac{\partial U}{\partial t} = D\nabla^2 \left(\nabla^2 U + a_2 U + a_4 U^3 \right) , \tag{3.6}$$

where Γ in Equation 3.3 is set to D, a constant that represents the diffusion constant. Equation (3.6) is also known as Model B in the Hohenberg and Halperin classification [196].

- A free energy density, originally developed by Elder et al. [5, 197] is employed to describe the behavior of a crystal lattice at an atomistic scale within a phase-field model. This formulation, known as the Phase-Field Crystal (PFC) model, incorporates elastic effects and aims to capture the characteristics of materials at fine length scales. The PFC free energy density can be given as [198]

$$F(U) = \int d\vec{x} \left[\frac{U^3}{3} + \frac{U^4}{4} + U \left((q_0 + \nabla^2)^2 - \varepsilon \right) \frac{U}{2} \right] . \tag{3.7}$$

In this equation, the order parameter is denoted by U, and the integral is performed

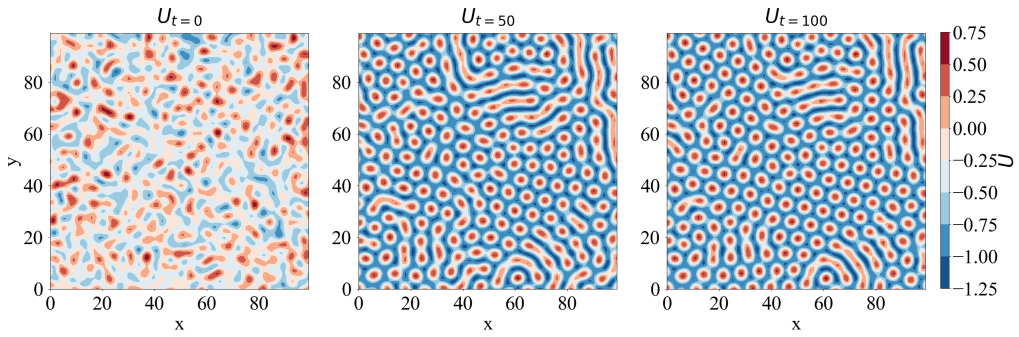


Figure 3.3: The field solutions for the PFC model (Eq. (3.8)) were captured at three specific time points: $t = 0$, $t = 50$, and $t = 100$. The simulations were performed on a uniformly discretized two-dimensional grid with dimensions of 100×100 and a spacing of $\Delta x = \Delta y = 1$. A time step of $\Delta t = 0.05$ was utilized in the simulations, and the simulation duration extended until $t = 100$.

over the spatial domain. The terms involving U^3 and U^4 contribute to the bulk energy, while the term involving the Laplacian operator ∇^2 accounts for the spatial variations of the order parameter. The parameters q_0 and ε are constants and represent the preferred wavevector associated with the hexagonal lattice structure and the lattice stiffness. Minimizing the PFC free energy functional in 2-d leads to the formation of a hexagonal periodic lattice configuration, which accurately describes the atomistic characteristics of the crystal lattice. The PFC model enables investigations into the behavior of materials at atomistic length scales and captures diffusive phenomena.The equation governing the dynamics of the PFC model with $\Gamma = 1$ is expressed as

$$\frac{\partial U}{\partial t} = \nabla^2 \left(U^2 + U^3 + \left((q_0 + \nabla^2)^2 - \varepsilon \right) U \right), \tag{3.8}$$

where the order parameter U represents the mass density. This equation captures the evolution of the system over time, considering the spatial variations and interactions within the crystal structure.

3.1.1 Simulation of phase-field models

The numerical simulations of the three phase-field systems discussed in this thesis were conducted using SymPhas [6], an open-source software package. This software provides a user-friendly interface that enhances the flexibility and ease of generating phase-field data for further analysis and investigation. It facilitates the direct definition of phase-field models using their corresponding PDE formulations.

For this study, the semi-implicit Fourier spectral method [199] was selected for the numerical solution. The simulations were performed on a two-dimensional grid with uniform discretization. Periodic boundary conditions were applied to the models, and the initial conditions were generated by populating the grid with values from a uniform random distribution between -1 and 1. The grid size was $n_x \times n_y$ and the grid spacing was $\Delta x = \Delta y = 1$. A time step of Δt was used, and the simulations were run until time t. The constants in the equations of motion were set to 1, except for ε in the phase field crystal (PFC) model, which was set to 0.1. The results of the simulation are presented in Figures 3.1, 3.2 and 3.3.

3.2 Droplet spreading

Droplet spreading is a fascinating phenomenon that occurs when a liquid droplet comes into contact with a solid surface. This process plays a significant role in various fields, including physics, chemistry, biology, and engineering. Understanding and controlling droplet spreading is crucial for applications such as surface coating, inkjet printing, microfluidics, and oil recovery, among others. The theoretical groundwork for understanding this phenomenon was laid down in the early 1800s by scientists Young and Laplace [200, 201]. Thomas Young made significant contributions to the understanding of liquid spreading through his work on the concept of contact angle. The contact angle is the angle formed between the liquid-vapor interface and the solid-liquid interface at the three-phase contact line, where the liquid, solid, and vapor phases meet.

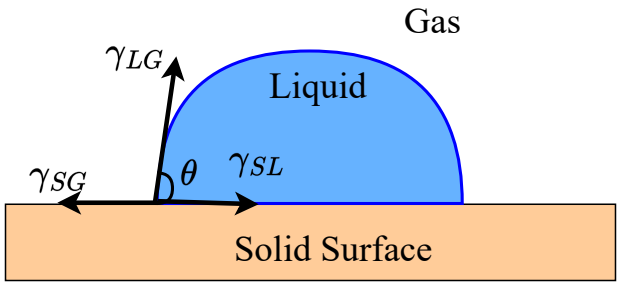


Figure 3.4: A schematic figure showing the various aspects related to the equilibrium contact angle ($\theta \equiv \theta_{\text{eq}}$) and surface tensions ($\gamma$) from Equation 3.9.

Young proposed an equation, now known as Young's equation, which relates the equilibrium contact angle to the interfacial tensions between the liquid, solid, and vapor phases [202]. Young's equation states that

$$\cos\theta_{\text{eq}} = \frac{\gamma_{SG} - \gamma_{SL}}{\gamma_{LG}}. \tag{3.9}$$

Here, θ_{eq} represents the equilibrium contact angle, while γ_{SG}, γ_{SL}, and γ_{LG} are the surface tensions between the solid-gas, solid-liquid, and liquid-gas phases, respectively (refer to Figure 3.4).

Young's equation is a fundamental relationship that establishes a connection between the contact angle and the interfacial tensions. While droplet spreading has been extensively studied for complete wetting scenarios, that is, when the droplet fully spreads over the surface, the behavior becomes more intricate in the case of partial wetting. Partial wetting occurs when the droplet only partially covers the solid surface. Understanding partial wetting is crucial for a comprehensive understanding of droplet spreading phenomena. To classify the contact angle regimes, four categories are commonly defined: complete wetting, the entire solid surface is covered by the fluid, resulting in ($\theta_{\text{eq}} = 0°$). This condition is satisfied when $\gamma_{SG} - \gamma_{LG} - \gamma_{SL} = 0$. High-wetting ($0° < \theta_{\text{eq}} < 90°$), low-wetting ($90° \leq \theta_{\text{eq}} < 180°$), and non-wetting ($\theta_{\text{eq}} = 180°$) [7, 8, 9].

One of the key aspects of droplet spreading analysis is the exploration of the relationship between the spreading area and time. The spreading of a droplet on a solid surface is often characterized by a power law relationship between the radius of the wetted area (r) and time

(t), expressed as $r \sim t^{\alpha}$. This power law describes how the size of the wetted region changes over time. Tanner's law, which is applicable to macroscopic complete wetting at late stages, states that the exponent α in the power law is approximately equal to $1/10$ [203, 204]. This means that the wetted area increases relatively slowly with time. It is worth noting that the power law behavior described by Tanner's law has also been observed at microscopic scales [8]. However, it is important to recognize that deviations from Tanner's law can occur in certain cases. For instance, surface topography and other complex phenomena can affect the spreading dynamics and lead to deviations from the ideal power law behavior [205, 206, 207].

3.2.1 Simulation of droplet spreading

In this thesis, the mDPD (multiphase many-body dissipative particle dynamics) method [208, 209, 210] was utilized to generate data for studying droplet spreading over a surface. The mDPD method, an extension of the traditional dissipative particle dynamics (DPD) model [211, 212], is a mesoscale simulation technique that allows for simulations of complex fluids and multiphase systems. DPD provides a framework for simulating fluid dynamics at an intermediate scale (see Figure 1.6), where the behavior of particles is described in a coarse-grained manner. By considering interactions between the particles and incorporating dissipative and random forces, DPD models capture mesoscopic phenomena and provide insights into complex fluid behavior.

In both the DPD and mDPD models, the motions of the particles are governed by Newton's equations of motion. The total force on particle i is the sum of three pairwise components: the conservative force ($\vec{F}^{\mathrm{C}}$), the dissipative force ($\vec{F}^{\mathrm{D}}$), and the random force ($\vec{F}^{\mathrm{R}}$),

$$\begin{aligned} \frac{d\vec{r}_i}{dt} &= \vec{v}_i, \\ m_i \frac{d\vec{v}_i}{dt} &= \vec{F}_i = \sum_{j \neq i} \vec{F}^{\mathrm{C}}_{ij} + \vec{F}^{\mathrm{D}}_{ij} + \vec{F}^{\mathrm{R}}_{ij}, \end{aligned} \tag{3.10}$$

where $\vec{r}_i$ and $\vec{v}_i$ are the position and velocity of particle i of mass m_i, and

$$\begin{aligned} \vec{F}_i^{\mathrm{D}} &= -\gamma\omega_{\mathrm{D}}(r_{ij})(\vec{v}_{ij}\cdot\vec{e}_{ij})\vec{e}_{ij}, \\ \vec{F}_{ij}^{\mathrm{R}} &= \zeta\omega_{\mathrm{R}}(r_{ij})(dt)^{-1/2}\xi_{ij}\vec{e}_{ij}, \\ F_{ij}^{\mathrm{C}} &= A\omega_{\mathrm{C}}(r_{ij})\vec{e}_{ij} + B(\rho_i+\rho_j)\omega_{\mathrm{B}}\vec{e}_{ij}, \end{aligned} \tag{3.11}$$

where $r_{ij} = |r_i - r_j|$, the unit vector $\vec{e}_{ij} = \frac{r_i - r_j}{|r_i - r_j|}$ represents the direction between particles i and j, and ξ_{ij} is a pairwise conserved Gaussian random variable. The weight functions ω_D and ω_R as well as the constants γ and ζ are related through fluctuation-dissipation relations

$$\omega_D = (\omega_R)^2, \tag{3.12}$$

$$\zeta = \sqrt{2\gamma k_{\mathrm{B}}T}, \tag{3.13}$$

ensuring the canonical equilibrium distribution. These releations were first derived by Español and Warren [211] In Equation 3.13, T denotes the temperature and k_{B} is the Boltzmann's constant.

The first term in the conservative force F_{ij}^{C} in Equation 3.12represents the conventional conservative force used in the DPD model, A denotes the magnitude of the force, and the weight function ω_{C} vanishes when the inter-particle distance r_{ij} is larger than a cutoff range r_c. The second term in F_{ij}^{C}, on the other hand, corresponds to the multi-body interaction term and the weight function ω_{B} vanishes for $r_{ij} > r_b$. The constants A and B are selected in such a way that $A < 0$ represents attractive interactions, while $B > 0$ corresponds to repulsive interactions. It is worth noting that in the conventional DPD model, $A > 0$ and $B = 0$. The weighted local density, denoted as ρ_i, is calculated by

$$\rho_i = \sum_{j\neq i}\omega_\rho(r_{ij}). \tag{3.14}$$

There are various approaches to selecting the weight function. In this thesis, the normalized

Lucy kernel in three dimensions, proposed by Lucy [213], is employed. The expression for the weight function $\omega_\rho(r_{ij})$ is given by

$$\omega_\rho(r_{ij}) = \frac{105}{16\pi r_{c\rho}^3}\left(1+\frac{3r_{ij}}{r_{c\rho}}\right)\left(1-\frac{r_{ij}}{r_{c\rho}}\right)^3, \tag{3.15}$$

where $r_{c\rho}$ represents the cutoff distance beyond which the weight function ω_ρ becomes zero.

In this thesis, simulations of molten CMAS, a mixture of calcia, magnesia, alumina, and silicate, was performed using the parameter mapping proposed by Koneru et al. [10]. The simulations were conducted using the open-source code LAMMPS [214].

Chapter 4

Machine learning based data-driven discovery of non-linear phase-field dynamics

The contents of this chapter have been published with the following citation: E. Kiyani, S. Silber, M. Kooshkbaghi, and M. Karttunen, Machine-learning-based data-driven discovery of nonlin- ear phase-field dynamics, Physical Review E, 106, 065303 (2022). American Physical Society, the publisher of our article, allows for the inclusion of the published article within this thesis.

4.1 Introduction

PDEs are widely used in modeling of complex physical, chemical and biological systems including fluid dynamics, chemical kinetics, population dynamics and phase transitions. The study of PDEs in the context of ML, broadly speaking, falls into two categories: 1) solving PDEs and 2) predicting unknown PDEs from data [108, 109, 110, 215, 216, 217, 114]. In simulations of phase-field and reaction-diffusion models, commonly used numerical techniques are

based on time and space discretization, such as finite difference and finite element methods. In recent years, a third approach based on ML has emerged with promising results for solving and even discovering unknown PDEs from data, see for example Ref. [125] and references therein.

The core idea for using ML algorithms to solve PDEs is representing the residuals of PDEs as a loss function of a neural network (NN) where the loss function is minimized; a loss function measures how far the predicted values are from their true values. This approach does not require discretization or meshing, which is beneficial when dealing with problems of high dimensions and/or complex geometries [108, 109, 218]. Since most deep learning frameworks are based on automatic differentiation, these methods are known as mesh-free approaches [219].

In the case of discovering unknown PDEs from data, the key idea of ML-based approaches is to estimate the time derivative of the desired (dependent) quantity. These approaches can be broadly categorized as follows:

1. An ensemble of macroscopic observations is available, and there is knowledge about the physics of the governing coarse PDE(s). The typical knowledge is that the time evolution of the field of interest depends on the field and its derivatives (e.g. Navier-Stokes equations). One can design the ML algorithm to find that dependency. This relation can be formulated based on any of the following methods: (i) Linear dependence of the field evolution using a dictionary of spatial derivatives with unknown coefficients [115, 116]; (ii) Non-linear dependence with black-box inference [114]; (iii) Non-linear dependence using a selective dictionary of spatial derivatives which were found by other data-driven approaches [117]; (iv) Non-linear dependence where spatial derivatives are informed by the memory (history) of the system using a feedback loop, e.g. recurrent neural network (RNN) together with long short-term memory (LSTM) and gated recurrent unit (GRU) [220, 221, 222].

2. An ensemble of microscopic observations is available, and the macroscopic field of interest is known. For example, the microscopic solutions of the Boltzmann equation are

available and one is looking for the time evolution of coarse fields such as density, velocity or temperature. Again, one can assume that the time evolution of the field depends on the spatial derivatives using physical intuition [223, 117].

3. An ensemble of microscopic observations is available, but the macroscopic field is unknown. Therefore, the first step is to discover the coarse-grained field, which is generally formulated as a model reduction problem [224]. The second step is to find the PDE(s) for the coarse variable(s). For example, Thiem et al. determined an order parameter for coupled oscillators using diffusion maps and the corresponding governing PDE using a Runge-Kutta network [225].

In this paper, we explore ML-based approaches which fall under the first category mentioned above. We assess two scenarios:

(i) There is an unknown relation between field evolution and its spatial derivatives.

(ii) The spatial derivatives, their orders and combinations are unknown (there is no spatial derivative dictionary).

Afterwards, we also solve the predicted PDEs in time and space. We should note that by discovering PDEs, we are referring to finding the aforementioned unknown relation implicitly.

For the first scenario, a flexible framework that can deal with large data sets and extract the unavailable PDE(s) from coarse-scale variables implicitly is developed. Two different approaches are presented for learning coarse-scale PDEs: 1) an MLP architecture and 2) a CNN-LSTM. Since LSTM only passes time information to its layers and misses the spatial features of previous time steps, CNN can be used to learn and detect the spatial features of the inputs [169, 226]. For the second scenario, a convolution operator is used to implicitly learn the dependence of the time derivative of the field on the spatial derivative(s) of unknown orders. The learned PDE is then marched in time with a time-integrator.

We demonstrate the capability of the above algorithms to learn PDEs using data obtained

from phase-field simulations of the well-known Allen–Cahn [194], Cahn–Hilliard [195], and the phase-field crystal (PFC) [5, 197] models using the open source software *SymPhas* [6].

4.2 Phase-field modeling

4.2.1 Phase-field modeling in a nutshell

Phase-field modeling provides a theoretical and computational approach for simulating non-equilibrium processes in materials, typically with the objective of studying the dynamics and structural changes. For an excellent overview of phase-field modelling, see the book by Provatas and Elder [141]. In its essence, phase-field modeling is a coarse-grained approach that uses continuum fields to describe slow variables such as concentrations. The continuum fields are given by order parameters which may be conserved or non-conserved. In this work, we consider only systems described by a single order parameter $U \equiv U(\vec{x}, t)$. In all of the descriptions below, we use the conventional dimensionless units [141].

The equations of motion for the non-conserved and conserved order parameters are (see Ref. [141] for a more detailed discussion) given as

$$\frac{\partial U}{\partial t} = -\Gamma \frac{\delta F}{\delta U} \qquad \text{(non-conserved)} \tag{4.1}$$

$$\frac{\partial U}{\partial t} = \Gamma \nabla^2 \frac{\delta F}{\delta U} \qquad \text{(conserved)}, \tag{4.2}$$

where F is a free energy functional and $\delta/\delta U$ is a functional derivative. We have neglected thermal noise from the above equations. The parameter Γ is a generalized mobility that is assumed to be constant, and is chosen based on a particular phase-field model. The free energy functional typically has the form

$$F = \int d\vec{x} \left[|\vec{\nabla} U|^2 + f(U) \right] \tag{4.3}$$

where $f(U)$ is a bulk free energy with a double well potential, which for this work we set to be

$$f(U) = \frac{a_4}{4}U^4 + \frac{a_2}{2}U^2. \tag{4.4}$$

It is also noteworthy that when no free energy functional is available, the equations of motion are often postulated. This is the case with reaction-diffusion systems, for example, the well-known Turing [227, 228] and Gray–Scott models [229, 228]. Phase-field models have been widely applied to various types of systems and phenomena, including dendritic and directional solidification [230, 231, 232], crystal growth [5, 233, 234, 235] and magnetism [236] as well as for phenomena such as fracture propagation [237, 238] to mention some examples.

4.2.2 Phase-field models used in the current work

We employed three different well-studied single order parameter phase-field models: 1) the Allen–Cahn model [194] for the case of a non-conserved order parameter, 2) the Cahn–Hilliard model [195] for the conserved order parameter, and 3) the phase-field crystal (PFC) model [5] that has a conserved order parameter and generates a modulated field that describes atomistic length scales and diffusive times. In addition to being well-studied, these models were chosen because they contain differing orders of spatial derivatives: the Allen–Cahn model is described using a 2nd order derivative, Cahn–Hilliard using 4th, and PFC using 6th. Moreover, they each exhibit various spatial patterns that evolve according to different time scales.

The Allen–Cahn Model

The Allen–Cahn model, a dynamical model for solidification originally developed by Allen and Cahn in 1972, has a single non-conserved order parameter corresponding to Equation (4.1), and is defined using the free energy of Equation (4.3). The equation of motion is then given as

$$\frac{\partial U}{\partial t} = -M\left(\nabla^2 U + a_2 U - a_4 U^3\right), \tag{4.5}$$

where Γ is set to M (see Equation (4.1)), a constant related to chemical mobility. The numerical values of the parameters M, a_2 and a_4 are given in Table 4.1. Equation ((4.5)) represents a physical system that evolves purely due to a chemical potential. It is also called Model A according to the Hohenberg and Halperin classification of phase-field models [196].

The Cahn–Hilliard Model

The Cahn–Hilliard model, formulated by Cahn and Hilliard in 1958, represents spinodal decomposition. It is a conserved order parameter model corresponding to Equation (4.2). Applying the free energy density of Equation (4.3) then gives

$$\frac{\partial U}{\partial t} = D\nabla^2 \left(\nabla^2 U + a_2 U + a_4 U^3 \right) , \tag{4.6}$$

where Γ is set to D (see Equation (4.2)), a constant that represents the diffusion constant. The parameters D, a_2 and a_4 are given in Table 4.1. Equation (4.6) is also known as Model B in the Hohenberg and Halperin classification [196].

The phase-field crystal model

A free energy density to describe a crystal lattice at an atomistic scale incorporating elastic effects into a phase-field model was originally developed by Elder et al. [5, 197]. The PFC free energy functional is minimized by a hexagonal periodic lattice, and can be defined [198] as

$$F(U) = \int d\vec{x} \left[\frac{U^3}{3} + \frac{U^4}{4} + U \left((q_0 + \nabla^2)^2 - \varepsilon \right) \frac{U}{2} \right] , \tag{4.7}$$

where q_0 and ε are constants. The equation of motion with a conserved order parameter is then determined using Equation (4.2) with $\Gamma = 1$ as

$$\frac{\partial U}{\partial t} = \nabla^2 \left(U^2 + U^3 + \left((q_0 + \nabla^2)^2 - \varepsilon \right) U \right) . \tag{4.8}$$

Table 4.1: The simulations were done on a uniformly discretized two dimensional grid of size $n_x \times n_y$ and $\Delta x = \Delta y = 1$. The simulations use a time step of Δt and continue to time t. All models use periodic boundary conditions, and initial conditions are populated using a uniform random distribution with values between -1 and 1 generated using the Mersenne Twister 19937 generator from the C++ standard library [11]. Additionally, all constants in the equations of motion are set to 1, except for ε used in the PFC model, which is set to 0.1. Later, in our phase field equations discovery, we used training sets with 60% (for MLP and CNN-LSTM) and 80% (for CNN) of the total snapshots ($n_k = 0.6n_t$ or $n_k = 0.8n_t$).

Phase-field model	$n_x \times n_y$	Δt	t	n_t	Equation parameters
Allen–Cahn, Equation (4.5)	256×256	0.1	20	100	$M = a_2 = a_4 = 1$
Cahn–Hilliard, Equation (4.6)	128×128	0.01	20	100	$D = a_2 = a_4 = 1$
PFC, Equation (4.8)	128×128	0.05	100	200	$q_0 = 1$ and $\varepsilon = 0.1$

The order parameter, U, represents the mass density. The PFC model can be used to describe elastic and plastic deformations in isotropic materials, i.e., crystal structures. The lattice structure can assume any orientation (based on the initial conditions) and interactions between grains (individual crystal structures) can lead to defects and dislocations.

4.2.3 Simulation of phase-field models

The open-source *SymPhas* [6] software package was used to numerically simulate the above three systems. *SymPhas* allows the user to define phase-field models directly from their PDE formulations, and control simulation parameters from a single configuration file. All simulations were done using dimensionless units [141]. In terms of the numerical solution, *SymPhas* has the ability to simulate models using either explicit finite difference methods or the semi-implicit Fourier spectral method. The latter was chosen. By virtue of its excellent error properties, the Fourier semi-implicit spectral method typically allows for larger time stepping than a finite difference solver [105]. For each of the models, five independent simulations with 100 frames of the field U were saved at equally spaced intervals. Parameters for the numerical simulations are summarized in Table 4.1. For models A and B, the simulation is stopped after the fast growth regime and the universal scaling of the system takes effect. The simulation of the PFC model was chosen to stop after approximately 10 diffusion times in order to

be well within the regime where the number of defects is slowly decreasing [5]. To illustrate the different dynamics of each model, snapshots at the end of each simulation are provided in Figure 4.1.

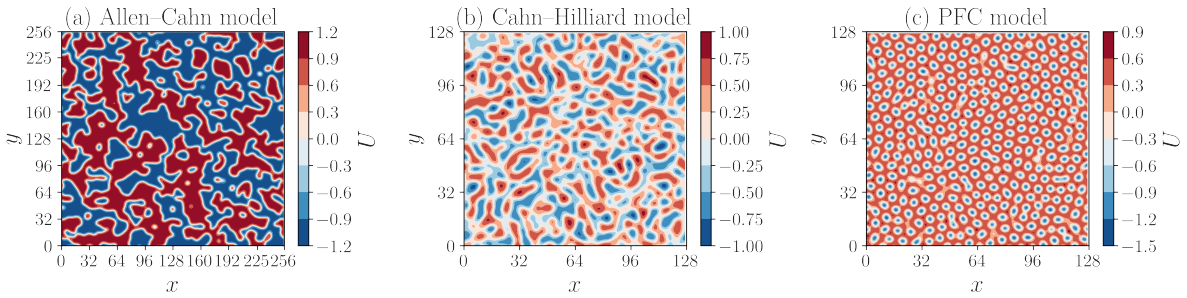


Figure 4.1: Snapshots from the three phase-field models. Field solutions for the Allen–Cahn model (Equation (4.5)) is shown on the left at $t = 20$, for the Cahn–Hilliard (Equation (4.6)) in the center at $t = 20$, and for the PFC (Equation (4.8)) on the right at $t = 100$. The longer simulation time of the PFC model is required to allow the number of initial defects to decrease [5]. The parameters of the numerical simulations are presented in Table 4.1. The vertical and horizontal axes display $x = n_x$ and $y = n_y$ respectively, and U represents the phase-field for the corresponding model, all in dimensionless units.

4.3 Data-driven PDEs with a spatial derivatives dictionary

As already discussed above, we consider two distinct types of methods for discovering PDEs from data, assuming the network is informed by spatial derivatives either explicitly or implicitly. In the first method, an MLP network is used to learn a function F_{MLP} that can be formulated as

$$U_t(t,x,y) = F_{\text{MLP}}\left(U(t,x,y), U_x(t,x,y), U_{xx}(t,x,y), U_y(t,x,y), U_{yy}(t,x,y), \ldots\right), \tag{4.9}$$

where $U_t(t,x,y)$ is the time derivative and $U_x(t,x,y), U_{xx}(t,x,y), U_y(t,x,y)$, and $U_{yy}(t,x,y)$ are the first and second spatial derivatives with respect to x and y, respectively.

In the second method, extending LSTM to a convolutional structure (CNN-LSTM) is used to learn an equation from local variables without giving spatial derivatives explicitly. Mathe-

matically, the network learns the time derivative $U_t(t,x,y)$ as a function of local macroscopic variables on a small square around each grid point,

$$U_t(t_k,x_i,y_j) = F_{\text{CNN-LSTM}}\Big(U(t_k,x_{i-1},y_j), U(t_k,x_i,y_j), \\ U(t_k,x_{i+1},y_j), U(t_k,x_i,y_{j-1}), U(t_k,x_i,y_{j+1})\Big), \tag{4.10}$$

where $U(t_k,x_{i-1},y_j)$, $U(t_k,x_i,y_j)$, $U(t_k,x_{i+1},y_j)$, $U(t_k,x_i,y_{j-1})$, and $U(t_k,x_i,y_{j+1})$ are the field values at the positions x_{i-1}, x_i, x_{i+1}, y_{j-1}, and y_{i+1}, respectively, and t_k corresponds to the time of the snapshots used in the training set for $1 \leq k \leq n_k$.

A schematic diagram of our framework for discovering PDEs with spatial derivatives dictionary is shown in Figure 4.2. Specifically, it shows how the spatial derivatives $(U, U_x, U_y, U_{xx}, U_{yy}, ...)$ and the local macroscopic variables $(U_{i-1,j}, U_{i,j}, U_{i+1,j}, U_{i,j-1}, U_{i,j+1})$ are fed through the MLP (Figure 4.2 (a)) and CNN-LSTM (Figure 4.2 (b)), respectively, to learn the time derivative $U_t(t,x,y)$.

4.3.1 Multi-layer perceptron network architecture and performance

An MLP is an example of a typical feedforward artificial neural network, consisting of a series of layers. Each layer calculates the weighted sum of its inputs and then applies an activation function to get a signal that is transferred to the next neuron [151].

In our MLP network, the number of layers, neurons, and the type of activation functions for each phase-field model was found by trial and error. We approximate spatial derivatives of the coarse variable U by finite differences, and along with U itself, feed this to the MLP network to learn the function F_{MLP} in Equation (4.9). The MLP architecture for learning the Allen–Cahn model (Equation (4.5)) is shown in Figure 4.2 (a) as an example. The five inputs $U(t,x,y), U_x(t,x,y), U_{xx}(t,x,y), U_y(t,x,y)$, and $U_{yy}(t,x,y)$ are passed to the first hidden layer, which is connected to the layers with $128/64/16/8$ neurons each. In the output layer, we use a dense layer with a single neuron to predict U_t. The network is trained for $2,000$

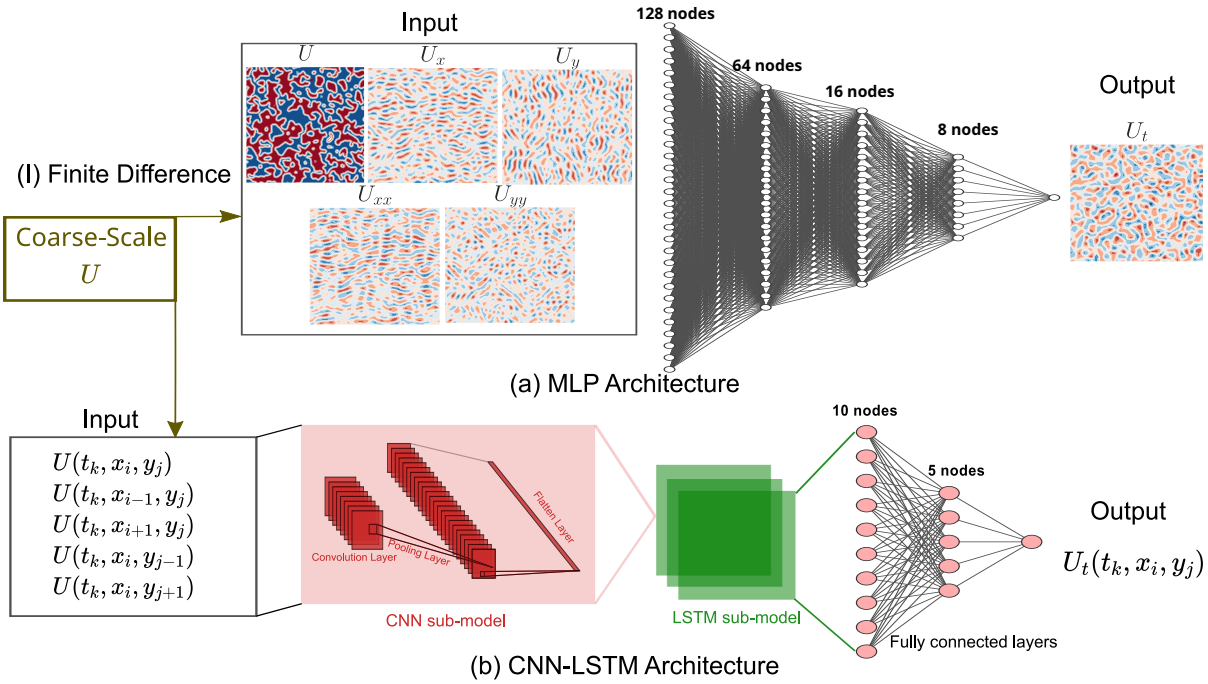


Figure 4.2: Schematic of the general steps in discovery of PDEs with a spatial derivatives dictionary. Learning of PDEs from spatial derivatives and local values of coarse variables using two different approaches, (a) MLP and (b) CNN-LSTM. Coarse-scale variables are collected as snapshots from the phase-field simulations. We used a 60:20:20 ratio to randomly choose the training, validation and test sets. Finite difference methods are used to approximate the spatial derivatives which are fed into panel (a) the MLP network according to Equation (4.9). The network connecting the input layer consists of a list of input features (the field U and its spatial derivatives) to the output layer of a single neuron (time derivative U_t). The values of the macroscopic field U evaluated around each grid point are fed through the panel (b) CNN-LSTM network to learn PDEs of the form Equation (4.10). CNN-LSTM network connecting the input layer consists of a list of input features (local variables $U(t_k, x_{i-1}, y_j), U(t_k, x_i, y_j), U(t_k, x_{i+1}, y_j), U(t_k, x_i, y_{j-1}), U(t_k, x_i, y_{j+1})$ for $1 \leq k \leq n_k$, $1 \leq i \leq n_x$, and $1 \leq j \leq n_y$) to the output layer of a single neuron U_t. Here n_k is the number of snapshots used for training which is a random set of n_t with size $n_k = 0.6n_t$. The corresponding values for n_t, n_x, and n_y are summarized in the Table 4.1.

epochs using the ADAM optimizer [12], rectified linear unit (ReLU) activation function [239], and mean squared error (MSE) as the loss function (see Table 4.2). For all three phase-fields models, the same architecture has been used. For the Cahn–Hilliard (Equation (4.6)) and PFC (Equation (4.8)) models, we used spatial derivatives up to fourth and sixth order for the input layers, respectively. The performance of the MLP network on learning the models is shown in

Table 4.2: MLP architecture for discovering phase-fields given in Equations (4.5), (4.6), and (4.8) consists of 4 dense layers with 128/64/16/8 neurons in each layer. The network was trained with learning rate of 10^{-3} for 2000 epochs. For each dataset, n_t snapshots wererandomly split into training, validation, and test with a 60:20:20 ratio (training set has n_k snapshots with a size of $0.6n_t$ for each dataset).

networks	layers	neurons	activation functions
MLP	4 dense layers	128/64/16/8	ReLU

Figure 4.3. The root mean squared error (rMSE) is the square root of MSE calculated as

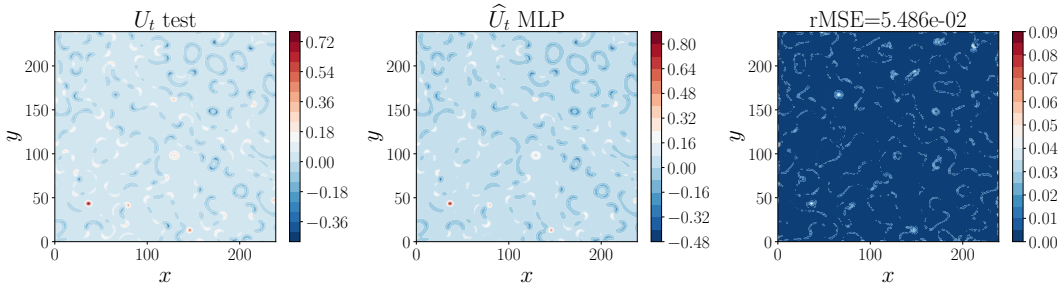
$$\text{MSE} = \frac{1}{n_x \times n_y} \sum_{i=1}^{n_x \times n_y} (U_t^i - \widehat{U_t^i})^2. \tag{4.11}$$

As shown in Figure 4.3, the rMSE values are small ($\sim 10^{-2}$), indicating that the target time derivatives ($\widehat{U}_t$) learned by the proposed MLP network are close to the true ones for all three models.

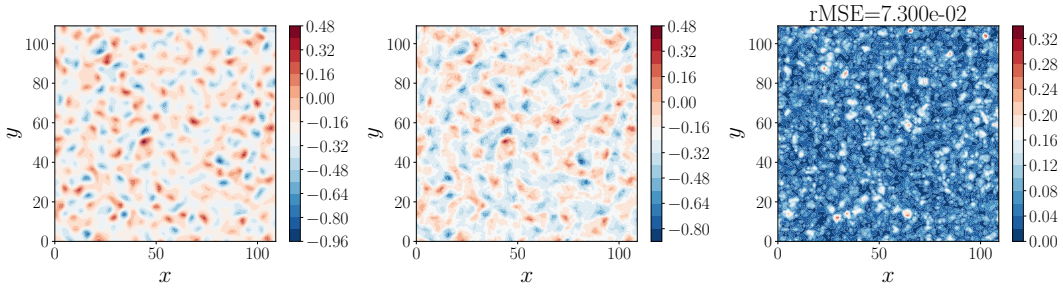
4.3.2 Convolution and long short-term memory (CNN-LSTM) Network Architecture and Performance

One of the main challenges in approximating coarse-scale PDEs is the estimation of spatial derivatives. While in previous studies PDEs have been successfully identified by learning time derivatives as a function of the estimated spatial derivatives, approximating derivatives remains challenging [240, 241]. Generally, the choice of the grid size is one of the most important considerations in numerical differentiation. While large step sizes can increase simulation speed, too large steps can create instabilities. On the other hand, if the steps are too small, numerical errors can dominate and the derivatives are of no use. Accordingly, the question that arises in discovering PDEs is the accuracy of numerical differentiation that has been used for training.

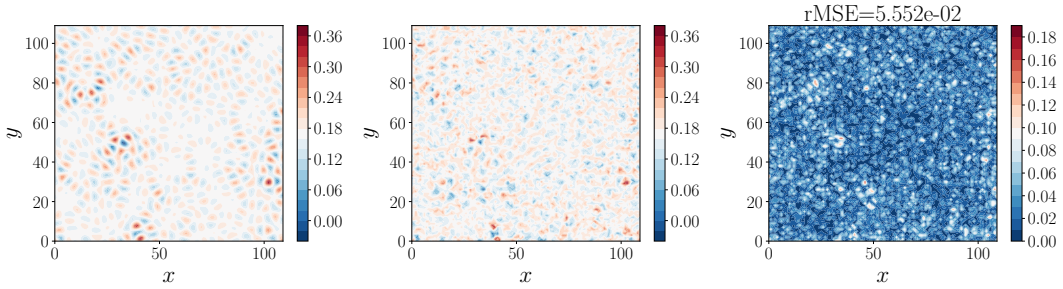
Unlike an MLP, CNN-LSTM is capable of automatically learning time derivatives from coarse-scale values. Using a combination of convolutional layers with other network struc-



(a) Allen–Cahn model



(b) Cahn–Hilliard model



(c) PFC model

Figure 4.3: Performance of the MLP network for predicting time derivatives of phase-fields given by Equations (4.5), (4.6), and (4.8). Allen–Cahn, Equation (4.5), as well as Cahn–Hilliard, Equation (4.6), were plotted at $t = 20$, and PFC, Equation (4.8), was drawn at $t = 100$. Left column shows U_t, the time derivative computed from the numerical solution generated by *SymPhas* [6], and the center column shows $\widehat{U}_t$, the learned time derivative. The right panel shows the difference between U_t and $\widehat{U}_t$, as well as the corresponding rMSE value for each phase-field model.

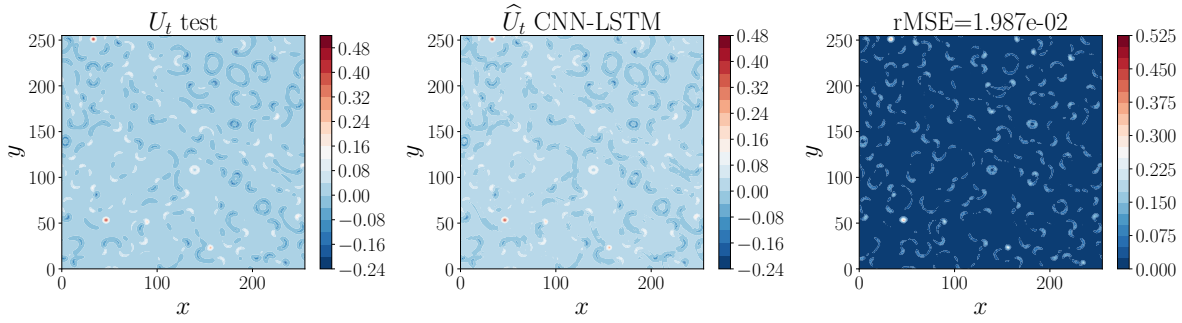
tures for data-driven differential equations is an active field of research (see, for example, Refs. [242, 243]). CNNs are widely used for image classification, and there have been several breakthroughs in image recognition with performance close to that of humans [170]. The CNN architecture can progressively extract higher level representations (color, shape, topology, etc.) of an input feature (image) and learn the dependency of the output (mostly a single class label)

Table 4.3: Details of the CNN-LSTM network used for field equation discovery. The network is trained for 2,000 epochs with learning rate 10^{-3}. n_t snapshots for each dataset are randomly split with 60:20:20 ratio for training, validation, and test (training set has n_k snapshots with size $0.6n_t$ for each dataset).

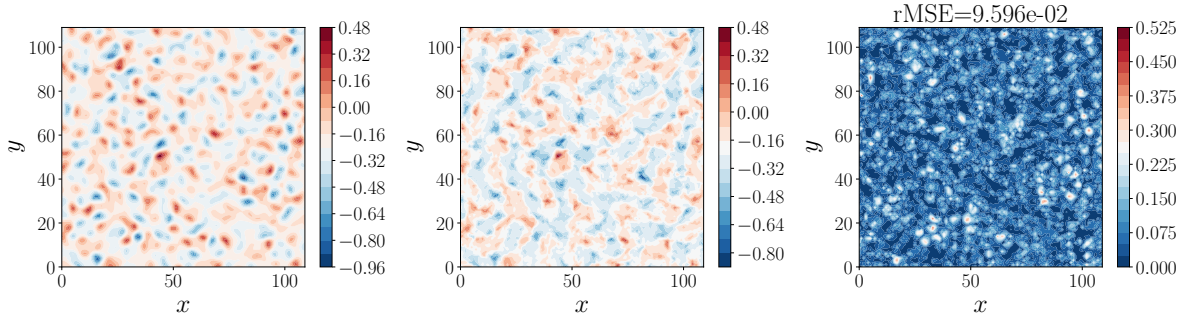
Layers	Structure	units	filter	kernel size	pool size	activation
0	Input	1	-	-	-	-
1	Conv1D	-	64	3	-	ReLU
2	TimeDistributed	-	-	-	-	-
3	MaxPooling1D	-	-	-	2	-
4	TimeDistributed	-	-	-	-	-
5	LSTM	80	-	-	-	ReLU
6	Dense	10	-	-	-	ReLU
7	Dense	5	-	-	-	ReLU
8	Dense (output)	1	-	-	-	Linear

on those representations. The convolution operation sweeps a filter across the entire input field and extracts the global features and local (pixel-to-pixel) variations. The convolutional layer can be considered as an efficient implementation of the convolution operator, hence, representing approximations of (potentially high order) derivatives of a scalar field. The relationship between the convolution-differentiation and derivatives-order of filters has been discussed in detail by Cai and Dong [244, 245].

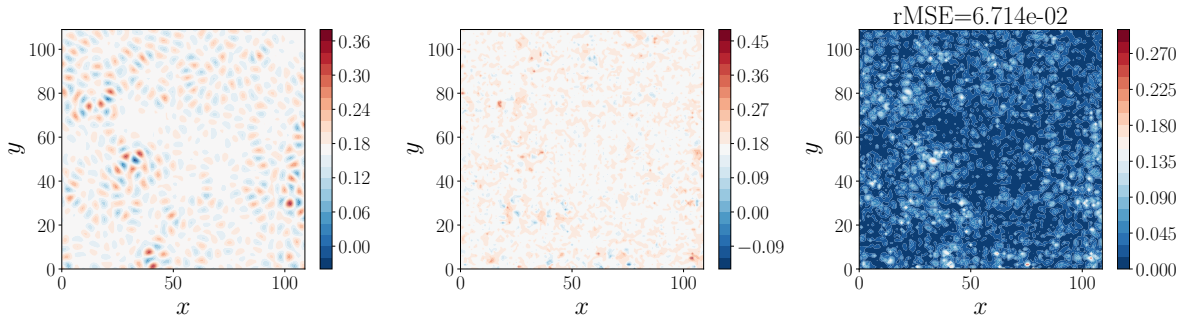
A schematic diagram of the proposed CNN-LSTM architecture is shown in Figure 4.2 (b). The architecture consists of two sub-networks: (i) A CNN sub-network, including one-dimensional convolution and maxpooling layers for feature extraction from input data and, (ii) a LSTM sub-network including sequential layers followed by one LSTM layer and two dense layers with ReLU activation. We feed the CNN-LSTM network with the five local coarse-scale variables, $U(t_k, x_{i-1}, y_j)$, $U(t_k, x_i, y_j)$, $U(t_k, x_{i+1}, y_j)$, $U(t_k, x_i, y_{j-1})$, $U(t_k, x_i, y_{j+1})$ tuple for $1 \leq k \leq n_k$, $1 \leq i \leq n_x$, and $1 \leq j \leq n_y$ for all phase-field models. Although we have second, fourth, and sixth order equations, CNN-LSTM training can be performed with only five local points (mentioned above). Due to CNN's ability to extract spatial features from inputs, CNN-LSTM shows that increasing training local points has no impact on performance. Here n_k correspond to 60% of the original datasets which is randomly selected for training. These coarse-scale variables at each grid point are fed into the CNN sub-network and the output of the convolutional layer passes through the LSTM layer followed by a dense layer to provide



(a) Allen–Cahn model



(b) Cahn–Hilliard model



(c) PFC model

Figure 4.4: CNN-LSTM predictions for (a) the Allen–Cahn Equation (4.5) at $t = 20$, (b) the Cahn–Hilliard Equation (4.6) at $t = 20$, and (c) the PFC Equation (4.8) at $t = 100$. Actual and learned time derivatives U_t and $\widehat{U}_t$ are shown in the left two panels. The difference between the predicted and the actual time derivatives as well as rMSE are presented in the right panel.

the output. The output of network is the single neuron approximating $U_t(t, x, y)$ at each grid point.

The LSTM network consists of a cell state which is the core concept of LSTM networks and memory blocks. Each block is composed of gates that can make decisions about which information passes through the cell state and which information can be removed. There are three kinds of gates: 1) input, 2) output, and 3) forget gate. Each memory block in an LSTM architecture has an input and an output gate which control information coming into the memory cell and information going out to the rest of the network, respectively. In addition, an LSTM architecture has a forget gate which contains an activation function and allows the LSTM to keep or forget information. Information from the previous hidden state and information from the current input is passed through the activation function. The output of each gate is a value between 0 (block the information) and 1 (pass the information) [162, 163].

In our setup, the network consists of a convolutional layer (Conv1D) with 64 filters before pooling layer, kernel size 3 followed by an LSTM layer with 80 neurons. There are two dense layers (fully connected) with 10 and 5 neurons each. Data has been reshaped to one dimension before training the network. The performance of the trained CNN-LSTM network is shown in Figure 4.4. In the left two panels, contours of U_t and $\widehat{U_t}$ for the test sets and the corresponding predictions by CNN-LSTM are compared. The snapshots of the differences between the true value and the predictions of the CNN-LSTM at time steps $t = 100$ are shown in the right panels, where rMSE is also reported for each phase-field model. A slight error can be observed along sharp boundaries in some isolated grid points, indicating CNN-LSTM does not identify smaller features in the field. Compared to true phase-field simulations, the spatial gradient of phase concentration is not as sharp. Similar behavior has been reported in other studies, see, e.g. Ref. [246]. The prediction errors from CNN-LSTM remain unchanged and an rMSE $\sim 10^{-2}$ is obtained for all three models.

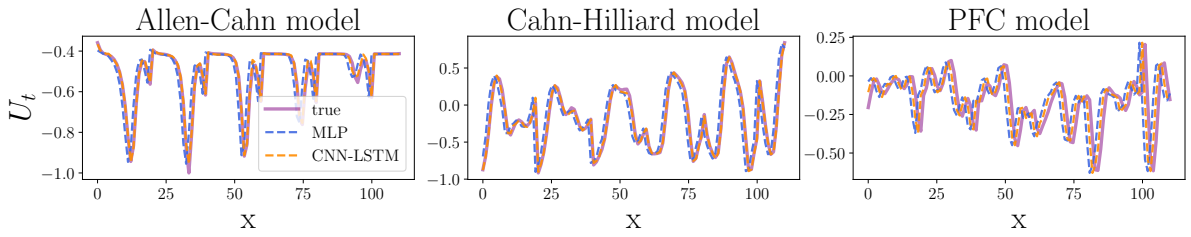


Figure 4.5: Comparisons between MLP and CNN-LSTM performance using the phase-field Equations (4.5), (4.6), and (4.8). In each plot, the horizontal axis indicates $x = n_x$ and the vertical axis represents the time derivative U_t and $\widehat{U_t}$ predicted by MLP and CNN-LSTM for each phase-field model. Two MLP and CNN-LSTM networks are trained and tested on the same data sets.

4.3.3 Hyper-parameter study

A comparison of regression results over the selected prediction period obtained by MLP and CNN-LSTM is shown in Figure 4.5. One can clearly see the ability of both MLP and CNN-LSTM to accurately reproduce the original data and make predictions of the phase-field models.

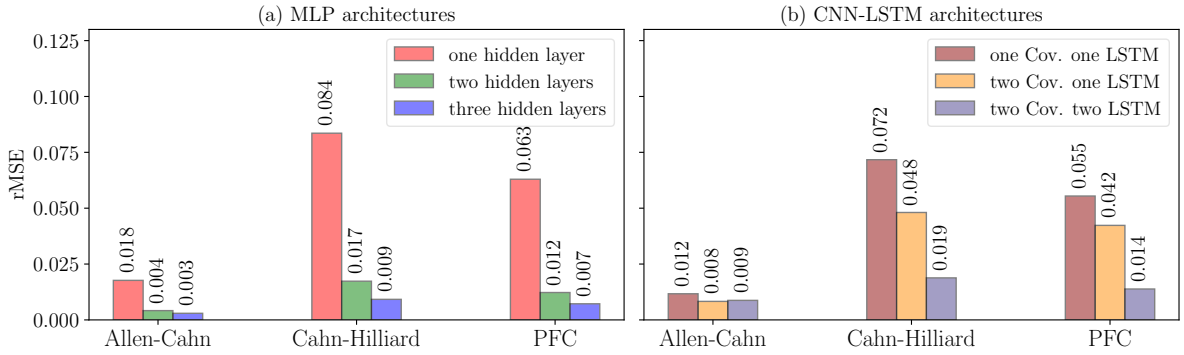
We used the coefficient of determination, R^2 to compare the performance of the networks,

$$R^2 = 1 - \frac{\sum_{i=1}^{n_x \times n_y} (U_t^i - \widehat{U_t^i})^2}{\sum_{i=1}^{n_x \times n_y} (U_t^i - \overline{U_t})^2}, \tag{4.12}$$

where $\overline{U_t}$ is the mean value of the time derivative for a single snapshot. Root mean squares and R^2 scores can be affected by different hyper-parameters such as learning rate, number of training epochs, and network depth and width. Here, we study the effect of adding/removing MLP and convolutional layers, while all the other parameters are fixed.

Figure 4.6 (a) shows the effect of adding layers to our MLP architecture. An MLP network with one layer consisting of 64 hidden neurons is expanded to a network with two and three layers with 128, 64 and 256, 128, 64 hidden neurons, respectively. It can be seen that adding hidden layers reduces the rMSE and increases the performance of prediction. Figure 4.6 (b) presents the effect of adding CNN and LSTM layers to the CNN-LSTM. Here, we use a single LSTM layer with two configurations for CNN layers: 1) single CNN layer with output filters

of size 64, 2) two CNN layers with 128, 64 output shape as well as two LSTM layers with 128, 64 neurons followed by two CNN layers with 128, 64 output sizes. Adding convolutional layers increases the performance. However, MLP networks are more sensitive to the choice of architecture than the CNN-LSTM networks. Moreover, the computational cost of training a multi-layer CNN-LSTM is huge compared to a single layer and should be taken into account for large-scale data. It can be roughly concluded that the optimal number of LSTM and CNN layers is 1 in our CNN-LSTM network. Conversely, the R^2 values show less sensitivity to the structural changes in our proposed neural networks, particularly in the CNN-LSTM network (see Figure 4.6 (c)).



(c) R^2 for MLP and CNN-LSTM

	Allen-Cahn	Cahn-Hilliard	PFC
MLP with one hidden layer	0.96	0.85	0.51
MLP with two hidden layers	0.99	0.99	0.98
MLP with three hidden layers	0.99	0.99	0.99
CNN-LSTM with one Cov. one LSTM	0.98	0.99	0.95
CNN-LSTM with two Cov. one LSTM	0.99	0.99	0.98
CNN-LSTM with two Cov. two LSTM	0.98	0.99	0.97

Figure 4.6: Effect of changing MLP and CNN-LSTM architectures on rMSE and R^2. (a) rMSE values obtained by three different MLP architectures, (b) rMSE values obtained by three different CNN-LSTM architectures. (c) R^2 values for the test set calculated by Equation (4.12) reported for three different MLP and CNN-LSTM architectures.

To further study the dynamics of the optimization process (training models), the MSE and mean absolute error (MAE) as a function of epochs are given in Figure 4.7. The MAE is the difference between the original and predicted values. This is calculated by averaging the

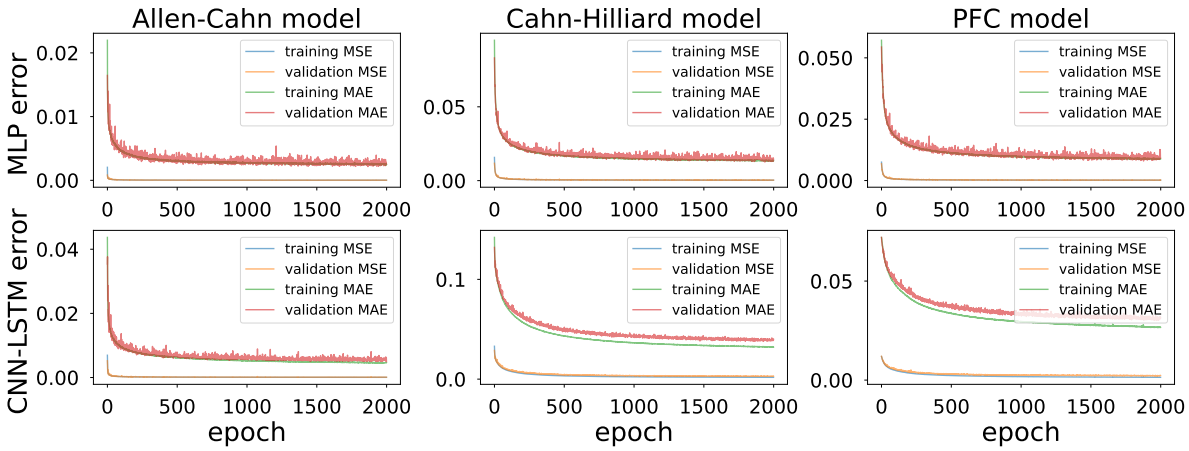


Figure 4.7: Trace of MSE and MAE (see Equations (4.11) and (4.13)) errors for MLP and CNN-LSTM networks. The blue and green lines represent the errors on the training sets as a function of epochs, and the orange and red lines correspond to the errors on the validation sets. Learning curves show that the training and validation curves are very similar for both MSE and MAE errors and they decrease to a point of stability.

absolute difference over the dataset and is expressed as

$$\text{MAE} = \frac{1}{n_x \times n_y} \sum_{i=1}^{n_x \times n_y} |U_t^i - \widehat{U_t^i}|. \tag{4.13}$$

In order to achieve sufficiently small error, we trained networks for 2,000 epochs with a batch size of 64. However, using approximately 500 epochs (e.g. early-stopping [247]) seems adequate for achieving optimal results, particularly for the Allen–Cahn and the Cahn–Hilliard models. Since training CNN-LSTM networks is computationally expensive, using smart early-stopping approaches can help in cases of large data PDE learning tasks.

4.4 Data-Driven PDEs without spatial derivatives dictionary

In this section, we reformulate the problem of learning PDEs as black-box supervised learning tasks, using convolutional neural network architecture where there is no selection of spatial derivatives and the field U is the only input to our deep learning model. The mathematical

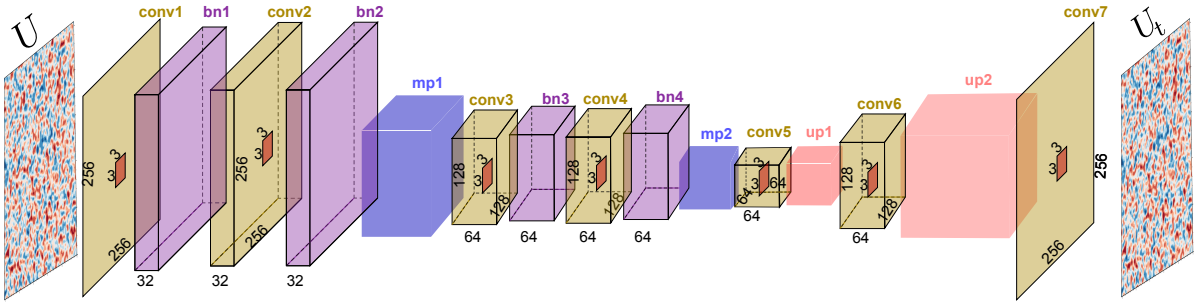


Figure 4.8: The proposed CNN architecture. The input and output of the CNN are the U and U_t fields, respectively. Input passes through several convolution (`conv`), batch normalization (`bn`), max pooling (`mp`) and up-sampling (`up`) layers. All the relevant parameters of the network architecture are described in Section 4.4.1.

representation of data-driven PDE learning task with CNN is

$$\begin{aligned} U_t(t,x,y) &= F_{\text{CNN}}\left(U(t,x,y)\right) \\ F_{\text{CNN}} &: \mathbb{R}^{n_x \times n_y} \rightarrow \mathbb{R}^{n_x \times n_y}, \end{aligned} \tag{4.14}$$

where n_x and n_y are the number of grid points in the x- and y-directions, respectively. We use U from our phase-field model simulations to train the CNN. After successful training of the CNN networks, arbitrary initial conditions were chosen for the field U and it was evolved in time by solving $U_t = F_{\text{CNN}}(U)$ numerically at each grid point.

4.4.1 Convolutional neural network (CNN) architecture

The CNN network architecture is illustrated in Figure 4.8. The full details of the mathematical operations and functionality of each layer are beyond the scope of this paper and can be found in reviews on CNNs such as the one by Rawar and Wang [167]. For particular applications where the desired outputs include localization (a class label is assigned to each pixel), a specific CNN architecture called "U-net" has been proposed [70]. Since in most engineering and physics applications, the time evolution of the scalar field depends on the local spatial derivatives, the U-net architecture is a reasonable candidate for such a learning task. The U-net-inspired network has also been successfully used in subgrid flame surface density estimation

Table 4.4: A CNN network used for discovering field equations without spatial derivatives. This network is trained for 20,000 epochs with the ADAM optimizer [12] with learning rate 10^{-4} and MAE loss function, Equation (4.13). A random sampling of 80% of snapshots (n_t) was used as the training set. Validation was performed on 10% and testing on the remainder.

Layers	Structure	filter	kernel size	pool size	activation	padding
1	Conv2D	32	(3,3)	-	ReLU	same
2	BatchNormalization	-	-	-	-	-
3	Conv2D	32	(3,3)	-	ReLU	same
4	BatchNormalization	-	-	-	-	-
5	MaxPooling2D	-	-	(2,2)	-	valid
6	Conv2D	64	(3,3)	-	ReLU	same
7	BatchNormalization	-	-	-	-	-
8	Conv2D	64	(3,3)	-	ReLU	same
9	BatchNormalization	-	-	-	-	-
10	MaxPooling2D	-	-	(2,2)	-	valid
11	Conv2D	64	(3,3)	-	ReLU	same
12	UpSampling2D	-	(2,2)	-	-	-
13	Conv2D	32	(3,3)	-	ReLU	same
14	UpSampling2D	-	(2,2)	-	-	-
15	Conv2D (output)	1	(3,3)	-	linear	same

for premixed turbulent combustion modeling [248].

The CNN structure proposed here, similar to the U-net [70, 248], resembles the encoding-decoding (auto-encoding) networks. The scalar field discretized on $n_x \times n_y$ grid points was fed as the input to the network. In the contracting path, two convolutional layers (`conv1, conv2` in Figure 4.8) with 32 filters each followed by ReLU and batch normalization (`bn1, bn2`) were applied. The kernel size was 3×3 for all the convolutional layers. After the `bn2` layer, the 2D max pooling operation (`mp1`) with zero stride (for dimensionality reduction purposes) was applied. The pool size for all the max pooling layers was 2×2. The same sub-structure is repeated with 64 filters (`conv3, bn3, conv4, bn4`) up to the bottleneck unit (output of `mp2`). The expansion path consists of two convolutional layers (`conv5,conv6`) with ReLU units, each followed by an upsampling layer (`up1, up2`) with the expansion factor of $(2, 2)$. Finally, at the last convolutional layer (`conv7`), a linear activation function was used with a filter of size one resulting in an output of shape $n_x \times n_y$. All the parameters used for the network are summarized in Table 4.4. The ADAM optimization was applied to find the parameters of the network, where the cost function is the mean absolute error between the network output and U_t from the training set. In total, our CNN network consists of 121057 trainable parameters.

Table 4.5: R^2 values for CNN performance of predicting U_t for test (unseen) data.

2D Model	Allen–Cahn Equation (4.5)	Cahn–Hilliard Equation (4.6)	PFC Equation (4.8)
R^2	0.98	0.975	0.985

4.4.2 CNN performance for learning PDEs

The phase-field models presented in Section 4.2 were used to evaluate the performance of the CNN network. For each model, the total of n_t two-dimensional U and U_t fields were used and randomly split 80:10:10 into training, validation and test sets, respectively. The U and U_t fields from training sets were provided as an input and output to the CNN. All models were trained for $20,000$ epochs and the performance of the network to recover the U_t (learning the RHS of a PDE) on the test sets is summarized in Table 4.5 in terms of R^2 values. The values indicate that all the trained models performed outstandingly in recovering the PDEs. The contours of U_t and the prediction of the CNN (for the Cahn–Hilliard model, Equation (4.6)) are compared in Figure 4.9 in the left two panels. The figure shows a qualitative agreement between the original and the data-driven models. In addition, the true and CNN predicted values of U_t for all the grid points for the test set are compared in the third panel (correlation plot). The data lie mostly on the diagonal line indicating good performance. The traces of the loss/cost functions during the training phase are also given in the rightmost panel of Figure 4.9. Similar results/plots were obtained for both the Allen–Cahn (Equation (4.5)) and the PFC (Equation (4.8)) models (data not shown here).

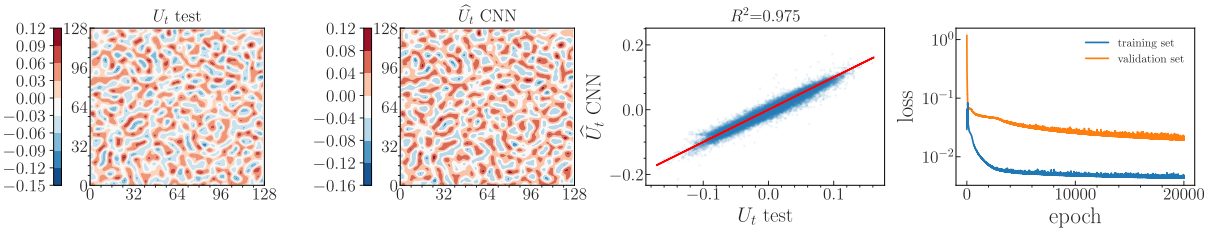


Figure 4.9: Results using the CNN model trained on the Cahn–Hilliard (Equation (4.6)) dataset. The left two panels show the color map of the U_t test set and the corresponding prediction by the CNN. The $\widehat{U}_t$ predictions for all test data as well as the traces of the loss functions are given in the right two panels.

4.4.3 Simulation of data-driven PDEs

In this section, the potential of the proposed method to predict the field U in time and space based on a given initial condition U_0 is presented. For all three phase-field models (Section 4.2), we are interested in solving a set of PDEs of the form

$$\begin{aligned} &\frac{\partial U(t,x,y)}{\partial t} = F_{\text{CNN}}\left(U(t,x,y)\right) \\ &U(0,x,y) = U_0; \quad \text{initial condition,} \end{aligned} \tag{4.15}$$

where the right hand side is the output (prediction) of the trained CNN networks. In the following, we used the U fields at $t = 2$ (simulation time) as the initial condition (U_0) for all the three models. The U field had $n_x \times n_y = 128 \times 128$ real values for the Cahn–Hilliard (Equation (4.6)) and the PFC (Equation (4.8)) models, and 256×256 for the Allen–Cahn model (Equation (4.5)). The different sizes were used to test if there is any size dependence. At each time t, the U_t values for every grid point were determined from our trained CNN models, and $n_x \times n_y$ ODEs (ordinary differential equations) were solved using the (stiff) integrator. We used the `scipy` Adams/BDF method with automatic stiffness detection and switching for time integration [249, 250]. Those ODEs were solved up to $t = 6$ in our benchmark datasets.

Figure 4.10 shows the solutions of the original and the data-driven PDEs. The color maps for U are given for qualitative comparison as well as the U values along the centerline $y = n_y/2$ for two snapshots at times $t = 2.2$ and $t = 6$. The results in Figure 4.10 showed that the data-driven PDEs learned by CNN approximate the original dynamics in both quantitative and qualitative manner.

Finally, we would like to emphasize the following points: 1) The explicit forms of the data-driven PDEs are not known and there is no obvious relation between the functional form of the original and the learned PDEs. Therefore, unlike with the phase-field models, there is no guarantee for existence and uniqueness for the learned PDEs. 2) There are some isolated points in which the U_t predicted values are different from the original models. This discrepancy

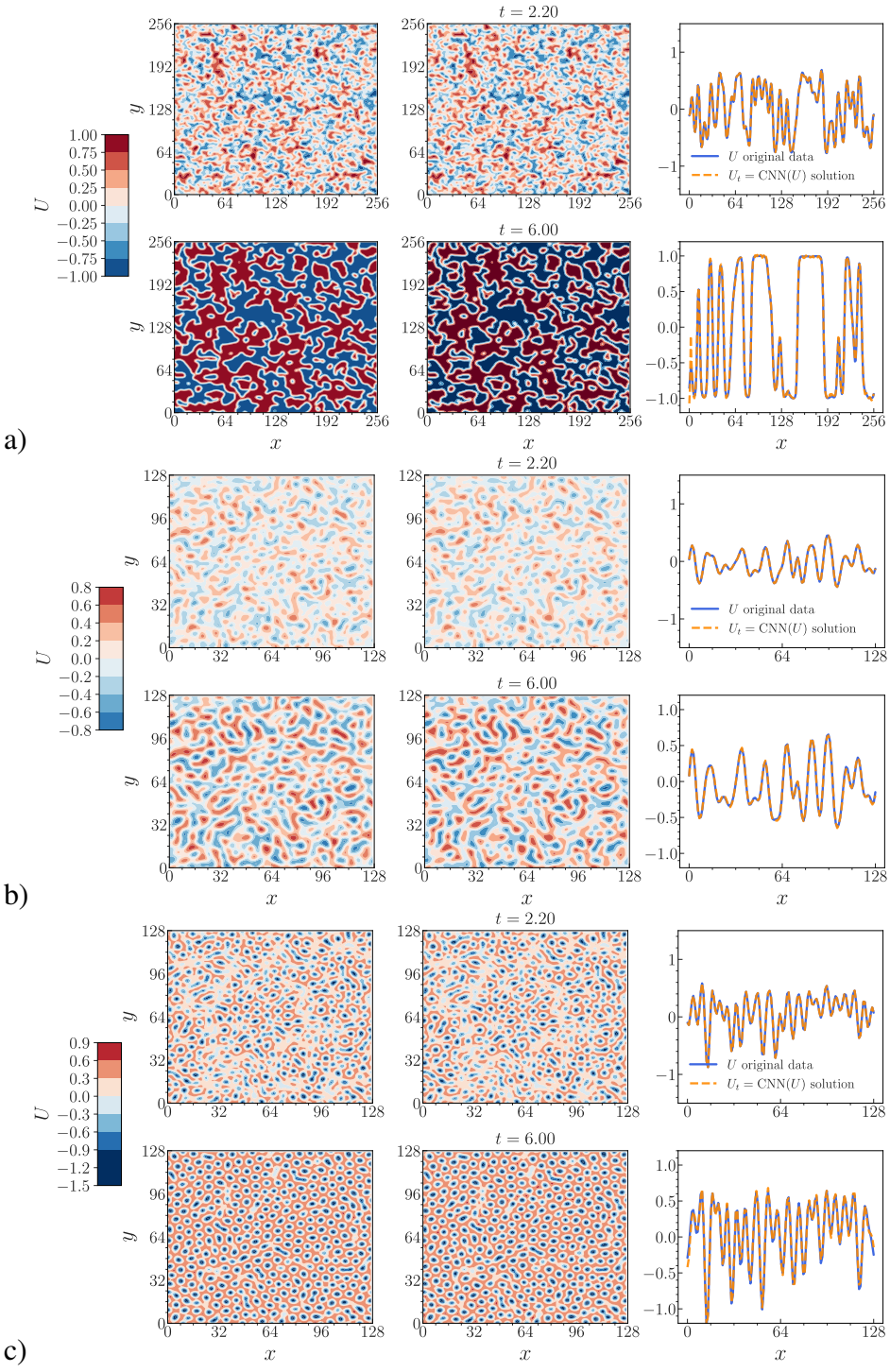


Figure 4.10: Time integration results of the PDEs learned by CNN for (a) Allen–Cahn (Equation (4.5)), (b) Cahn–Hilliard (Equation (4.6)) and (c) PFC (Equation (4.8)) at $t = 2.2$ and $t = 6$. Left panels: U field for original data. Middle panels: U field from simulations of the learned PDEs. Right panel: U values along the centerline $y = n_y/2$ for the original PDEs (solid lines) and from simulations of the learned PDEs (dashed lines).

propagates in time and space and can lead to finite time blow-up in simulations. This is a known issue in (almost all) machine learning algorithms for time series forecasting where there is no periodicity in time [251, 252]. In the case of no underlying periodicity, it may occur that the system trajectories do not span the whole phase space properly. Therefore, the observations do not properly represent the possible outcomes of the system and, hence, models trained with those data may not be adequate. Such a situation may limit the applicability of the approach to short simulation times.

4.5 Conclusion

We have presented several data-driven methodologies for discovering PDEs from phase-field dynamics. The well-known Allen–Cahn, Cahn–Hilliard and phase-field crystal models were used as the test cases to predict the underlying equations of motion.

First, we provide an MLP architecture to learn the PDEs where the spatial derivatives are explicitly computed by finite differences. Second, CNN-LSTMs were employed to learn the governing PDEs from coarse-scale local values. Third, we proposed a special CNN architecture for cases where there is no information about spatial dependence. In addition, using numerical integration, we showed how the learned PDEs can be used to predict coarse-scale variables as a function of time and space, starting from given initial conditions. The evolution of the learned and original PDEs showed excellent agreement. We emphasize that all of the above algorithms yield a black-box-type discovery of PDEs with no obvious connection to the functional form of the physical models.

In general, MLP networks are extremely flexible with data, and PDEs can be learned from various types of data using these networks. More specifically, they can be used to learn a mapping from a coarse field and its spatial derivatives as the inputs. However, the performance of an MLP network is greatly affected by the choice of architecture as shown in Section 4.3.3. Along with approximating derivatives, we need to know the derivatives' orders, as that is required to

train an MLP network.

In CNN networks, however, spatial derivatives are not required, and thus a CNN can be thought of as a finite-difference method capable of estimating derivatives in its first convolution layer. Moreover, one major advantage in using CNNs is their capability to extract spatial features from inputs. Since LSTMs pass only time information to the layers and keep the missing spatial information from the previous steps, a combination of CNNs and LSTMs can be applied more generally on data with spatial relationships, and, in the current case, to learn phase-field models. In spite of these advantages, CNN networks are memory intensive and require a large amount of data and several iterations in order to be trained effectively, and LSTMs are computationally expensive. Despite the above limitations, we believe that the techniques introduced here offer approaches that are both general and systematic, and provide a basis for future developments.

The study will be extended in two directions in the future: (a) predicting two-dimensional noisy phase-field models, (b) predicting three-dimensional phase-field models. As a result of a limited amount of memory, it becomes increasingly challenging to train networks efficiently in the second scenario. As a consequence, we will use frameworks that can handle large datasets.

Acknowledgments

Mahdi Kooshkbaghi was partially supported by NIH Grant GM133777. Mikko Karttunen thanks the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs Program. Computing facilities were provided by SHARCNET (www.sharcnet.ca) and Compute Canada (www.computecanada.ca).

Chapter 5

A Framework Based on Symbolic Regression Coupled with eXtended Physics-Informed Neural Networks for Gray-Box Learning of Equations of Motion from Data

The contents of this chapter have been published with the following citation: E. Kiyani, K. Shukla, G. Em Karniadakis, and M. Karttunen, A Framework Based on Symbolic Regression Coupled with eXtended Physics-Informed Neural Networks for Gray-Box Learning of Equations of Motion from Data, Computer Methods in Applied Mechanics and Engineering, 415, 116258 (2023)

5.1 Introduction

Partial differential equations (PDEs) are commonly used for modeling the evolution of dynamical systems in, e.g., fluid dynamics, heat transfer, financial derivatives, chemical reactions and phase tranformations. From the physical perspective, one of the main problems is constructing models that contain all the relevant information about the system at hand. This involves, for example, identifying the order parameters, relevant symmetries and possible couplings, and their nature, between the order parameters. Machine learning (ML) offers a method for automated construction of models directly from experimental or other data: it can be used to identify PDEs from data without prior/or only with partial knowledge about the underlying physics.

Being able to predict the PDEs from data involves training a neural network to recognize patterns in the data set(s), and then using the learned network to identify the underlying PDEs. Various methods have been proposed including PDE-Net [120, 121], neural networks [253, 122], Gaussian processes [107, 117], and the sparse identification of non-linear dynamics (SINDy) algorithm [254, 255, 256, 123, 124]. However, when dealing with sparse and high-dimensional data, these methods may not be sufficient to obtain a high level of accuracy. To circumvent these issues, Raissi et al. [114] proposed Physics-Informed Neural Networks (PINNs), which utilize a novel approach that incorporates physics-based constraints into the loss function allowing for accurate predictions in complex systems with varying initial and boundary conditions. PINNs are capable of discovering unknown equations and solving them [125].

Their leading idea is to integrate the fundamental physical principles of a system with neural networks. They are also flexible in the sense that they can handle complex and non-convex geometries, and different boundary and initial conditions. The loss function for a PINN incorporates data fitting, residuals of the PDEs (computed using automatic differentiation), initial and boundary conditions. The network parameters are updated during training to minimize the loss function, resulting in a solution that meets the constraints applied in the loss function.

One of the key benefits of using PINNs, as compared to other ML techniques, is their ability

to effectively learn from limited data, while also incorporating prior knowledge of the system being studied. This prior knowledge is incorporated into the loss function of the network, and it helps to guide the network towards solutions that are physically plausible and consistent with the known properties of the system being studied. This allows the network to make more accurate predictions even with limited training data. PINNs possess the ability to tackle both forward and inverse problems, and identifying unknown parameters in differential equations based on observed data. To achieve this, the available data is introduced into the network's training process as solutions to the PDEs, allowing it to learn and approximate the underlying physics of the system, and subsequently recognize the unknown parameters. This feature is beneficial in instances where the governing equations are only partially known, and the aim is to deduce the parameters based on gathered experimental or observational data [114, 257, 258].

Since the introduction of PINNs, various extensions such as Physics-Informed Attention-Based Neural Networks, (PIANNs) [175], Generative Adversarial Physics-Informed Neural Networks (GA-PINNs) [176], Graph Convolutional Networks (GCNs) [177], and Bayesian Physics-Informed Neural Networks (B-PINNs) [178] have been developed to enhance performance and to extend the applicability of PINNs to different problems. In this work, we focus on the eXtended Physics-Informed Neural Networks (X-PINNs) [127, 128]. They involve generalized space-time domain decomposition in order to provide computationally efficient solutions to PDEs across large spatial and temporal domains. In this approach, the domain is first split into smaller subdomains. Then, the PDEs are solved in each subdomain using PINNs and at the interfaces with certain continuity conditions imposed as soft-constraints in the loss function. This allows X-PINNs to use large neural networks without the common problem of overfitting. X-PINNs also reduce the computational cost associated with training due to their implicit concurrent implementation [259, 260]. X-PINN is an approach that involves dividing a computational domain into smaller subdomains. In this method, independent PINNs are deployed in each subdomain, allowing for parallelization and distributed computing. At the end of each epoch or iteration, the solution, flux, and the residuals of the PDEs at the interfaces

between the subdomains are communicated using point-to-point protocols. These protocols enable efficient and cost-effective communication between the subdomains.

In this study, we propose a framework combining X-PINNs with data-driven methods to uncover the non-linear term of the underlying PDE, while assuming the presence of a Laplacian term as a diffusion operator. The well-known Allen-Cahn model is used as the test case [194]. Our study presents a promising approach for gray-box learning of equations with X-PINNs. Gray-box learning refers to discovering equations when only partial knowledge of the equation is available and it combines the strengths of both white-box and black-box learning. It can be particularly useful in situations where the known parts of the equation provide valuable insights into the behavior of the system, despite the complete equation being unknown [148, 149]. For instance, in the current study, the Laplacian term represents the known part of the equation, while the non-linear term is unknown. After discovering the non-linear term of the gray-box Allen-Cahn equation using X-PINN, we feed the discovered term and the data from phase-field simulations into a symbolic regression model to predict the explicit mathematical formula of the unknown term. Symbolic regression is an ML technique used to discover the explicit mathematical expressions or equations that best fit a given dataset [261]. The framework is implemented using Python, and utilizes the Tensorflow (version 2.0) deep learning framework for its efficient automatic differentiation capabilities [262].

The rest of this article is structured as follows: Section 5.2 provides an overview of the phase-field approach and data preparation. Section 5.3 gives a brief summary of the PINNs and X-PINNs, followed by a presentation of X-PINN results and a comparison of the predictions using PINNs and X-PINNs. Section 5.4 presents the symbolic regression results. The performance of the framework for noisy data is discussed in Section 5.5. In Section 5.6, we examine the framework's performance for different sizes of training data sets to investigate the amount of optimal data required for training. Finally, a summary of the current work is provided in Section 6.7.

5.2 Phase-field modeling

In phase-field modeling the time evolution of the order parameter ($U(\vec{x}, t)$) is described by a time-dependent PDE [141]. The order parameter takes the value zero in the disordered, or high-temperature phase, and a small finite value in the ordered, or low-temperature phase. The order parameter field is called the "phase-field" and the order parameter itself can be a scalar, vector or even a tensor depending on the nature of the system at hand [263, 141, 264]. The typical way of constructing such models is by postulating a phenomenological Ginzburg–Landau free energy, or Lyapunov functional, in terms of a gradient expansion of the order parameter and taking a functional derivative [263, 141]. The terms that are included from the expansion must obey the symmetries of the system. In cases when a free energy functional cannot be constructed, one typically postulates an equation of motion. This is, for example, the case with reaction-diffusion models [265, 266]. Systems may also have several order parameters that are coupled, examples include such diverse systems as magnetocrystallinity [267] and cell migration [268]. Recently, open source software for phase-field simulations has also started to emerge, see e.g. Refs. [269, 270].

We utilize the well-known Allen–Cahn model [194, 271]. This model was first introduced by Stuart Allen and John Cahn in 1972 to describe solidification dynamics in binary alloys by employing a non-conservative scalar order parameter [194]. Its wide-ranging applications in solidification include, for example, dendritic growth and pattern formation. For broader discussions see, e.g. Refs. [271, 272, 273]. The model describes the evolution of a phase interface via a PDE that accounts for the thermodynamic driving force for phase separation, as well as kinetic effects arising from diffusion and surface tension. The equation of motion for the order parameter $U \equiv U(\vec{x}, t)$ can be given in a dimensionless form as [141]

$$\frac{\partial U}{\partial t} = -M \left(\nabla^2 U + a_2 U - a_4 U^3 \right) , \tag{5.1}$$

where the rate of change of the order parameter over time is determined by the mobility coeffi-

cient M. It is related to the interfacial energy, and it controls the speed of interface propagation and the wavelength of resulting patterns. Here we set $M = 1$. The constants a_2 and a_4 determine the shape and behavior of the free energy density of the system. We set $a_2 = a_4 = 1$ in our simulations. Figure 5.1 displays snapshots at three different time steps ($t = 0$, $t = 50$, and $t = 100$) to illustrate the dynamics of the Allen–Cahn model.

In addition to traditional numerical methods, recent developments in machine learning have shown promise in solving the Allen-Cahn equation as a forward problem. The PINN framework has been demonstrated to be highly accurate and efficient for solving the equation [114, 274, 275]. In addition, the Laplacian part has elliptic regularity, resulting in fast convergence of PINNs [276]. Another emerging field is physics-informed machine learning (PIML), which combines traditional numerical methods with ML to solve physical problems such as the Allen-Cahn equation. In other recent studies, data driven deep learning frameworks have been proposed and shown to significantly accelerate the traditional numerical methods for solving phase-field equations [277, 246, 253].

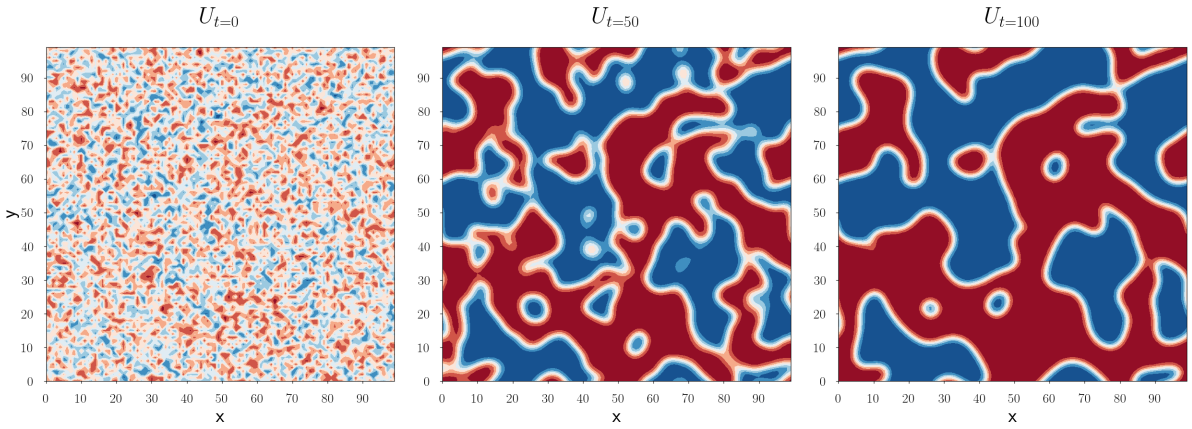


Figure 5.1: Snapshots from a simulation of the Allen–Cahn model, Equation (5.1), at $t = 0$, $t = 50$, and $t = 100$. The simulation was performed using dimensionless units, and on a uniformly discretized grid of size $n_x \times n_y = 100 \times 100$ with a spatial resolution of $\Delta x = \Delta y = 1.0$. A time step of $\Delta t = 0.1$ was used. Periodic boundary conditions were applied and the initial configuration was randomly generated from a uniform distribution.

Here, numerical simulations were conducted using a two-dimensional grid with dimensions $[n_x \times n_y] = [100 \times 100]$ and grid spacing of $[\Delta x, \Delta y] = [1, 1]$. The simulations were run

from $t = 0$ to $t = 10$ with a time step of $\Delta t = 0.1$, resulting in a total of $n_t = 100$ time steps. Periodic boundary conditions were applied and uniform random distribution was used for initial conditions. Figure 5.1 shows snapshots from a simulation starting from random initial conditions.

5.3 Extended physics-informed neural network (X-PINN)

PINNs are a type of ML algorithm that can accurately solve differential equations by incorporating the known physics (e.g., PDEs, ODEs, integro-differential equations) of the problem into a neural network architecture as soft constraints [114, 125, 278, 279]. The network is trained to minimize the loss functions constructed by computing the misfit between the data, initial and boundary conditions, and residuals of the PDE. To minimize the loss function, a first order optimizer (ADAM) or a combination of first and second order optimizer (ADAM + L-BFGS) is typically used [280, 281]; L-BFGS stands for the limited-memory Broyden–Fletcher–Goldfarb–Shannon algorithm. This approach allows PINNs to capture the underlying physics in the data and to make predictions. PINNs are particularly suited for modeling non-linear relationships, and handling multiple physical processes [279, 175, 282, 283, 284, 285, 286, 287, 278, 288].

In general, using a single PINN may not be sufficient to accurately capture chaotic, highly non-linear, and complex solutions [126].

Increasing the depth of the neural network may seem like an obvious solution, but it can lead to several issues. Firstly, using deeper networks with sparse data can result in over-parameterization, which can lead to over-fitting. This means that the network may fit the training data too closely and fail to generalize to new data [289, 290, 291].

Secondly, deep neural networks can result in a complex and highly non-convex loss landscape, which can make it difficult to optimize the network. This can lead to issues such as getting stuck in local minima and slow convergence [291, 292]. Lastly, using deeper networks

can be computationally intractable, which can limit their usefulness in practical applications. To circumvent these issues, Jagtap et al. [127] proposed X-PINNS which are primarily based on the domain decomposition approach frequently adopted in the classical numerical methods.

The X-PINN approach extends PINNs by offering space-time domain decomposition, which can be useful for problems with irregular, and non-convex geometries. X-PINNs aim to improve the accuracy and efficiency of solving and discovering PDEs by introducing additional physics-inspired constraints. In this approach, the domain is decomposed into smaller subdomains both in space and time, and separate neural networks are trained for each subdomain. The solutions at the spatio-temporal interfaces of the subdomains are unified by enforcing residual continuity. This allows for more efficient and accurate modeling of the physics in each subdomain, leading to an improved overall accuracy of the solutions.

As discussed in Section 5.2, we use the Allen-Cahn model as the test case. The underlying assumption is that the Laplacian term of Equation (5.1) is considered to be the known part of the equation and used as prior knowledge during the training of X-PINNs. Thus, the objective is to determine the function $F(U)$ such that

$$\frac{\partial U}{\partial t} = \nabla^2 U + F(U)\,. \tag{5.2}$$

The rationale behind considering the Laplacian operator is rooted in its extensive utilization and well-established significance in describing diffusion processes across diverse scientific and engineering domains. By assuming the Laplacian term as the known diffusion operator, we capitalize on existing knowledge and established physical theories that emphasize the significance of diffusion in the system under investigation. This approach allows us to focus on the reaction part, which is more complex and non-linear, and may involve intricate interactions and phenomena that are not explicitly captured by known physical laws or models.

Figure 5.2 shows a schematic diagram of the proposed framework to discover the unknown function $F(U)$ in Equation (5.2). To train the X-PINN, the domain Ω is divided into four

subdomains such that, Ω_{11} with $0 \leq x \leq 50$ and $0 \leq y \leq 50$, Ω_{12} with $50 \leq x \leq 100$ and $0 \leq y \leq 50$, Ω_{21} with $0 \leq x \leq 50$ and $50 \leq y \leq 100$, and Ω_{22} with $50 \leq x \leq 100$ and $50 \leq y \leq 100$.

In this study, X-PINNs are used with four sub-PINNs, each consisting of two sub-networks. The sub-networks $NN_{U_{i,j}}$, for i, j in $1, 2$, predict $U_{i,j}$ as a function of x, y, and t, and the sub-networks $NN_{F_{i,j}}$ are used to predict the corresponding $F_{i,j}(U)$. The network architectures for both sub-networks are shown in Table 5.1. The architectures of the networks $NN_{U_{i,j}}$ include six dense layers, each having 20 neurons, while the networks $NN_{F_{i,j}}$ comprise of four dense layers with 20 neurons in each layer. The networks were trained using a learning rate of 10^{-3} and the ADAM optimizer for $300,000$ epochs. The L-BFGS can be used as an optimizer when dealing with slow convergence rates and no mini-batch is required.

Each subdomain has $n_t = 100$ snapshots randomly split into training and test sets with an $80 : 20$ ratio.

Networks	# of layers	Layer type	Neurons in each layer	Activation function
$NN_{U_{i,j}}$	6	dense and fully connected	20	tanh
$NN_{F_{i,j}}$	4	dense and fully connected	20	tanh

Table 5.1: The neural network architectures in both sub-networks. Networks $N_{U_{i,j}}$ consist of 6 layers with 20 neurons in each layer. The $N_{F_{i,j}}$ networks are comprised of 4 layers and 20 neurons in each layer. The networks were trained with learning rate of 10^{-3} for $300,000$ epochs using the ADAM optimizer. Each subdomain has $n_t = 100$ snapshots were randomly split into training and testing with a $80 : 20$ ratio.

As shown in Figure 5.2, the first set of networks $N_{U_{i,j}}$ takes $x_{i,j}$, $y_{i,j}$, and t as inputs and predicts $\widehat{U}_{i,j}$ as a function of these inputs. The predicted $\widehat{U}_{i,j}$ is then fed through the second set of networks $N_{F_{i,j}}$ to discover the function $F(\widehat{U}_{i,j})$. Additionally, there are four interfaces $\partial\Omega_{11-12}$, $\partial\Omega_{11-21}$, $\partial\Omega_{12-22}$, and $\partial\Omega_{21-22}$, between two neighboring subdomains. $\partial\Omega_{11-12}$ and $\partial\Omega_{11-12}$ are depicted in Figure 5.2 as examples which $\partial\Omega_{11-12}$ is a common boundary between subdomains Ω_{11} and Ω_{12}, and has $x = 50$ and $0 \leq y \leq 50$. Therefore, each subdomain has two interfaces and two boundaries, marked by purple and green, respectively, in Figure 5.2.

It is important to note that in X-PINNs, the loss functions are defined separately for each

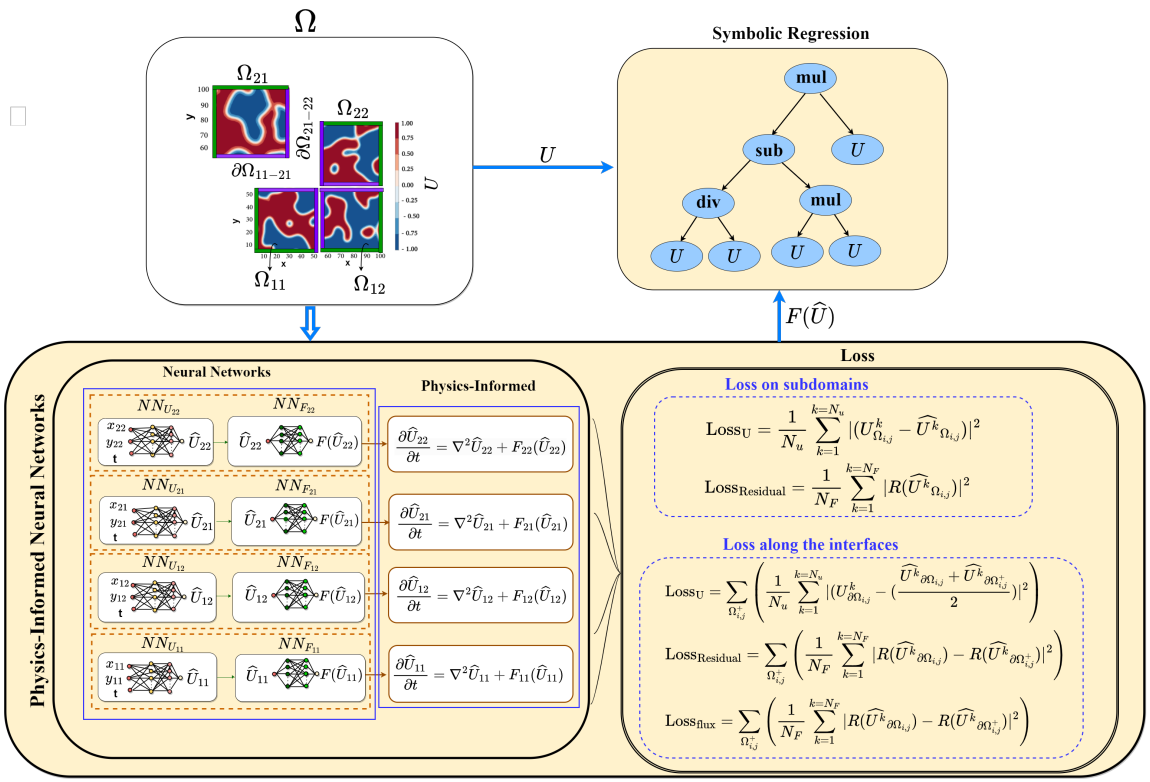


Figure 5.2: The X-PINN methodology for discovering the Allen–Cahn model with four subdomains, Ω_{11} ($0 \leq x, y \leq 50$), Ω_{12} ($50 \leq x \leq 100$ and $0 \leq y \leq 50$), Ω_{21} ($0 \leq x \leq 50$ and $50 \leq y \leq 100$), and Ω_{22} ($50 \leq x, y \leq 100$) involves several steps. Four sub-PINNs corresponding to the four subdomains are composed, each consisting of two sub-networks, NN_U and NN_F, and a physics-informed part. NN_U takes inputs x, y, and t at each subdomain to predict the output $\widehat{U}$. The output $\widehat{U}$ is then fed into a second network NN_F to predict the output $F(\widehat{U})$. Using the predicted $\widehat{U}$ and $F(\widehat{U})$, the physics-informed part creates Equation (5.2). The loss function is composed of two categories: 1) loss on subdomains and 2) loss along the interfaces, where Loss_U and $\text{Loss}_\text{residual}$ minimize data mismatch and residual on each subdomain, respectively. Additionally, the average solution continuity term and the residuals across the subdomain interfaces are included in the loss function, along with Loss_flux, which represents the normal flux continuity term. After minimizing the loss function, the next step involves feeding U and predicted $F(\widehat{U})$ into symbolic regression to predict the general form of F as a function of U.

subdomain. Each subdomain has the same terms as the standard PINN loss function, that is, a data-fitting term and the residuals of the PDE expressed in Equation (5.2). Let N_u and N_F denote the number of training and residual data points, respectively. The unknown function F

Algorithm 1 Pseudo-algorithm for gray-box learning of the Allen-Cahn equation

Require: U, x, y, t: sampled using Latin hypercube sampling
Require: N_d: Number of sub-domains to partition Ω in subdomains
Require: i_d: iterator for subdomains
Require: $\Theta_i = \{W_i, b_i\}_{i=1}^{N_d}$ ▷ Initialize trainable parameters
Require: ϵ: 10^{-5} ▷ Initialize training convergence criteria
Require: N: Total number of iterations ▷ Number of iterations
Require: Update ∈ { Adam, L-BFGS} ▷ Set of optimizers
Require: $\mathcal{F}_{\widehat{U}}(\boldsymbol{\Theta_i}), \mathcal{F}_{F_{\widehat{U}}}(\boldsymbol{\Theta_i})$ ▷ Initialize neural networks for each subdomain for U and $F(U)$
Require: $n,\ i_d,\ \mathcal{L}$: iteration counters and initial loss ▷ Initialize
  for all $i_d \in N_d$ **do** ▷ Loop 1: X-PINN Training
    while $\mathcal{L} > \epsilon$ and $n < N$ **do**
      $U_{i_d},\ F(\widehat{U})_{i_d} \leftarrow \mathcal{F}_{\widehat{U}}(\boldsymbol{\Theta_{i_d}}),\ \mathcal{F}_{F_{\widehat{U}}}(\boldsymbol{\Theta_{i_d}})$
      $\mathcal{L}_{i_d} \leftarrow \mathcal{L}_{U_{i_d}} + \mathcal{L}_{F(\widehat{U})_{i_d}} + \mathcal{R}(U)_{i_d}$ ▷ $\mathcal{R}$ is residual loss
      $\boldsymbol{\Theta}_{i_d} \leftarrow \text{Update}(\boldsymbol{\Theta})$
      $n \leftarrow n + 1$
    end while
    $i_d \leftarrow i_d + 1$
  end for
  $U_{i_d}, F(\widehat{U})_{i_d} \leftarrow$ Loop: 1
Require: $F(\widehat{U})_{i_d} : \alpha_{i_d} U_{i_d}^m + \beta_{i_d} U_{i_d}^n$
Require: $\epsilon_{rR_{Tol}}$
Require: $\epsilon_{Sr} = 100$ ▷ Tolerance for symbolic regression
  for all $i_d \in N_d$ **do** ▷ Loop 2: Symbolic regression
    while $\epsilon_{Sr} < \epsilon_{Sr_{Tol}}$ **do**
      $\{\alpha_{i_d}, \beta_{i_d}, m_{i_d}, n_{i_d}, \epsilon_{Sr}\} \leftarrow$ Symbolic Regressor$((U_{i_d}, F(\widehat{U})_{i_d}))$
    end while
  end for

can be obtained by minimizing the mean squared error loss

$$MSE_{U_{\Omega_{i,j}}} = \frac{1}{N_u} \sum_{k=1}^{k=N_u} |(U^k_{\Omega_{i,j}} - \widehat{U^k}_{\Omega_{i,j}})|^2, \tag{5.3}$$

$$MSE_{\text{Residual}_{\Omega_{i,j}}} = \frac{1}{N_F} \sum_{k=1}^{k=N_F} |R(\widehat{U^k}_{\Omega_{i,j}})|^2, \tag{5.4}$$

where $MSE_{U_{\Omega_{i,j}}}$ is a data-fitting term, and $U_{\Omega_{i,j}}$ and $\widehat{U}_{\Omega_{i,j}}$ are the simulation data and the predicted solution by sub-PINNs over subdomain $\Omega_{i,j}$. $MSE_{Residual_{\Omega_{i,j}}}$ is the loss for the PDE residual that enforces the governing Equation (5.2) over subdomains. For the predicted $\widehat{U}_{\Omega_{i,j}}$ it

is defined as

$$R(\widehat{U}_{\Omega_{i,j}}) = \frac{\partial \widehat{U}_{\Omega_{i,j}}}{\partial t} - \nabla^2 \widehat{U}_{\Omega_{i,j}} - F(\widehat{U}_{\Omega_{i,j}}) \,. \tag{5.5}$$

Subsequently, each subdomain's loss function is modified to enforce flux and residual continuity at the interface. This binds the subdomains together and ensures unique solutions at the interface between two neighboring subdomains.

For any method based on the PINN framework, there are three types of errors: estimation error, approximation error and optimization error. The generalization error is the addition of approximation and estimation errors. The approximation error is very well understood as single layer with sufficiently large number of neurons that can approximate the function to an arbitrary level of error. However, the estimation error is caused by a finite number of the data points and converges with the rate of $\propto \frac{1}{\sqrt{n}}$ [291]. The optimization error, however, is poorly understood as the objective function is high-dimensional, highly non-convex and non-linear.

Optimization often involves many engineering tricks and tedious trial and error type fine-tuning of parameters such as adjusting the learning rate, usage of learning rate scheduler, type of optimizers e.g., SGD [293], Adam, RProp [294] etc. The imposition of flux or residual continuity is bounded from above by the optimization error and can be viewed as $O \leq \mathcal{F}(u^+, u^-)$, where O represents the optimization error and $\mathcal{F}$ is the flux or residual computed at the interface. A detailed study of convergence of PINN for parabolic and elliptic PDEs was done by Shin et al. [276]. The convergence of X-PINN specifically for generalization was studied in detail by Hu et al. [295]. Also the error bounds for PINN in solving the incompressible Navier-Stokes equation have been shown by De Ryck et al. [296], wherein they proved

- Smallness of the PDE residual in the class of neural networks.
- A small residual contributes to a small total error.
- Small training errors imply small total errors for sufficient number of quadrature points.

It is to be noted that in the original implementation of X-PINN [127, 258], only continuity of the solutions and residuals are enforced at the interfaces of two neighboring subdomains.

In this study, we augment the loss function with flux continuity along with continuity of the residuals and the solution. This enables the model to consider physical constraints that are not explicitly included in the training data. By doing so, the model's accuracy and the rate of convergence are improved, enabling it to determine more precise predictions regarding the interface behavior. The loss terms representing the continuity of solutions, residual and flux at the interface of two subdomains are expressed as

$$MSE_{U_{\partial\Omega_{i,j}}} = \sum_{\Omega^+_{i,j}} \left(\frac{1}{N_u} \sum_{k=1}^{k=N_u} \left| U^k_{\partial\Omega_{i,j}} - \left(\frac{\widehat{U^k}_{\partial\Omega_{i,j}} + \widehat{U^k}_{\partial\Omega^+_{i,j}}}{2} \right) \right|^2 \right), \tag{5.6}$$

$$MSE_{\text{Residual}_{\partial\Omega_{i,j}}} = \sum_{\Omega^+_{i,j}} \left(\frac{1}{N_F} \sum_{k=1}^{k=N_F} |R(\widehat{U^k}_{\partial\Omega_{i,j}}) - R(\widehat{U^k}_{\partial\Omega^+_{i,j}})|^2 \right), \text{ and} \tag{5.7}$$

$$MSE_{\text{flux}_{\partial\Omega_{i,j}}} = \sum_{\Omega^+_{i,j}} \left(\frac{1}{N_F} \sum_{k=1}^{k=N_F} \left| \left(\frac{\partial \widehat{U^k}_{\partial\Omega_{i,j}}}{\partial x} + \frac{\partial \widehat{U^k}_{\partial\Omega_{i,j}}}{\partial y} \right) \cdot \hat{\boldsymbol{n}} - \left(\frac{\partial \widehat{U^k}_{\partial\Omega^+_{i,j}}}{\partial x} + \frac{\partial \widehat{U^k}_{\partial\Omega^+_{i,j}}}{\partial y} \right) \cdot \hat{\boldsymbol{n}} \right|^2 \right). \tag{5.8}$$

The term $MSE_{U_{\partial\Omega_{i,j}}}$ represents the data-fitting term along the interfaces, $U_{\partial\Omega_{i,j}}$ denotes the simulation data, and $(\frac{\widehat{U^k}\partial\Omega i,j+\widehat{U^k}\partial\Omega i,j^+}{2})$ is the average of the solutions along the interfaces predicted by two different networks on subdomains $\Omega_{i,j}$ and $\Omega_{i+1,j}$ or $\Omega_{i,j+1}$. Here, $\partial\Omega^+_{i,j}$ refers to the interfaces between subdomains. Furthermore, $MSE_{\text{Residual}_{\partial\Omega_{i,j}}}$ and $MSE_{\text{flux}_{\partial\Omega_{i,j}}}$ represent the residual continuity conditions and fluxes across common interfaces, respectively. The residuals and fluxes at the interfaces are calculated by two different networks on $\Omega_{i,j}$ and other connected subdomains $\Omega_{i+1,j}$ or $\Omega_{i,j+1}$. For the Allen-Cahn (Equation (5.1)), the fluxes in the x- and y-directions are $\frac{\partial \widehat{U}}{\partial x}$ and $\frac{\partial \widehat{U}}{\partial y}$, respectively, and $\hat{\boldsymbol{n}}$ denotes the direction of the outward normal to the interfaces.

It is important to highlight that we define the following loss function:

$$\text{Loss} = W_{U_{\Omega_{i,j}}} \text{MSE}_{U_{\Omega_{i,j}}} + \text{W}_{R_{\Omega_{i,j}}} MSE_{\text{Residual}_{\Omega_{i,j}}} + W_{U_{\partial\Omega_{i,j}}} \text{MSE}_{U_{\partial\Omega_{i,j}}} + W_{R_{\partial\Omega_{i,j}}} \text{MSE}_{\text{Residual}_{\partial\Omega_{i,j}}} + W_{F_{\partial\Omega_{i,j}}} \text{MSE}_{\text{flux}_{\partial\Omega_{i,j}}}.$$

The weights assigned in our approach play a crucial role in achieving convergence for the minimizer. In our method, we utilize the following values for the weights: $W_{U_{\Omega_{i,j}}} = 20$, $W_{R_{\Omega_{i,j}}} = 1$, $W_{U_{\partial\Omega_{i,j}}} = 20$, $W_{R_{\partial\Omega_{i,j}}} = 20$, $W_{F_{\partial\Omega_{i,j}}} = 20$. Based on our numerical experiments, we have observed that increasing the weights results in accelerated convergence.

In comparison to conservative physics-informed neural networks (CPINNs) [126], our approach introduces an additional loss term at the interface between subdomains. While CPINNs emphasize enforcing flux continuity and the average solution in their loss function, our method incorporates the continuity of the residuals term to further improve the accuracy and consistency of the model's predictions at the subdomain interface. Furthermore, CPINN has been utilized to uncover the unknown parameters of equations in white-box learning. However, in the current study, to fulfill the requirements for gray-box learning, an additional network has been incorporated into each sub-PINNs to train the unknown term $F(U)$.

Figure 5.3 displays the performance of the trained sub-PINNs. The left panels in Figure 5.3 (a) show the contours of the predicted $\widehat{U}$ at time $t = 100$, while the middle panels in Figure 5.3 (b) show snapshots of the exact solution of U. The right panels in Figure 5.3 (c) illustrate the point-wise relative errors. Moreover, Figures 5.3 (d) and (e) depict a comparison between the true values with the predicted values of U using X-PINNs and PINNs, the vertical axis represents U and the horizontal axis the x-coordinate. The results indicate that the sub-networks in X-PINNs are effective at accurately capturing the primary characteristics of the solution. In contrast, the predictions obtained using PINNs are significantly divergent from the true values, suggesting that PINNs may not be a suitable option for predicting this particular equation. However, some slight errors are present in isolated grid points near sharp boundaries, indi-

cating that the network may not detect smaller features. Such behavior has been previously reported in other studies [246, 253].

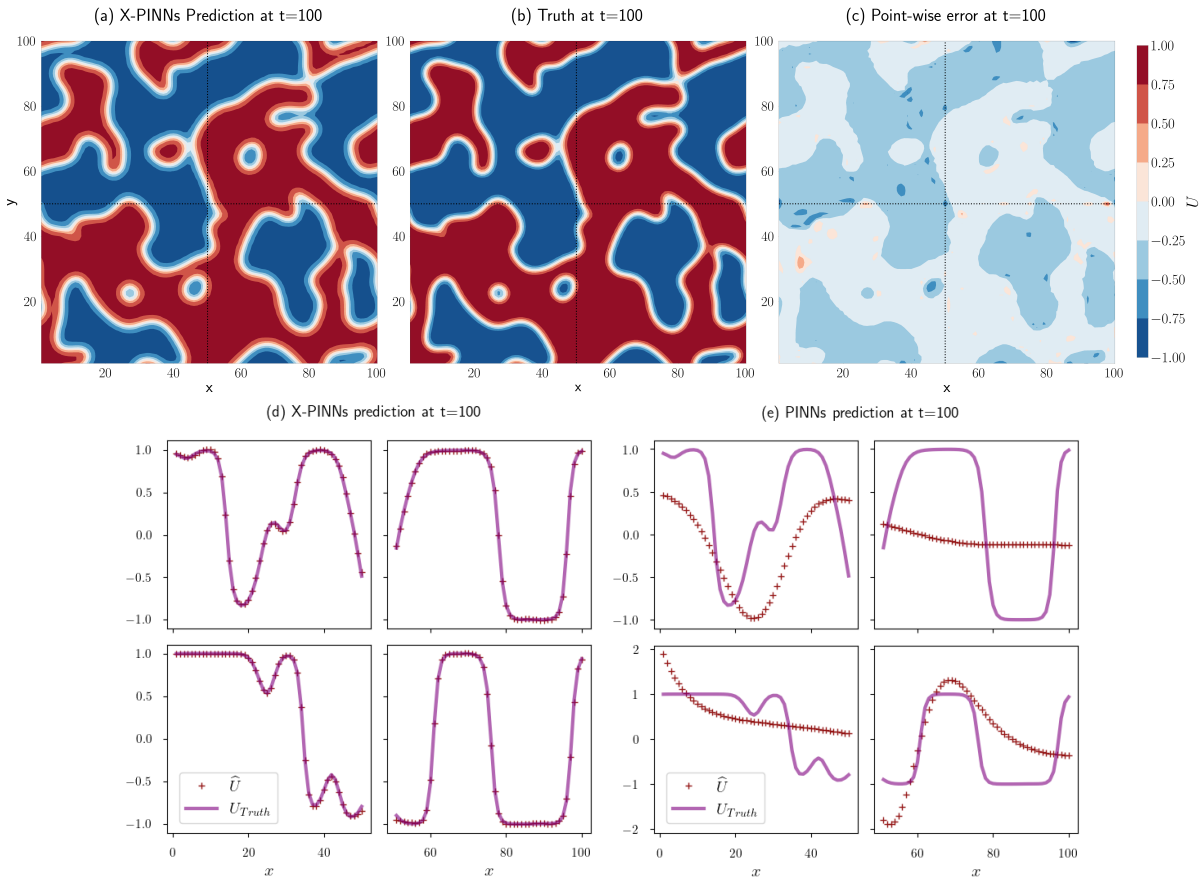


Figure 5.3: Snapshots of (a) PINNs predictions $\widehat{U}$, (b) the true solution of the Allen–Cahn Equation (5.1) at time $t = 100$, and (c) the point-wise relative errors. A comparison of the predictions from X-PINNs and PINNs frameworks with true values of U is shown in (d) and (e), respectively. It is worth noting that the plots were generated specifically for the value of $y = 50$. This positioning corresponds to one of the interfaces, specifically at $y = 50$ and x ranging from 0 to 100.

Figure 5.4 displays the predicted $F(\widehat{U})$ at time $t = 100$. The left panels in Figure 5.4 (a) show the contours of $F(\widehat{U})$, while the right panels in Figure 5.4(b) depict $U - U^3$. The results demonstrate that PINNs in each subdomain can accurately identify function $F(U)$ with a high degree of accuracy. Figure 5.4 (c) presents a visual comparison between the predicted and exact $F(U) = U - U^3$. The vertical axis represents $F(U)$, and the horizontal axis the x-coordinate. These results illustrate that the proposed framework can uncover the unknown part of the non-

linear Equations (5.2) with a great accuracy, despite using no information about the function F during model training.

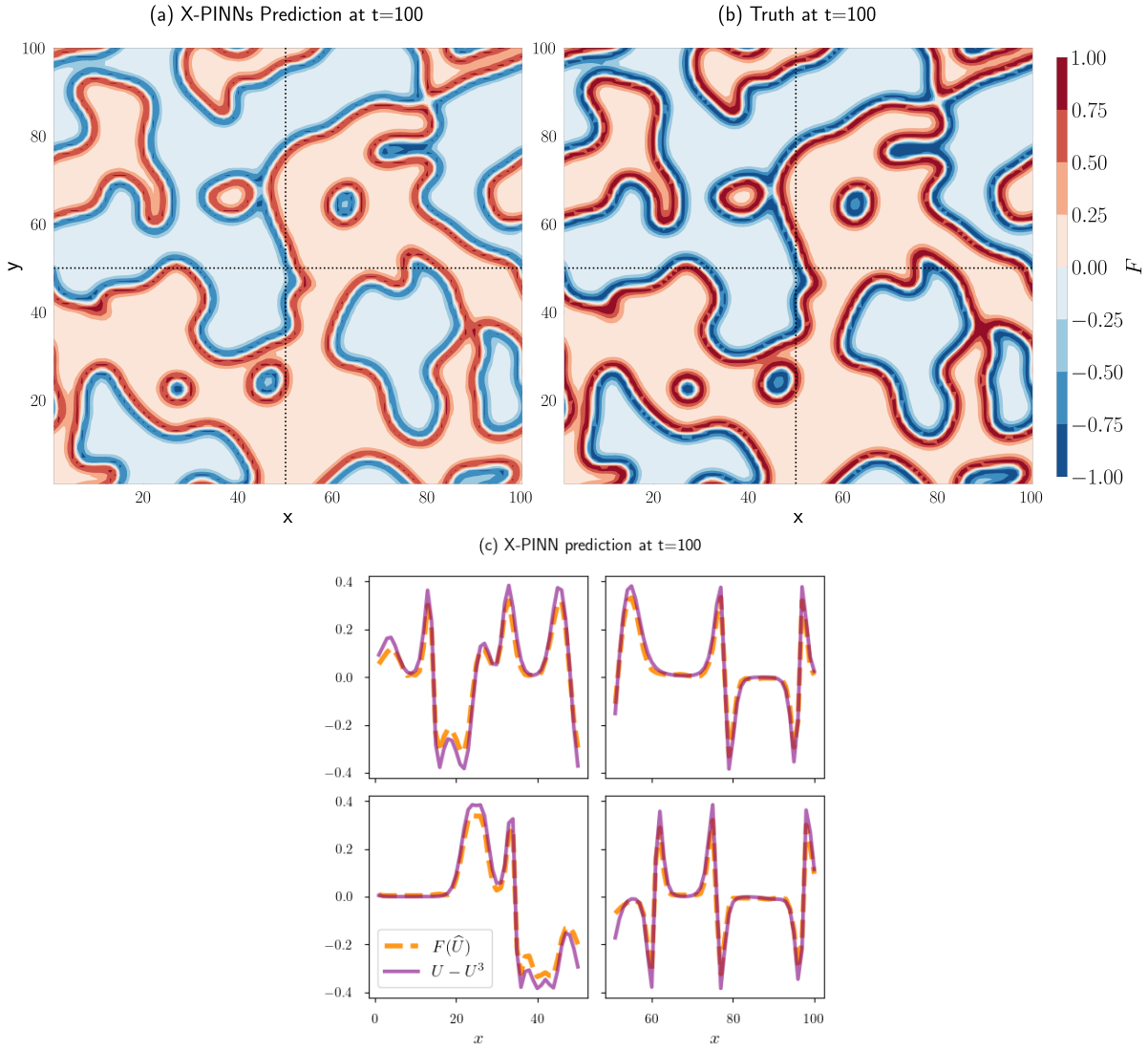


Figure 5.4: Snapshots of (a) predicted unknown function $F(\widehat{U})$, (b) the exact $F(U) = U - U^3$ at time $t = 100$ as well as (c) a comparison of the predicted $F(\widehat{U})$ and exact $U - U^3$, where the vertical axis represents $F(\widehat{U})$ and $U - U^3$ and the horizontal axis is the x-coordinate. It is important to mention that the plots were specifically generated for the value of $y = 50$

.

The primary reason behind the underperformance of PINN compared to X-PINN is spectral bias [174, 297, 298, 299].

Spectral bias in neural network causes it to perform better with good convergence for low frequencies/wavenumbers over high frequencies/wavenumbers. The solution of the Allen-

Cahn equation comprises broad range of wavenumbers associated with respective the energy modes of the solutions, and they are difficult to recover for a single PINN due to limited expressive power of a single neural network. X-PINN, however, comprising multiple PINNs, offers a better representation of the solution by splitting the domain into various subdomains. In this way, each subdomain is represented by a subset of the global wavenumbers and offering rapid to accurate solution.

To evaluate the accuracy of the predictions, we use the Frobenius matrix norm [300] to measure the errors between the predicted $\widehat{U}$ and $F(\widehat{U})$, and the exact values of U and $U - U^3$ in $\mathbb{R}^{100\times100}$. The Frobenius matrix norm is commonly used in linear algebra, numerical analysis, and ML. It has several useful properties, including being invariant under orthogonal transformations and sub-multiplicative, similar to the magnitude of a vector. The Frobenius norm for an $m \times n$ predicted matrix $\widehat{U}$ is defined as

$$\|\text{error}_U\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |U_{i,j} - \widehat{U}_{i,j}|^2}. \tag{5.9}$$

Similarly, we calculate the $F(\widehat{U})$ error using

$$\|\text{error}_{F(U)}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |(U_{i,j} - U_{i,j}^3) - F(\widehat{U}_{i,j})|^2}. \tag{5.10}$$

Figure 5.5 presents the Frobenius norm errors. Figure 5.5 (a) shows the errors for $\widehat{U}$ and Figure 5.5 (b) for $F(\widehat{U})$. The errors are given as the overall difference between the predicted and exact values of U and $F(U)$ according to Equations (5.9) and (5.10). The x-axis represents the time and the y-axis the Frobenius norm error. It can be observed that the error decreases as the model is trained, indicating that the model is learning and makes better predictions as time progresses. This trend is observed in all subdomains. As the model becomes more familiar with the problem, it can identify and capture more complex features of the solution. However, as the error approaches a plateau around 10^{-2}, the improvement in accuracy slows down. This

could be due to several factors such as the model reaching its capacity, the complexity of the problem, or the quality of the training data. Overall, the observed trend in the Frobenius norm error indicates that the model is learning and improving in accuracy.

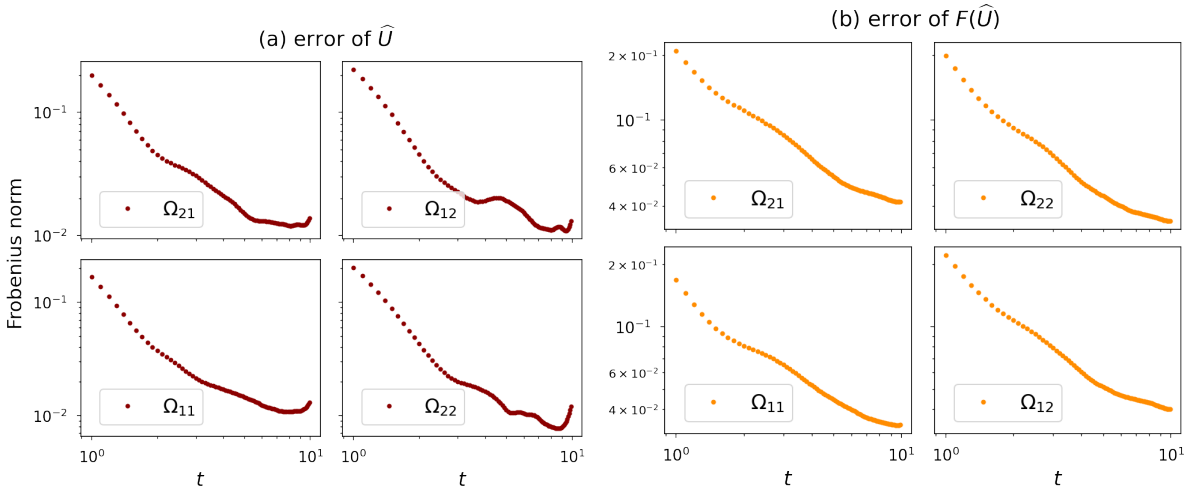


Figure 5.5: Frobenius norm error for (a) $\widehat{U}$ and (b) $F(\widehat{U})$ calculated by Equations (5.9) and (5.10).

In order to investigate the stability and consistency of the trained model, we compute the means and the standard deviations of the predicted solutions across ten runs with different initialization parameters for the neural network. This helps us to determine how much the predicted values vary from their expected values due to the stochastic initialization process. Smaller standard deviation indicates that the predicted values are more accurate because their values cluster more tightly around the mean. The results are presented in Figure 5.6, where the left set of figures shows the predicted $\widehat{U}$ and the right one the predicted function $F(\widehat{U})$. The dashed lines represent the mean values, while the highlights above and below the mean indicate the standard deviation along the x-axis.

5.4 Symbolic regression

Symbolic regression is an ML technique that involves identifying a mathematical expression or an equation that closely approximates a given dataset. This approach has been used in

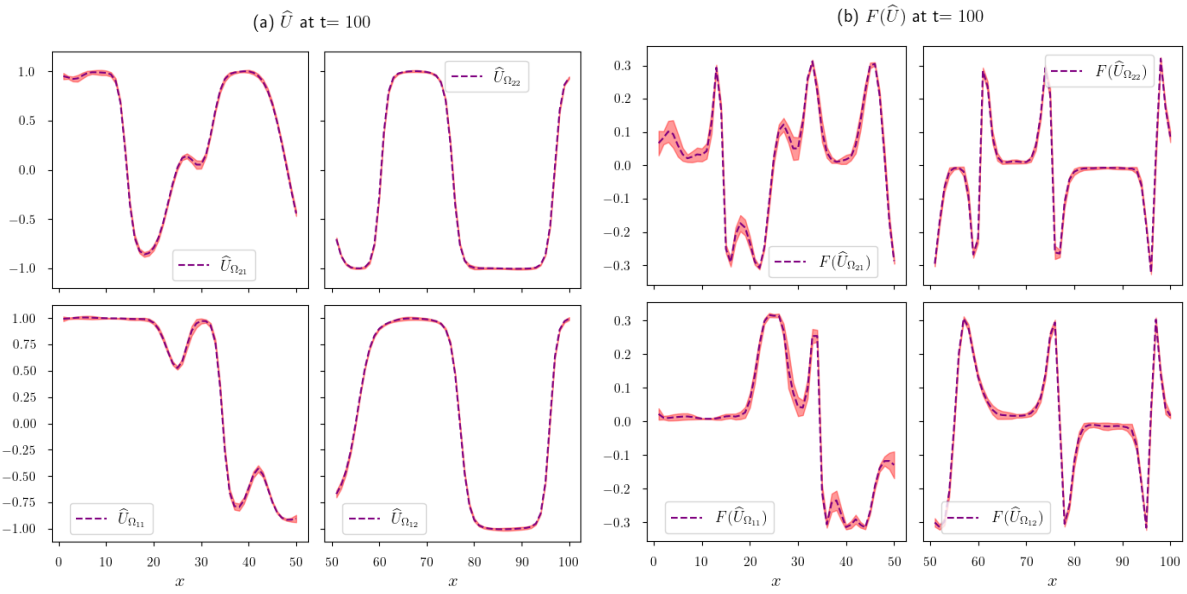


Figure 5.6: The mean value of the predictions (a) $\widehat{U}$ and (b) $F(\widehat{U})$ as well as the standard deviations for the ten different runs. The dashed lines are the averages of the ten different predictions with random selection of training sets. The highlights indicate the standard deviations. Predicted $\widehat{U}$ and $F(\widehat{U})$ are shown in the vertical axis and the horizontal axis is the x-coordinate.

various fields, including engineering, finance, biology, and modeling of complex systems by discovering correlations between input/output data pairs [261, 187, 301, 190, 302]. Instead of using predefined models, such as linear or polynomial regression, symbolic regression searches through a space of mathematical functions to find the function that best fits the data. In this study, we utilize the API for symbolic regression provided by the Python library gplearn [303] to discover mathematical expressions for $F(\widehat{U})$ as a function of simulation data U.

Typically, gplearn provides a set of predefined mathematical functions that can be utilized during the regression process. These functions encompass arithmetic operators (addition, subtraction, multiplication, and division), trigonometric functions (sine, cosine), logarithmic functions, exponential functions, and potentially others. The effectiveness of gplearn in uncovering specific functions hinges on both the functions incorporated in its search space and the complexity of the desired function. While gplearn can generally succeed in identifying functions like polynomials or basic trigonometric functions, the process of symbolic regression, in general, may face challenges if the desired functions are not included in the predefined set of

functions. In addition to gplearn, one can use Sparse Identification of Non-linear Dynamics (SINDY) [254, 255, 256] and PySR [304] package to discover the mathematical expression of an unknown term F. SINDY provides a valuable tool for inferring the dynamics of a system based on data, allowing for the discovery of meaningful mathematical expressions. PySR is a high-performance symbolic regression package that implements an algorithm for optimizing symbolic expressions using evolutionary algorithms.

Moreover, the success of function discovery in symbolic regression is influenced by various factors, including the quality and quantity of the available data. Having an ample amount of high-quality data is crucial for accurately uncovering complex functions. The greater the representativeness and diversity of the data, the higher the likelihood of capturing the underlying patterns and relationships. Inadequate or noisy data can impede the discovery process and result in inaccurate or incomplete outcomes [181, 182, 183].

On the other hand, as mentioned by Udrescu et al. [193], the task of discovering functions can be exceedingly complex and often deemed nearly impossible for symbolic regression to achieve. However, it is worth noting that functions encountered in physics and various scientific applications frequently exhibit certain simplifying properties that facilitate their discovery. These simplifying properties may include symmetries, separability, and compositionality. By capitalizing on these properties, several symbolic regression algorithms have been developed to uncover the mathematical expressions of functions. For instance, the AI Feynman algorithm [193] was applied to analyze a collection of 100 equations from the Feynman Lectures on Physics [305, 306, 307]. Remarkably, the algorithm successfully discovered all of the equations, demonstrating its effectiveness in revealing symbolic expressions that accurately capture the underlying physical relationships. This example highlights the potential of symbolic regression in unraveling complex equations and further underscores the significance of leveraging simplifying properties when attempting to discover mathematical representations of functions.

It is worth noticing that here the exact function for $F(U)$ is given by $U - U^3$, as described by Equations (5.1) and (5.2). We set the population size to $5,000$ and evolved 20 generations

until the error became close to 1 %. Since the equation $U - U^3$ consists of basic operations, it does not require the use of custom functions. The results of symbolic regression for multiple runs are presented in Table 5.2, where the predicted function $F(\widehat{U})$ for each subdomain is denoted as $F(\widehat{U}_{\Omega_{i,j}})$ with $i, j \in 1, 2$. The findings demonstrate that the model has accurately identified the underlying pattern between the input and output variables, and that the predicted functions contain the correct terms of U and U^3 with coefficients that are relatively close to the coefficients (both equal to one) of the equation $U - U^3$. A comparison between the exact function $F = U - U^3$ and the predicted functions further highlights the effectiveness of X-PINN in discovering the unknown components of equations, and is further validated by symbolic regression. The detailed pseudo-algorithm pairing X-PINN and symbolic regression is shown in Algorithm 1.

	$F(\widehat{U}_{\Omega_{11}})$	$F(\widehat{U}_{\Omega_{12}})$	$F(\widehat{U}_{\Omega_{21}})$	$F(\widehat{U}_{\Omega_{22}})$
1st run	$0.96U(1 - 0.89U^2)$	$0.96U(1 - 0.89U^2)$	$0.88(1 - U^3)$	$0.96U(1 - 0.89U^2)$
2nd run	$0.88U(0.99 - U^2)$	$0.96U(1 - 0.89U^2)$	$0.88U(0.994 - U^2)$	$0.96U(1 - 0.89U^2)$
3rd run	$0.96U(1 - 0.89U^2)$	$0.96U(1 - 0.89U^2)$	$0.88U(0.99 - U^2)$	$0.88U(1 - U^2)$
4th run	$0.88U(0.99 - U^2)$	$0.88U(0.99 - U^2)$	$0.96U(1 - 0.89U^2)$	$0.88U(0.99 - U^2)$
5th run	$0.88U(0.994 - U^2)$	$0.88U(0.994 - U^2)$	$0.88U(0.99 - U^2)$	$0.96U(1 - 0.89U^2)$

Table 5.2: Symbolic regression results for multiple runs to fit a mathematical formulation to the predicted function $F(\widehat{U}_{\Omega_{i,j}})$ on subdomain $_{\Omega_{i,j}}$. It is worth noting that the exact formulation for $F(\widehat{U})$ is $U - U^3$, and this formulation was closely approximated in multiple runs.

5.5 Noisy data analysis

To demonstrate the robustness of the proposed framework to discover equations from noisy data, we add noise to the original datasets to evaluate the performance of the framework. The purpose of adding noise to the data is to simulate real-world scenarios where data is often subject to randomness (e.g. thermal noise) and errors, rather than being precise and clean. Uniform noise of various magnitudes, sampled from a uniform distribution of zero mean and unit standard deviation, was applied to the original data set, which was used for training the

PINNs.

A summary of results using noisy data is presented in Figure 5.7. The predicted $F(\widehat{U})$ is shown in Figure 5.7(a) with 1% noise, while Figures 5.7 (b), (c), and (d) present the results for 2%, 3%, and 9% noise, respectively. The plots show the mean and standard deviation values that were calculated across ten runs for each set of predictions. Our results demonstrate that the sub-PINNs were able to accurately identify the unknown function F from noisy data. Notably, predictions on subdomains Ω_{12} and Ω_{21} were found to be less sensitive to noise than the other two subdomains. Additionally, we observe that the subdomain Ω_{11} has a higher standard deviation, indicating greater sensitivity to noise.

As above, we employ symbolic regression to obtain the mathematical formula for the predicted function $F(\widehat{U})$ using noisy data, and compare it with the exact expression $U - U^3$. The predicted $F(\widehat{U})$ and U were fed into the symbolic regression model and the outcomes are presented in Table 5.3. The results show that the algorithm has successfully identified the correct terms of U and U^3.

	$F(\widehat{U}_{\Omega_{11}})$	$F(\widehat{U}_{\Omega_{12}})$	$F(\widehat{U}_{\Omega_{21}})$	$F(\widehat{U}_{\Omega_{22}})$
(1% noise)	$0.96U(1-0.89U^2)$	$0.96U(1-0.89U^2)$	$0.87U(1-U^2)$	$0.96U(1-0.89U^2)$
(2% noise)	$0.96U(1-0.89U^2)$	$0.96U(1-0.89U^2)$	$U(0.96-0.98U^2)$	$0.96U(1-0.89U^2)$
(3% noise)	$0.96U(1-0.89U^2)$	$0.96U(1-0.89U^2)$	$0.956U(1.02-U^2)$	$0.956U(1-0.85U^2)$
(9% noise)	$0.96U(1-0.89U^2)U$	$0.956U(1-0.89U^2)$	$0.96U(1.02-U^2)$	$0.96U(1-0.89U^2)$

Table 5.3: Symbolic regression results for noisy data. $F(\widehat{U}_{\Omega_{i,j}})$ is the predicted function corresponding to subdomain $\Omega_{i,j}$. It is noteworthy that the exact formulation for $F(\widehat{U})$ is given by $U - U^3$. Despite the presence of 9% noise in the dataset, the results demonstrate that the algorithm has successfully identified a closely approximated formulation for the predicted $F(\widehat{U})$.

To evaluate the model's performance in recovering all the energy modes of system, we performed a modal analysis using proper orthogonal decomposition (POD) [308]. Figure 5.8 shows the ratios of the singular values of the matrices U (modal energy) and $\widehat{U}$ that have been reshaped to $(n_x \times n_y, n_t)$ to the sum of all singular values (total energy of the system). The singular values were obtained by performing singular value decomposition (SVD) on the matrices U and $\widehat{U}$. Specifically, the SVD of the matrix U is computed using the equation

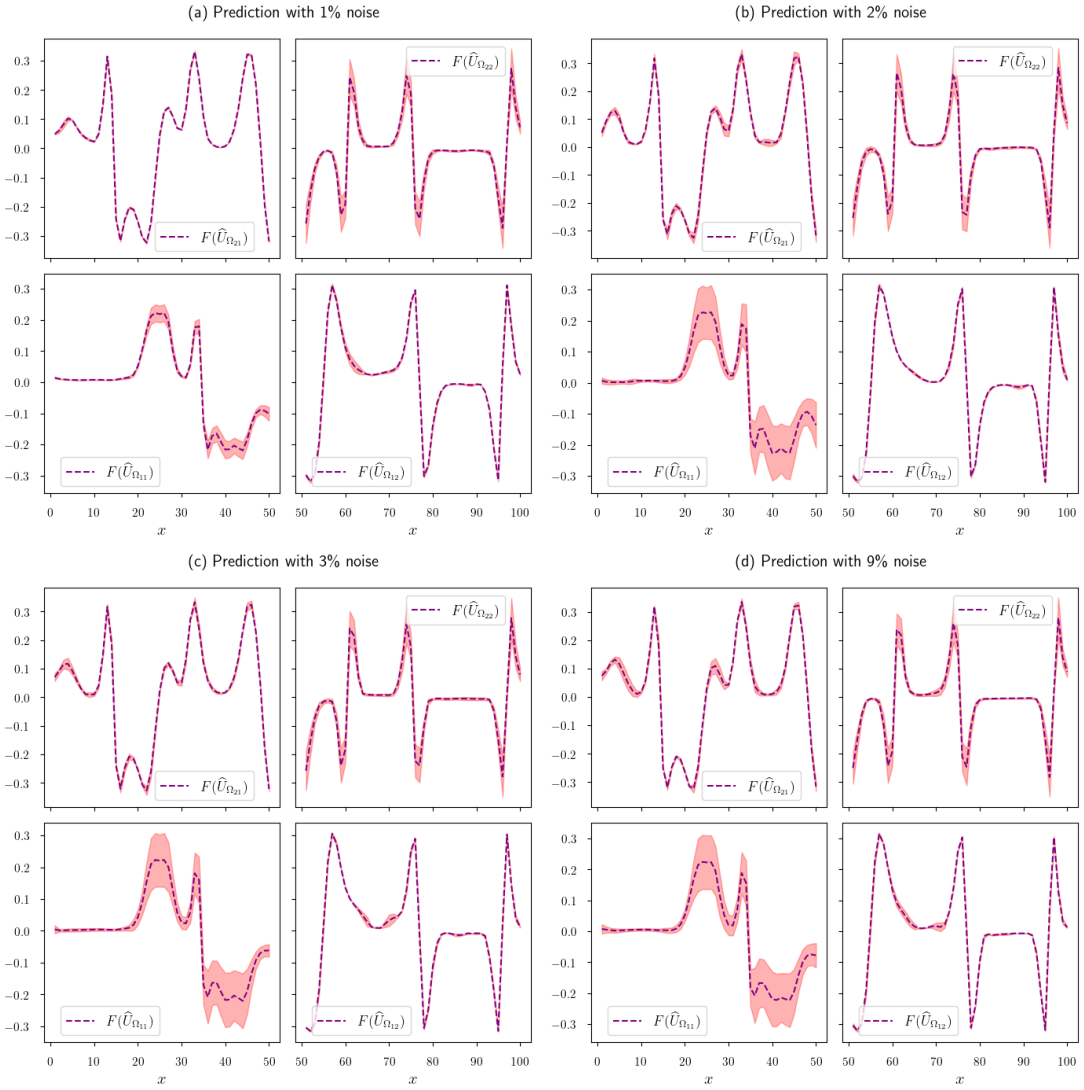


Figure 5.7: Mean and standard deviation of the predicted $F(\widehat{U}_{\Omega_{ij}})$ for each subdomain Ω_{ij} derived from noisy data with (a) 1% noise, (b), (c), and (d) 2%, 3%, 9% noise respectively. The vertical axis represents $F(\widehat{U}_{\Omega_{ij}})$ values, while the horizontal axis represents the corresponding values of x. The dashed lines represent the mean values of the predicted $F(\widehat{U}_{\Omega_{ij}})$, while the red highlights around the dashed lines indicate the standard deviation.

$U = W\Sigma V^T$, where W and V are orthogonal matrices, and Σ is a diagonal matrix containing the singular values of U on the diagonal [309, 310]. Noise that is uncorrelated and evenly distributed across the entire dataset can lead to instability in predictions or model performance. Therefore, the ratio of each singular value to the sum of all singular values can be used to assess the stability of predictions. As shown in Figure 5.8, the ratios of singular values decrease

over samples, indicating that the informative singular vectors are becoming progressively less dominant, or relevant, in the prediction task. Therefore, as we increase the number of samples, the importance of these specific singular vectors diminishes. Figure 5.8 shows that the model is able to recover the dominant energy modes very accurately even when the level of the noise is increased.

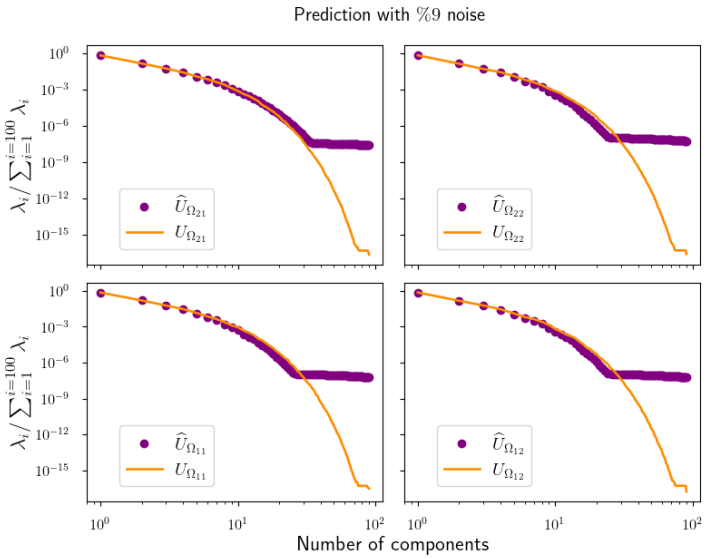


Figure 5.8: The ratio of each singular value to the sum of all singular values for the predicted $\widehat{U}_{\Omega_{ij}}$ for each subdomains Ω_{ij} with 9% noise. The vertical axis represents $\lambda_i / \sum_{i=1}^{i=100} \lambda_i$ values, while the horizontal axis represents the number of samples. The ratio of singular values decreases, which suggests that the informative singular vectors are becoming less dominant or relevant in predicting the target variable.

5.6 Optimal training datasets

In this section, we reduce the number of training points while still preserving its overall shape and distribution to understand how much information is needed to train a network. The technique is called downsampling [311, 312], which involves reducing the number of training points without sacrificing too much accuracy or losing too much information. The goal is to enhance the generalization performance of the model on unseen data and to reduce its complexity by minimizing the number of training points without compromising accuracy or losing

crucial information. For this purpose, we choose a downsampling rate that is appropriate for the dataset and ensure that the selected subset of data is representative of the entire dataset. To select an appropriate downsampling rate, one can consider factors such as the size of the original dataset, the complexity of the problem, and the computational resources available. It is also important to ensure that the selected subset of data is representative of the entire dataset, meaning that it captures the overall distribution and patterns of the original data. To evaluate the performance of the networks, we randomly select samples from the original dataset and consider four different sizes to ensure that the downsampling rate is not too high, resulting in a loss of critical information or a decrease in accuracy.

In the previous sections, our datasets were randomly split into training and test with a $80 : 20$ ratio. Each subdomain has thus been trained by using 80 samples, which implies a training set of size $(80, 50, 50)$, where 80 is the number of samples, and each subdomain has dimensions of $n_x = n_y = 50$.

To reduce the number of samples, we decreased the size of training data by reducing the number of training points in both time and space as illustrated in Figures 5.9 and 5.10. Figure 5.9 (a) indicates that for each subdomain, a training set of size $(60, 30, 30)$ was selected to represent 60% of the available data. As depicted in Figure 5.9 (b), (c), and (d), we trained the model using 50%, 30%, and 10% of the data, respectively. This corresponds to training sets of sizes $(50, 25, 25)$, $(30, 15, 15)$, and $(10, 5, 5)$ for each subdomain. The findings demonstrate that the networks can effectively capture the information in Figures 5.9 (a) and (b).

In contrast, in Figures 5.9 (c) and (d) the model failed to accurately identify the unknown function, suggesting a lack of sufficient training. Hence, at minimum, 50% of the data must be used to train the network to adequately capture the information.

We present the standard deviations and averages of the predicted $F(\widehat{U})$ for ten runs in Figure 5.10 to evaluate the sensitivity of the model's predictions to changes in the training data. Figure 5.10 (a) shows the predicted $F(\widehat{U})$ computed using 60% of the available training data, and Figures 5.10 (b), (c), and (d) show the predicted $F(\widehat{U})$ for models trained with 50%,

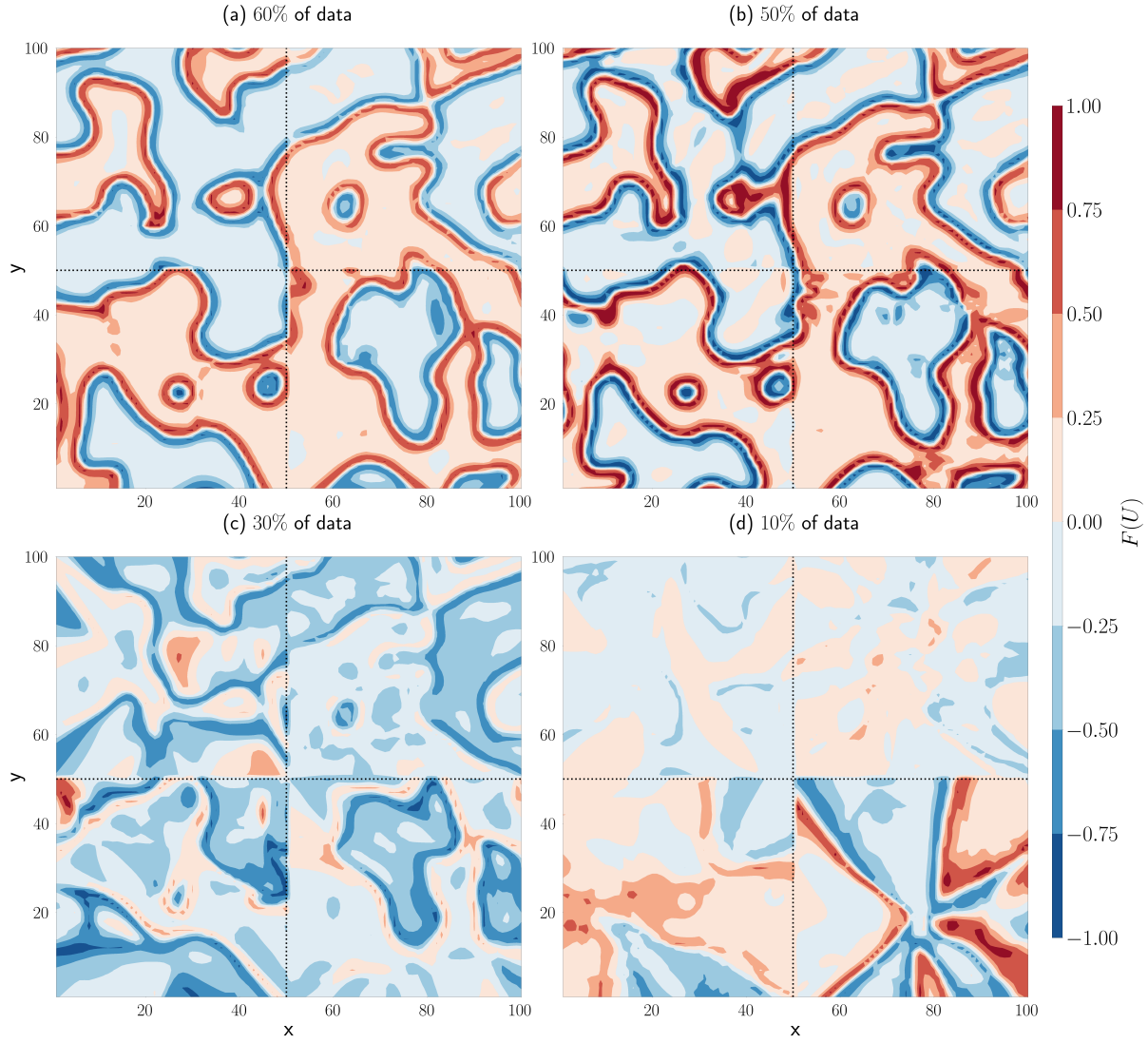


Figure 5.9: Snapshots of the predicted $F(\widehat{U})$ obtained from varying sample sizes of data. (a) illustrates the predicted $F(\widehat{U})$ when 60% of the data was used for training. (b), (c), and (d) correspond to cases where 50%, 30%, and 10% of the data were used for training, respectively. The results indicate that the neural networks can effectively capture the underlying information in (a) and (b) using only 60% and 50% of the original training data, respectively. However, in the case of (c) and (d), which have a much smaller subset of training points, the network is not able to perform well due to the lack of sufficient training data.

30%, and 10% of the training data, respectively.

The results in Figures 5.10 (a) and (b) indicate that the subdomains Ω_{11}, Ω_{12}, and Ω_{21} exhibit low sensitivity (i.e., low standard deviation), that is, they are less affected by changes in the training data and are more likely to generalize well to new data. However, in Ω_{21},

higher sensitivity was observed, which could indicate reduced robustness and generalization ability to new, unseen data. In contrast, in Figures 5.10 (c) and (d), all subdomains exhibited high sensitivity, with a high standard deviation around mean predictions. This suggests that the model is highly sensitive to changes in the training data, which could affect its ability to generalize to new and unseen data.

	$F(\widehat{U}_{\Omega_{11}})$	$F(\widehat{U}_{\Omega_{12}})$	$F(\widehat{U}_{\Omega_{21}})$	$F(\widehat{U}_{\Omega_{22}})$
(60% data)	$0.96U(1-0.89U^2)$	$0.96U(1-0.89U^2)$	$0.96U(1-0.89U^2)$	$0.96U(1-0.89U^2)$
(50% data)	$0.96U(1-0.89U^2)$	$0.56U(1-0.144U^2)$	$0.56U(1-0.144U^2)$	$0.96U(1-0.89U^2)$
(30% data)	$0.088U$	$0.088U$	$0.088U$	$0.088U$
(10% data)	$0.04U$	$0.02U$	$0.04U$	$0.02U$

Table 5.4: The mathematical formula for $F(\widehat{U}_{\Omega ij})$ for each subdomain Ω_{ij} using symbolic regression. The model was trained using different percentages of the available data, and the results show that the correct terms of U and U^3 were accurately identified when the model was trained with (a) 60% and (b) 50% of the data. However, the model's accuracy decreased with less data, and it could not accurately predict coefficients when trained with only 50% of the data. Moreover, the results demonstrate that training the model with only 30% and 10% of the data is insufficient to even predict the correct terms of U and U^3 in the function.

The mathematical expression of the function $F(\widehat{U})$ was discovered using a symbolic regression model by feeding in the predicted $F(\widehat{U})$ for different training data set sizes. The outcomes of the symbolic regression model are summarized in Table 5.4. The model was able to identify the correct terms of U and U^3 for the cases where 60% and 50% of the data were used for training. However, it was found that training the model with only 50% of the data is not sufficient for predicting the coefficients accurately. In contrast, the results for the cases where only 30% and 10% of the data were used for training indicate that the model could not even predict the correct terms of U and U^3 in the function. Therefore, based on these results, it is recommended to use at least 60% of the available datasets for training in order to obtain the correct form of the equation and good approximations for the coefficients in the symbolic regression model.

It is worth noting that the recommendation of using at least 60% of the available dataset for training was specific to the Allen-Cahn equation in the context mentioned. It is important to emphasize that the optimal fraction of data for training may vary for different datasets

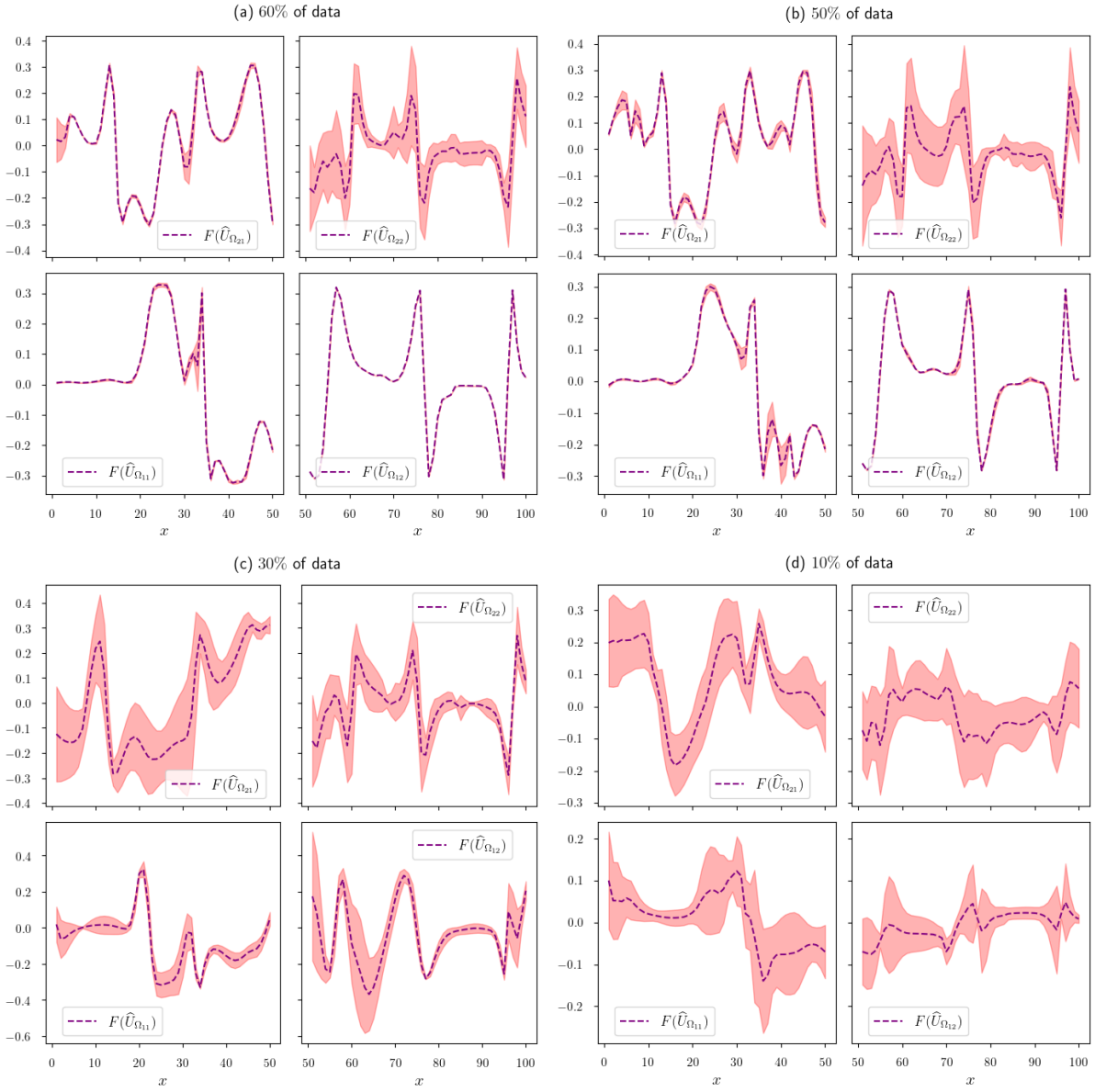


Figure 5.10: The standard deviation of predicted $F(\widehat{U}_{\Omega_{ij}})$ for each subdomains Ω_{ij} was computed using various sets of training points across multiple runs. The vertical axis represents $F(\widehat{U}_{\Omega_{ij}})$, while the horizontal axis represents x. In (a), the predicted $F(\widehat{U})$ was generated using 60% of the available training points, while (b), (c), and (d) show the predicted $F(\widehat{U})$ for models trained with only 50%, 30%, 10% of the training points, respectively. The plots depict the mean values of the predicted $F(\widehat{U}_{\Omega_{ij}})$ using dashed lines. The red highlights surrounding the dashed lines indicate the standard deviation of ten runs.

and equations. Depending on the complexity and characteristics of the problem at hand, other equations and datasets may require a different fraction or a more tailored approach to achieve accurate results. The 50% data refers to the number of snapshots taken to reach a steady state

solution for Allan-Cahn equation. The snapshots are stored at $(0, \Delta t, 2\Delta t, 3\Delta t....)$ where Δt is the optimal time step determined by the Courant-Friedrichs-Lewy (CFL) criterion [313]. The purpose of these snapshots is to capture the evolving behavior of the system over time. The spatial resolution of each snapshot is the optimal resolution, the minimum resolution necessary to prevent aliasing in the solution [314]. Aliasing refers to the distortion or loss of information that can occur when a signal is not properly sampled or represented. By choosing an appropriate spatial resolution, the snapshots can accurately represent the system's spatial features without introducing artifacts or inaccuracies due to aliasing. Therefore, it is crucial to consider the specific requirements and characteristics of each dataset and equation when determining the appropriate fraction for training.

5.7 Summary

In this study, we have applied modified X-PINNs to perform gray-box learning of the Allen-Cahn equation by decomposing the computational domain into four subdomains. We assumed that the equation comprises a Laplacian component, and our objective was to discover the nonlinear component of the equation, that is, $U-U^3$. The results indicate that X-PINNs have highly expressive capabilities of uncovering the unknown components across various subdomains. Given the X-PINNs' capability to handle complex data, they have the potential to become a valuable tool for gray-box learning of equations.

In addition, we implemented symbolic regression with the aim to uncover the general mathematical (closed) form of the unknown component (here, $U - U^3$) and its coefficients. The results show great agreement with the exact form of the equation. This outcome serves as a strong evidence of the efficacy of our proposed framework in accurately identifying unknown components of an equation from data/noisy data.

Additionally, we determined the number of data samples required for training a neural network to accurately identify the unknown component of the equation. Our results show that

at least 60% of the available data is necessary to accomplish that. Training with a smaller percentage of data resulted in less accurate and less close-to-exact predicted results. We also demonstrated the robustness of the trained model by computing the epistemic uncertainty using different initializations of the neural network parameters.

In this study, we have employed artificially generated data to assess the performance of an inverse method. Experimental data could come from a number of different types of time-dependent experiments. In particular, time-dependent microscopy (atomic for or optical, and scanning electron microscopy) and time-dependent scattering are the most natural sources for experimental data. Essentially, any imaging technique that is capable of producing time dependent data of interface/domain motion would be a suitable source for data.

Overall, the findings of this study show that the proposed framework is a powerful tool for discovering the unknown components of non-linear and complex PDEs by using the domain decomposition approach. Future research could explore the potential of the proposed framework to investigate how to further optimize the performance in different subdomains and using the method for solving PDEs involving higher order derivatives e.g., bi-Laplacians.

Competing interests

The authors declare no competing interests.

Acknowledgments

EK thanks Dr. Aniruddha Bora, Dr. Ehsan Kharazmi, and Zhen Zhang for their invaluable suggestions throughout the different phases of the project, and Mitacs Globalink Research Award Abroad and Western University's Science International Engagement Fund Award. MK thanks the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs Program. The work was partially supported by DOE grant (DE-SC0023389). Computing facilities were provided by the Digital Research Alliance of Canada

(https://alliancecan.ca). This research was partially conducted by using computational resources and services at the Center for Computation and Visualization, Brown University.

Chapter 6

Characterization of partial wetting by CMAS droplets using multiphase many-body dissipative particle dynamics and data-driven discovery based on PINNs

The contents of this chapter have been submitted with the following citation: E. Kiyani, M. Kooshkbaghi, K. Shukla, R. Babu Koneru, Z. Li, L. Bravo, A. Ghoshal, G. Em Karniadakis, and M. Karttunen, Characterization of partial wetting by CMAS droplets using multiphase many-body dissipative particle dynamics and data-driven discovery based on PINNs, submitted to Journal of Fluid Mechanics, JFM-23-1205, 2023. Preprint: arXiv:2307.09142

6.1 Introduction

Recent advancements in machine learning (ML) have opened the way for extracting governing equations directly from experimental (or other) data [115, 315, 316]. One particularly exciting use of ML is the extraction of partial differential equations (PDEs) that describe the evolution

and emergence of patterns or features [225, 253, 117, 317].

Spreading of liquids on solid surfaces is a classic problem [318, 204]. Although the theoretical foundations were laid by Young and Laplace already in the early 1800's [200, 201], there are still many open questions and it remains a highly active research field especially in the context of microfluidics [319] as well as in the design of propulsion materials [320]. As discussed in detail in the review of Popescu et al. [9], there are two fundamentally different cases: non-equilibrium spreading of the droplet, and the case of thermodynamic equilibrium when spreading has ceased and the system has reached its equilibrium state.

In thermodynamic equilibrium, the Laplace equation relates the respective surface tensions of the three interfaces via [318, 9]

$$\cos\theta_{\text{eq}} = \frac{\gamma_{SG} - \gamma_{SL}}{\gamma_{LG}}, \tag{6.1}$$

where θ_{eq} is the equilibrium contact angle, and γ_{SG}, γ_{SL}, and γ_{LG} are the surface tensions between solid-gas, solid-liquid and liquid-gas phases, respectively (see Figure 6.1). Two limiting situations can be identified, namely, partial wetting and complete wetting. In the latter, the whole surface becomes covered by the fluid and $\theta_{\text{eq}} = 0°$, that is, $\gamma_{SG} - \gamma_{LG} - \gamma_{SL} = 0$. When the equilibrium situation corresponds to partial wetting, $\theta_{\text{eq}} \neq 0°$, it is possible to identify the cases of high-wetting ($0° < \theta_{\text{eq}} < 90°$), low-wetting ($90° \leq \theta_{\text{eq}} < 180°$), and non-wetting ($\theta_{\text{eq}} = 180°$).

When a droplet spreads, it is out of equilibrium and properties such as viscosity and the associated processes need to be addressed [318, 204, 9]. In experiments, the most common choice is to use high viscosity liquids in order to eliminate inertial effects. An early classic experiment by Dussan and Davis [321] gave a beautiful demonstration of some of the phenomena. They added tiny drops of marker dye on the surface of a spreading liquid. They observed a caterpillar-type rolling motion of the marker on the surface giving rise to dissipation via viscous friction. Effects of viscosity and dissipation remain to be fully understood and they have

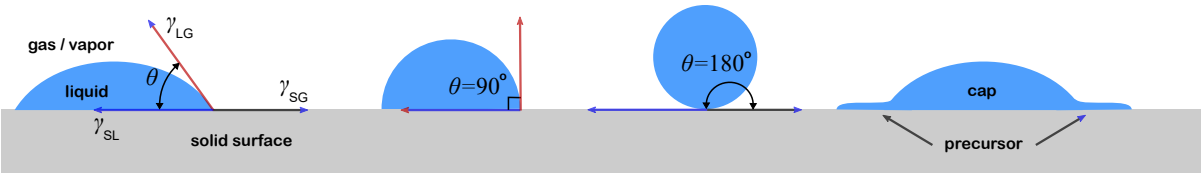


Figure 6.1: A schematic showing the equilibrium contact angle (that is, $\theta \equiv \theta_{eq}$), the surface tensions (γ), the threshold between low- and high-wetting regimes ($\theta_{eq} = 90°$), and a situation of a non-wetting droplet ($\theta_{eq} = 180°$). The last panel demonstrates the occurrence of a precursor that is observed in some cases. In that case, the (macroscopic) contact angle is defined using the macroscopic part of the droplet as indicated by the black line in the rightmost figure. The height of the precursor is in the molecular length scales [7, 8, 9].

a major role in wetting phenomena [206, 322, 323].

Calcium-magnesium-aluminosilicate, CMAS, is a molten mixture of several oxides, including calcia (CaO), magnesia (MgO), alumina (Al_2O_3), and silicate (SiO_2). It has a high melting point, typically around 1,240°C [324] (although it can be significantly higher, see, e.g., Wiesner et al. and references therein [325]), which allows it to exist in the molten state even at high temperatures encountered in modern aviation gas turbine engines [326, 327]. With high viscosity, high density, and high surface tension, CMAS tends to form non-volatile droplets of $\theta_{eq} \neq 0$ rather than completely wetting the surfaces [328, 329, 330]. When it solidifies, it forms a glass-like material that can adhere to surfaces and resist erosion. The buildup of CMAS on turbine engine components can lead to clogging of the cooling passages and degradation of the protective coatings, resulting in engine performance issues and even damage or failure [327, 331, 325, 326].

The spreading of a droplet over a solid surface is commonly characterized using a power law, $r \sim t^{\alpha}$, which expresses the radius of the wetted area as a function of time. The relationship is called Tanner's law for macroscopic completely wetting liquids at late times with $\alpha = 1/10$ [203, 204]. Power laws have also been demonstrated at microscopic scales [8]. However, several conditions such as surface properties, droplet shape, and partial wetting result in deviations from Tanner's law [205, 206, 207].

A common method, and as the above suggests, for analyzing the spreading dynamics is to investigate existence of the power law behavior. To determine the presence of such power-law

regimes in data, one can simply employ

$$\alpha(t) = \frac{d\ln(r)}{d\ln(t)}. \tag{6.2}$$

While this has worked remarkably well for complete wetting by viscous fluids, the situation for partial wetting is different [207]. In our study, we investigate the spreading behavior of CMAS droplets using multiphase many-body dissipative particle dynamics (mDPD) simulations. We generalize Equation (6.2) such that it includes dependence on the initial droplet radius R_0 and θ_{eq} in order to describe partial wetting, that is, $\alpha \equiv \alpha(t, R_0, \theta_{\text{eq}})$.

Our objective is to gain a comprehensive understanding of the behavior of CMAS droplet spreading dynamics by integrating knowledge about the fundamental physics of the system into the neural network architecture. To achieve this, we employ the framework of Physics-Informed Neural Networks (PINNs) [114], an emerging ML technique that incorporates the physics of a system into deep learning. PINNs address the challenge of accurate predictions in complex systems with varying initial and boundary conditions. By directly incorporating physics-based constraints into the loss function, PINNs enable the network to learn and satisfy the governing equations of the system.

The ability of PINNs to discover equations makes them promising for applications in scientific discovery, engineering design, and data-driven modeling of complex physical systems [125]. Their integration of physics-based constraints into the learning process enhances their capacity to generalize and capture the underlying physics accurately. Here, we also employ symbolic regression to generate a mathematical expression for each unknown parameter. Furthermore, we employ Bayesian Physics-Informed Neural Networks (B-PINNs) [178] to quantify the uncertainty of the predictions.

The rest of this article is structured as follows: In Section 6.2, we provide an overview of mDPD simulation parameters and system setup. Simulation outcomes and the data preparation process are presented in Section 6.3. Section 6.4 gives a brief introduction to the PINNs

architecture, followed by a presentation of the results of PINNs and parameter discovery. The symbolic regression results and the mathematical formulas for the parameters are presented in Section 6.5. Section 6.6 covers the discussion on B-PINNs as well as the quantification of uncertainty in predicting the parameters. Finally, we conclude with a summary of our work in Section 6.7.

6.2 Multiphase many-body dissipative particle dynamics simulations

Three-dimensional simulations were performed using the mDPD method [208, 209, 210], which is an extension of the traditional dissipative particle dynamics (DPD) model [211, 212]. DPD is a mesoscale simulation technique for studies of complex fluids, particularly multiphase systems, such as emulsions, suspensions, and polymer blends [332, 333, 334]. The relation between DPD and other coarse-grained methods and atomistic simulations have been studied and discussed by Murtola et al. [4], Li et al. [335, 336], and Español and Warren [337].

In DPD and mDPD models, the position ($\vec{r}_i$) and velocity ($\vec{v}_i$) of a particle i with a mass m_i are governed by Newton's equations of motion in the form of

$$\begin{aligned} \frac{d\vec{r}_i}{dt} &= \vec{v}_i, \\ m_i \frac{d\vec{v}_i}{dt} &= \vec{F}_i = \sum_{j \neq i} \vec{F}^{\mathrm{C}}_{ij} + \vec{F}^{\mathrm{D}}_{ij} + \vec{F}^{\mathrm{R}}_{ij}. \end{aligned} \tag{6.3}$$

The total force on particle i, that is, $\vec{F}_i$, consists of three pairwise components, i.e., the conservative $\vec{F}^{\mathrm{C}}$, dissipative $\vec{F}^{\mathrm{D}}$, and random forces $\vec{F}^{\mathrm{R}}$. The latter two are identical in DPD and mDPD models, given by,

$$\vec{F}^{\mathrm{D}}_{ij} = -\gamma \omega_{\mathrm{D}}(r_{ij})(\vec{v}_{ij} \cdot \vec{e}_{ij})\vec{e}_{ij}, \tag{6.4}$$

$$\vec{F}^{\mathrm{R}}_{ij} = \zeta \omega_{\mathrm{R}}(r_{ij})(dt)^{-1/2}\xi_{ij}\vec{e}_{ij}, \tag{6.5}$$

where $\vec{e}_{ij}$ is a unit vector, ω_D and ω_R are weight functions for the dissipative and random forces, and ξ_{ij} a pairwise conserved Gaussian random variable with zero mean and second moment $\langle \xi_{ij}(t)\xi_{kl}(t')\rangle = (\delta_{ik}\delta_{jl}+\delta_{il}\delta_{jk})\delta(t-t')$, where δ_{ij} is the Kronecker delta and $\delta(t-t')$ the Dirac delta function. Together, the dissipative and random forces constitute a momentum conserving Langevin-type thermostat. The weight functions and the constants γ and ζ are related via fluctuation-dissipation relations first derived by Español and Warren [211]

$$\omega_D = (\omega_R)^2, \tag{6.6}$$

$$\zeta = \sqrt{2\gamma k_B T}, \tag{6.7}$$

in which k_B is the Boltzmann constant and T the temperature. This relation guarantees the canonical distribution [211] for fluid systems in thermal equilibrium. The functional form of the weight function is not specified, but the most common choice (also used here) is

$$\omega_D(r_{ij}) = \begin{cases} (1 - r_{ij}/r_d)^s & \text{for } r_{ij} \le r_d \\ 0 & \text{for } r_{ij} > r_d. \end{cases} \tag{6.8}$$

Here, $s = 1.0$ is used and r_d defines a cutoff distance for the dissipative and random forces.

Although the above equations are the same for both DPD and mDPD, they differ in their conservative forces. Here, we use the form introduced by Warren [338, 339],

$$F_{ij}^C = A\omega_C(r_{ij})\vec{e}_{ij} + B(\rho_i + \rho_j)\omega_B\vec{e}_{ij}. \tag{6.9}$$

The functional form of both weight functions ω_C and ω_B is the same as ω_D in Equation (6.8) but with different cutoff distances r_c and r_b. The first term in Equation (6.9) is the standard expression for the conservative force in DPD, and the second one is the multi-body term. The constants A and B are chosen such that $A < 0$ for attractive interactions and $B > 0$ for repulsive interactions; note that in conventional DPD $A > 0$ and $B = 0$. The key component is the

weighted local density

$$\rho_i = \sum_{j \neq i} \omega_\rho(r_{ij}). \tag{6.10}$$

There are several ways to choose the weight function [332] and here, the normalized Lucy kernel [213] in 3-dimension is used,

$$\omega_\rho(r_{ij}) = \frac{105}{16\pi r_{c\rho}^3}\left(1 + \frac{3r_{ij}}{r_{c\rho}}\right)\left(1 - \frac{r_{ij}}{r_{c\rho}}\right)^3, \tag{6.11}$$

with a cutoff distance $r_{c\rho}$ beyond which the weight function ω_ρ becomes zero.

6.2.1 Simulation parameters and system setup

To simulate molten CMAS, the parameter mapping of Koneru et al. [10] was used together with the open-source code LAMMPS [214]. In brief, the properties of molten CMAS at about 1,260° C based on the experimental data from Naraparaju et al. [340], Bansal and Choi [341], and Wiesner et al. [325], were used. In physical units, density was 2,690 kg/m^3, surface tension 0.46 N/m, and viscosity 3.6 Pa·s. Using the density and surface tension to estimate the capillary length ($\kappa = (\sigma/(\rho g))^{1/2}$) gives 4.18 mm. The droplets in the simulations (details below) had linear sizes shorter than the capillary length and hence gravity was omitted. In terms of physical units, time: 6.297×10^{-6} s, length: 17.017×10^{-6} m, mass: 1.964×10^{-8} kg.

Using the above values, droplets of initial radii of $d = 8, 9, 10, 11$, and 12 in mDPD units, corresponding to $R_0 = 0.136$ mm, 0.153 mm, 0.17 mm, 0.187 mm, and 0.204 mm, respectively, were used in the simulations; all of them are smaller than the capillary length. The time step was 0.002 (mDPD units) corresponding to 12.59 ns. In addition, $k_\mathrm{B}T = 1$, $r_c = 1.0$, $r_b = r_{c\rho} = 0.75$, $r_d = 1.45$, $\gamma = 20$, and $B = 25$. The attraction parameter, A in Equation (6.9) has to be set for the interactions between the liquid particles (A_ll), and the liquid and solid particles (A_ls). The former was set to $A_\mathrm{ll} = -40$ and A_ls was chosen based on simulations that provided the desired θ_eq, thus allowing for controlled variation of θ_eq (see Figure 6.2).

The initial configuration of the droplet and the solid wall were generated from a random

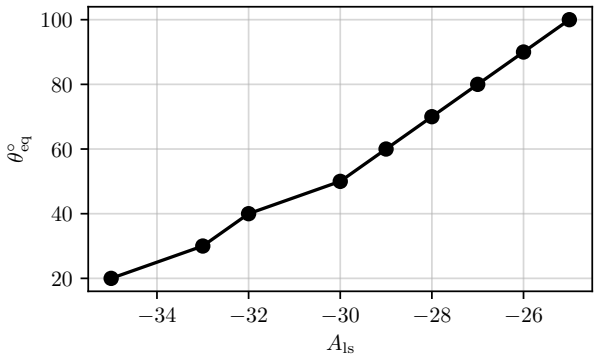


Figure 6.2: The equilibrium contact angles θ_{eq} for the different attraction parameters between the liquid and solid particles (A_{ls}; see Equation (6.9)). It is worth noting that the data for this figure has been extracted from Koneru et al. [10].

distribution of equilibrated particles with a number density $\rho = 6.74$. This amounts to about 60,660 particles in the wall and depending on the initial radius, anywhere between 14,456 and 48,786 particles in the droplet. Periodic boundary conditions are imposed along the lateral directions and a fixed, non-periodic boundary condition is imposed along the wall-normal direction. Since mDPD is a particle-based method, the spreading radius and the dynamic contact angle are approximated using surface-fitting techniques. First, the outermost surface of the droplet is identified based on the local number density, i.e., particles with $\rho \in [0.45, 0.6]$. The liquid particles closest to the wall are fitted to a circle of radius r, i.e., the spreading radius. On the other hand, a sphere with the centroid of the droplet as the center is fit to the surface particles to compute the contact angle. The contact angle is defined as the angle between the tangent at the triple point (liquid-solid-gas interface) and the horizontal wall. The wall in these simulations is made-up of randomly distributed particles to eliminate density and temperature fluctuations at the surface. Following [342], the root mean squared height (R_q) of the surface scales linearly with $1/\sqrt{N_w}$ where $N_w = 2\pi r_{cw}^3/3 \cdot \rho_w$ is the number of neighboring particles. In this work, R_q comes out to be around 0.0708 mDPD units or 1.2 μm.

As the CMAS droplet spreads on the substrate, it loses its initial spherical shape and begins to wet the surface as depicted in Figure 6.3, forming a liquid film between the droplet and the substrate. Understanding how droplets behave on surfaces is important for a wide range of applications, including in industrial processes, microfluidics, propulsion materials, and the

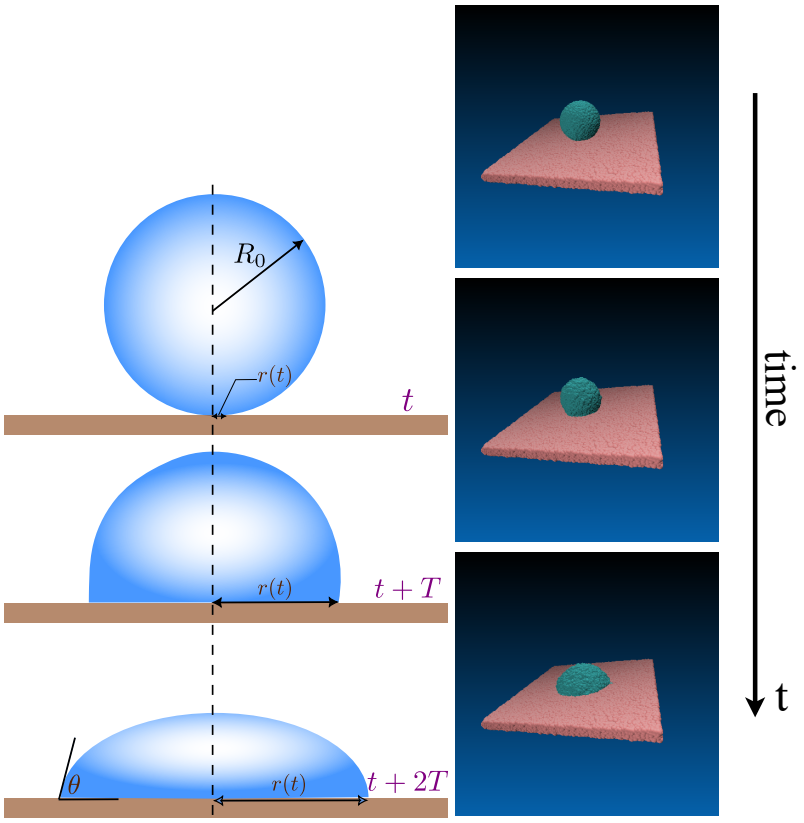


Figure 6.3: Left: Illustration of the spreading behavior of a CMAS droplet on a high surface energy surface at different times. The droplet with initial size of R_0 spreads on the surface with radius $r(t)$ and contact angle $\theta(t)$. Right: A series of snapshots from a simulation of a droplet with initial size of $R_0 = 0.136$ mm and an equilibrium contact angle of $\theta_{eq} = 93.4°$.

design of self-cleaning surfaces [343, 344, 345, 320, 328].

6.3 Simulation results

The size of a droplet changes over time. By tracking the changes, we can gain insight into the physical processes involved in spreading. The time evolution of the droplet radius ($r(t)$) is shown in Figure 6.4. The log-log plots show the effect of the initial drop size R_0 and equilibrium contact angles θ_{eq} on the radius $r(t)$. Figure 6.4 (a) displays $r(t)$ for initial drop sizes R_0 of 0.136 mm, 0.153 mm, 0.17 mm, 0.183 mm, and 0.204 mm and equilibrium contact angle of $\theta_{eq} = 54.6°$. Similarly, Figure 6.4 (b) shows the spreading radius for different equilibrium contact angles ($\theta_{eq} = 93.4°$, 85.6°, 77.9°, 70.1°, 62.4°, 54.6°, 45.3°, and 39.1°) with an initial

drop size of $R_0 = 0.136\,\text{mm}$.

Eddi et al. [346] used high-speed imaging with time resolution covering six decades to study the spreading of water-glycerine mixtures on glass surfaces. By varying the amount of glycerine, they were able to vary the viscosity over the range 0.0115-1.120 Pa·s. They observed two regimes, the first one for early times with α changing continuously as a function of time from $\alpha \approx 0.8$ to $\alpha \approx 0.5$. This was followed by a sudden change to the second regime in which α settled to $0.1 < \alpha < 0.2$. As pointed out by Eddi et al. [346], the second regime agrees with Tanner's law [203]. All of their systems displayed complete wetting.

Based on $r(t)$ of the CMAS drops and α shown in Figure 6.4, it can be observed that $r(t)$ (and based on the power-law α) depends both on the initial drop size and the equilibrium contact angle. The values of α for some simulation datasets are plotted over time in Figure 6.5. The plot shows the behaviour of α for different initial drop sizes and equilibrium contact angles of $\theta_{eq} = 62.4°$ and $\theta_{eq} = 85.6°$.

Inspired by the experimental results of Eddi et al. [346], the simulations of Koneru et al. [10], and the current simulations, we propose a simple sigmoid type dependence for α,

$$\frac{d\ln(r)}{d\ln(t)} = \alpha(t, R_0, \theta_{eq}) := \eta\left[\frac{1}{1+\exp\left(\beta(\tau - \ln(t))\right)} - 1\right]. \tag{6.12}$$

The two constant values of α discussed in the above references are the two extrema of the sigmoid curve, given that the transition between the two regimes occurs at $\ln(t_{transition}) = \tau$. The parameters of Equation (6.12) are discovered by PINNs and their dependence on R_0 and θ_{eq} is then expressed using symbolic regression. The general steps in the discovery of the droplet spreading equation and the extraction of the unknown parameters $\eta(R_0, \theta_{eq})$, $\beta(R_0, \theta_{eq})$, and $\tau(R_0, \theta_{eq})$ are shown in Figure 6.6 and can be summarized as follows:

- Data collection: For this study, data is collected by conducting three-dimensional simulations using the mDPD method in LAMMPS with varying initial drop sizes R_0 and equilibrium contact angles θ_{eq}.

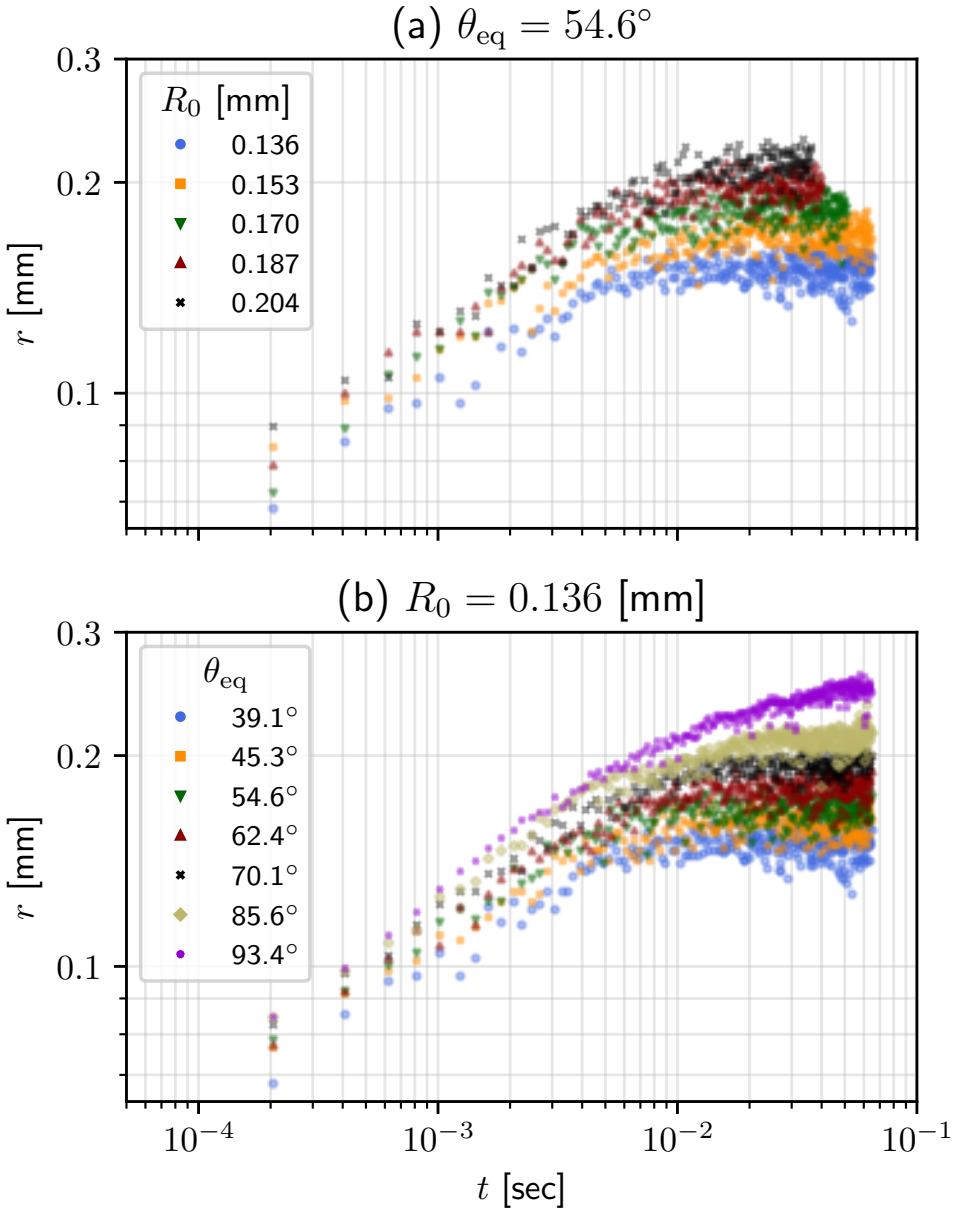


Figure 6.4: The impact of θ_{eq} and R_0 on the droplet radii as a function of time for various (a) initial drop sizes with equilibrium contact angle of $\theta_{\text{eq}} = 54.6°$ corresponding to $A_{\text{ls}} = 30.0$, and (b) equilibrium contact angles (corresponding to $A_{\text{ls}} = -25.0, -25.8, -27.0, -28.0, -29.0, -30.0, -31.4, -32.2$) and initial drop size $R_0 = 0.136$ mm.

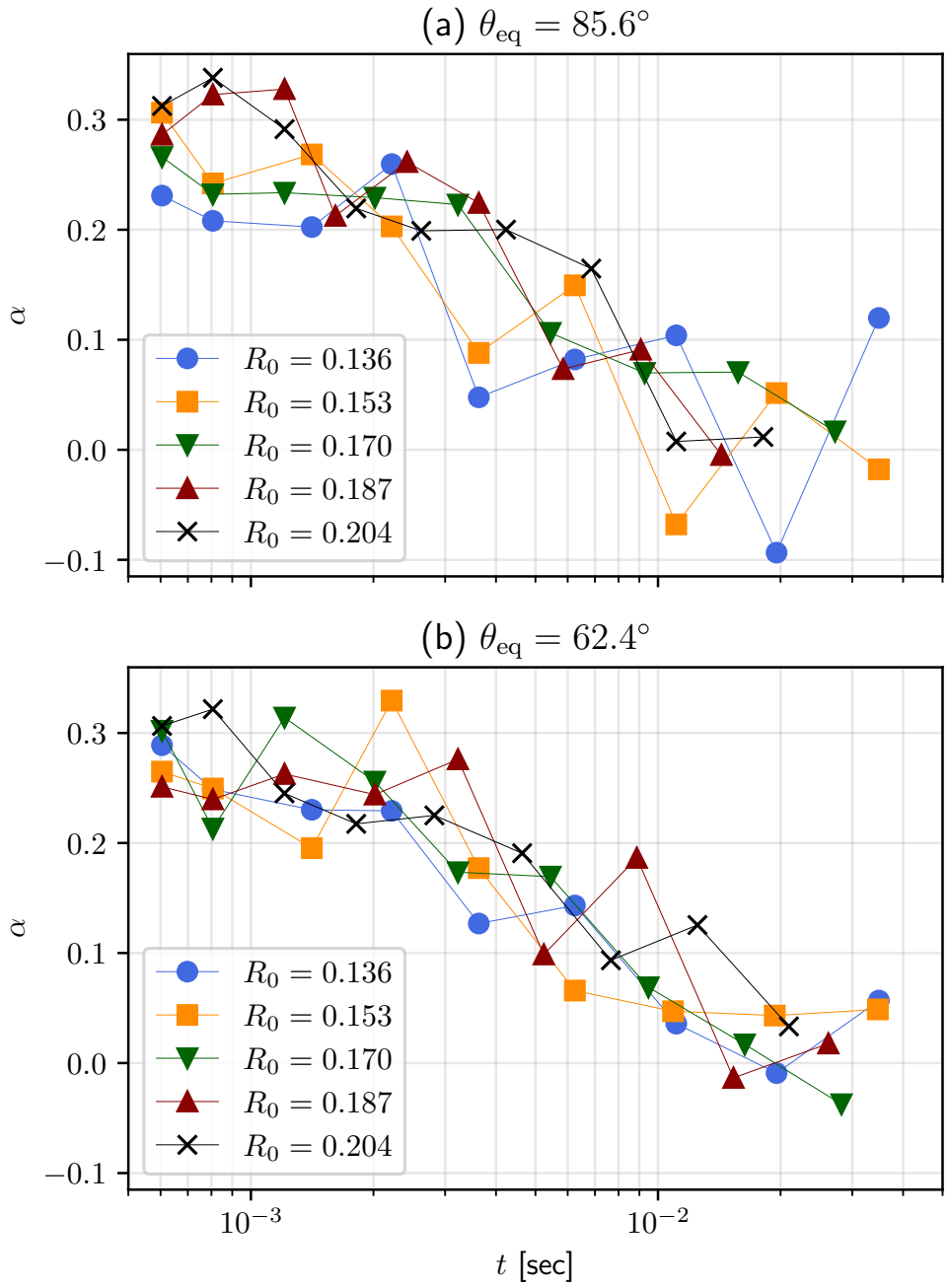


Figure 6.5: The value of α, calculated using Equation (6.2), varies for different initial radii and fixed equilibrium contact angles $\theta_{eq} = 62.4°$ and $\theta_{eq} = 85.6°$. The figure illustrates that α is influenced by both the initial drop size R_0 and the equlibrium contact angle θ_{eq}.

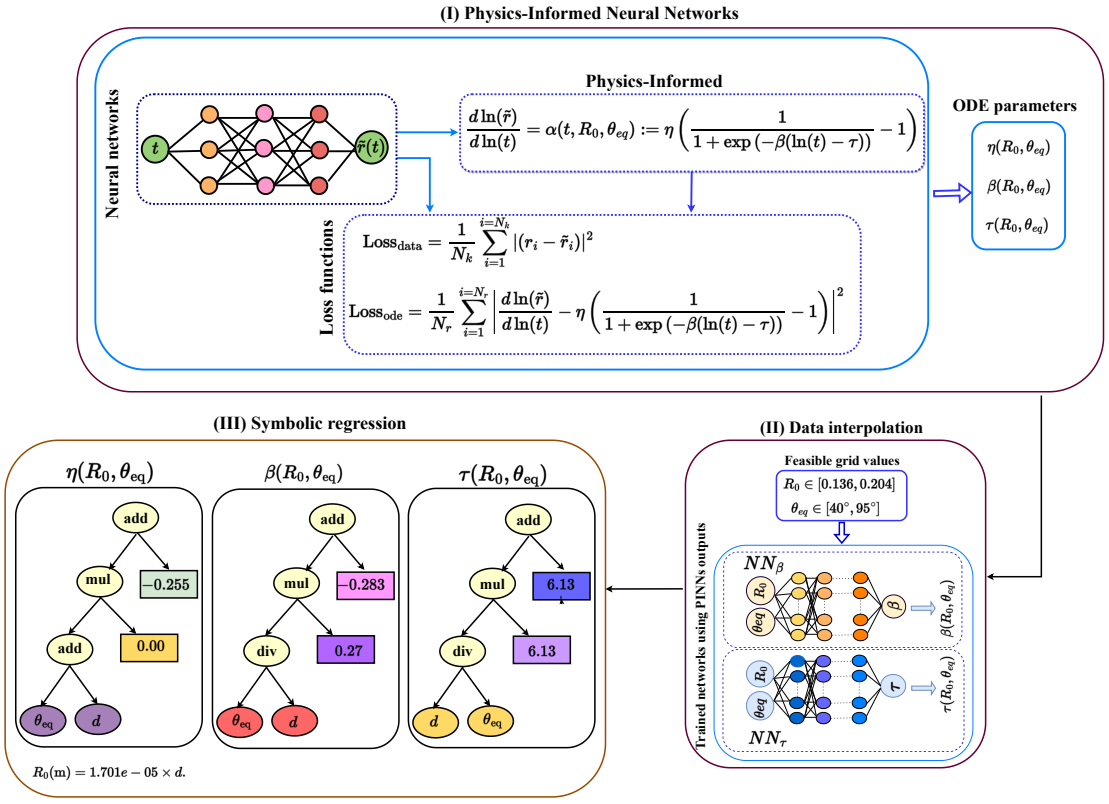


Figure 6.6: The process of utilizing PINNs to extract three unknown parameters of the ODE (6.12), using three-dimensional mDPD simulation data. First, a neural network is trained using simulation data, where the input is time t and the output is spreading radii $\tilde{r}(t)$. This neural network comprises four layers with three neurons and is trained for 12,000 epochs. Subsequently, the predicted $\tilde{r}(t)$ is used to satisfy Equation (6.12) in the physics-informed part. The loss function for this process consists of two parts: data matching and residual. By optimizing the loss function, the values of $\eta(R_0, \theta_{\text{eq}})$, $\beta(R_0, \theta_{\text{eq}})$, and $\tau(R_0, \theta_{\text{eq}})$ are determined for each set of R_0 and θ_{eq}. After predicting the unknown parameters using PINNs, two additional neural networks, denoted as NN_β and NN_τ, are trained using these parameters to generate values for the unknown parameters at points where data is not available. The outputs of these networks, together with the outputs of the PINNs, are then fed through a symbolic regression model to discover a mathematical expression for discovered parameter.

- PINNs: The input of the network is time t and output of the network is the spreading radii $\tilde{r}(t)$. The physics informed part of PINNs encapsulated in designing the loss function. In this study, the "goodness" of the fit is measured by (a) deviation from trained data together with (b) deviation of network predictions and those from ODE (6.12) solutions. This optimization process reveals the the unknown parameters $\eta(R_0, \theta_{\text{eq}})$, $\beta(R_0, \theta_{\text{eq}})$, and $\tau(R_0, \theta_{\text{eq}})$.

- Data interpolation: After PINNs are trained, we used their predictions together with two additional multilayer perceptron neural networks to fill the sparse parameter space. This step helps our next goal which is relating the ODE parameters to R_0 and θ_{eq} without performing three-dimensional simulations.

- Symbolic regression: Discovering a mathematical expression for each unknown parameter, $\eta(R_0, \theta_{\text{eq}})$, $\beta(R_0, \theta_{\text{eq}})$, and $\tau(R_0, \theta_{\text{eq}})$ of the Equation (6.12).

- In order to quantify the uncertainty associated with our predictions, we utilize B-PINN and leverage the insights gained from PINNs' prediction and symbolic regression, with a specific emphasis on the known value of η. By employing B-PINN, we can effectively ascertain the values of two specific parameters, β and τ, which in turn enable us to quantify the uncertainty in our predictions.

6.4 Physics-informed neural networks (PINNs)

PINNs are a promising approach that leverages the flexibility and scalability of deep neural networks to solve or even to discover governing equations, incorporating physical laws and constraints into the network structure [114, 125, 347, 348, 349, 350].

The training phase of PINNs consists of an optimization process applied on top of a neural network structure to identify the set of parameters in the governing equations with a pre-defined form (PDEs or ODEs). Those parameters aim to satisfy both data and the physical constraints.

6.4.1 Discovering parameters of ODE

As discussed in Section 6.3, our study aims to identify the values of the parameters $\eta(R_0, \theta_{eq})$, $\beta(R_0, \theta_{eq})$, and $\tau(R_0, \theta_{eq})$ in the ODE given by Equation (6.12). The general steps of the framework are shown in Figure 6.6 (I). PINNs take time t as an input to predict $\tilde{r}(t)$ at each time. The network architecture consists of four layers with three neurons each and is trained for $12,000$ epochs with the learning rate of 10^{-3}.

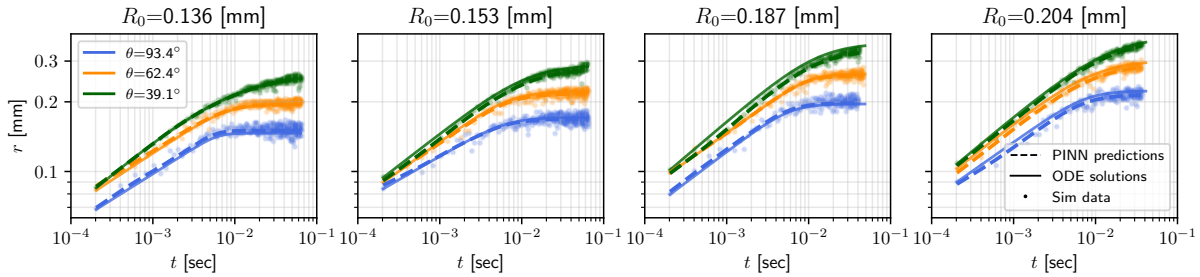


Figure 6.7: Comparison of the time evolution of the droplet radii: mDPD simulations (symbols), ODE model (6.12) (solid lines) and PINN predictions (dashed lines) for $\theta_{eq} = \{39.1°, 62.4°, 93.4°\}$ and $R_0 = \{0.136, 0.153, 0.187, 0.204\}$ mm parameter sets.

The predicted values for the time evolution of the radii $\tilde{r}(t)$ should satisfy the data and the physics-informed step, i.e., meet the requirements of the ODE, Equation (6.12). The two-component loss function, designed to meet the requirements, consists of $\text{Loss}_{\text{data}}$ and Loss_{ODE},

$$\text{Loss}_{\text{data}} = \frac{1}{N_k} \sum_{i=1}^{i=N_k} |(r_i(t) - \tilde{r}_i(t))|^2, \tag{6.13a}$$

$$\text{Loss}_{\text{ODE}} = \frac{1}{N_r} \sum_{i=1}^{i=N_r} \left| \frac{d \ln(\tilde{r})}{d \ln(t)} - \eta \left(\frac{1}{1 + \exp\left(-\beta(\ln(t) - \tau)\right)} - 1 \right) \right|^2, \tag{6.13b}$$

where $\tilde{r}(t)$ and $r(t)$ stand for the radii from the prediction and simulation, respectively. N_k is the number of training points and N_r is the number of residual points. Figures 6.7, 6.8, and 6.9 illustrate the results obtained by utilizing PINNs to discover the parameters of the ODE

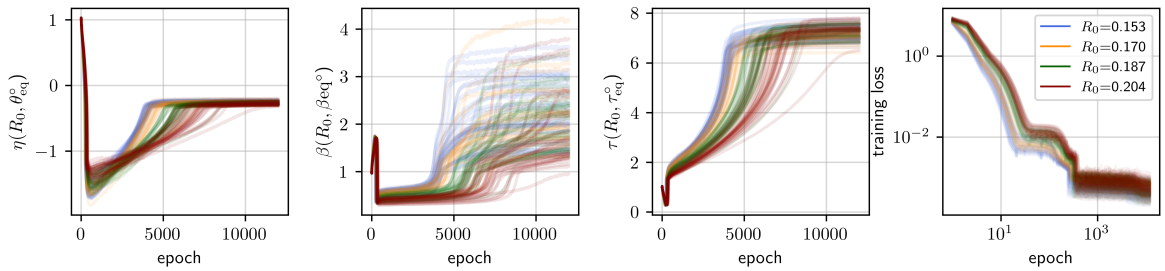


Figure 6.8: The first three plots show the evolution of parameters $\eta(R_0, \theta_{eq})$, $\beta(R_0, \theta_{eq})$, and $\tau(R_0, \theta_{eq})$ over multiple epochs. These plots demonstrate that the parameters gradually converge to a stable state after 12,000 epochs. The rightmost figure displays the traces of the loss function for the PINNs framework. The learning curves demonstrate the decreasing trend of the loss functions, indicating that they converge to a stable point for all initial drop sizes and θ_{eq}.

(Equation (6.12)), which describes the dynamics of the radii of the CMAS drops.

Figure 6.7 shows comparisons of the simulation data ($r(t)$), the prediction ($\tilde{r}(t)$) and solution of the ODE, Equation (6.12). The figure demonstrates remarkable degree of agreement between simulations, PINNs and our ODE model.

The first three panels in Figure 6.8 show the convergence of $\eta(R_0, \theta_{eq})$, $\beta(R_0, \theta_{eq})$, and $\tau(R_0, \theta_{eq})$ parameters during training. One can conclude that, all three parameters stabilize roughly after 10,000 epochs. In the rightmost panel of Figure 6.8, the loss function (Equation (6.13)) history is plotted against the training epochs, stabilizing around 10^{-4}. This indicates successful training of the PINNs model. The results shown in Figures 6.7 and 6.8 demonstrate the capability of our proposed framework to accurately predict the spreading radius of CMAS across the different initial radii and equilibrium contact angles.

Each column in Figure 6.9 shows the values of $\eta(R_0, \theta_{eq})$, $\beta(R_0, \theta_{eq})$, and $\tau(R_0, \theta_{eq})$ obtained by PINNs for different initial radii R_0 ranging from 0.136 to 0.204 mm and equilibrium contact angles θ_{eq} ranging from 39.1° to 93.4°. The results show that η changes within a small window between −0.325 and −0.200 for all R_0 and θ_{eq}. However, the changes in β (between 1 and 5) and η (between 6.0 and 8.0) are significant, indicating that these parameters are strongly depend on R_0 and θ_{eq}.

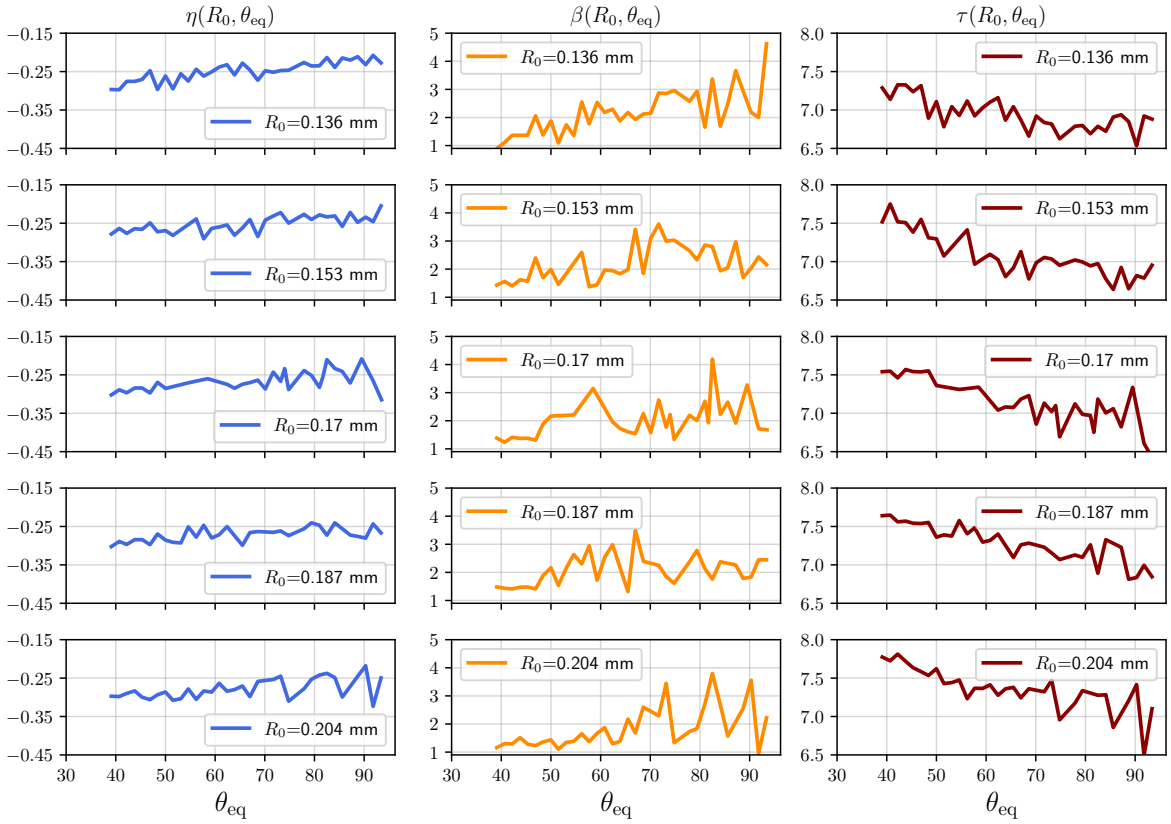


Figure 6.9: The values of η, β, and τ obtained through PINNs. These values exhibit varying behavior depending on the initial radius R_0 and equilibrium contact angles θ_{eq}. The horizontal axes display the equilibrium contact angles θ_{eq}. The vertical axes of all figures represent the values of η and β, and τ. η remains nearly constant within a small range of values between -0.325 and -0.200 and β as well as τ change within a range of 1.0 to 5.0 and 6.5 to 8.0, respectively.

6.4.2 Generate more samples of feasible radii and contact angles

As discussed earlier, the parameters in our ODE model (Equation (6.12)) are functions of the initial radius and the equilibrium contact angle. Using PINNs, we were able to find the values for those parameters. To find a closed-form relation between the parameters, R_0, and θ_{eq}, more data than the rather small current set is needed. Performing three-dimensional mDPD simulations are, however, computationally expensive. In this section, we train two additional neural networks to capture the nonlinear relation between the ODE parameters found by PINNs, and the variables R_0 and for θ_{eq}. Then, we will use these trained networks to fill our sparse parameter space to perform symbolic regression in the next section.

Specifically, we generate values for R_0 in the interval $[0.136\,\text{mm}, 0.204\,\text{mm}]$ and θ_{eq} in the range $[40°, 95°]$, as shown in Figure 6.6 (II). Two fully connected networks, NN_β and NN_τ consist of eight dense layers with $256/256/256/128/64/32/16/8$ neurons. These networks are trained using an Adam optimizer with a learning rate of 10^{-2} for a total of $4,000$ epochs.

The parameter values obtained from NN_τ and NN_β are visualized in Figures 6.10 (a) and (b), respectively. The parameter values obtained from PINNs are denoted by green and red circles, indicating the training data for NN_β and NN_τ, respectively. The parameter values generated by the networks are depicted as light orange and light blue dots. Visually, it is evident that these dots have filled the gaps between parameters that were absent in our LAMMPS dataset.

6.5 Symbolic regression

In this section, we use symbolic regression to find the explicit relation between the ODE parameters, and the initial radii and equilibrium contact angles. Symbolic regression is a technique used in empirical modelling to discover mathematical expressions or symbolic formulas that best fit a given dataset [261]. The process of symbolic regression involves searching a space of mathematical expressions to find the equation that best fits the data. The search is typically

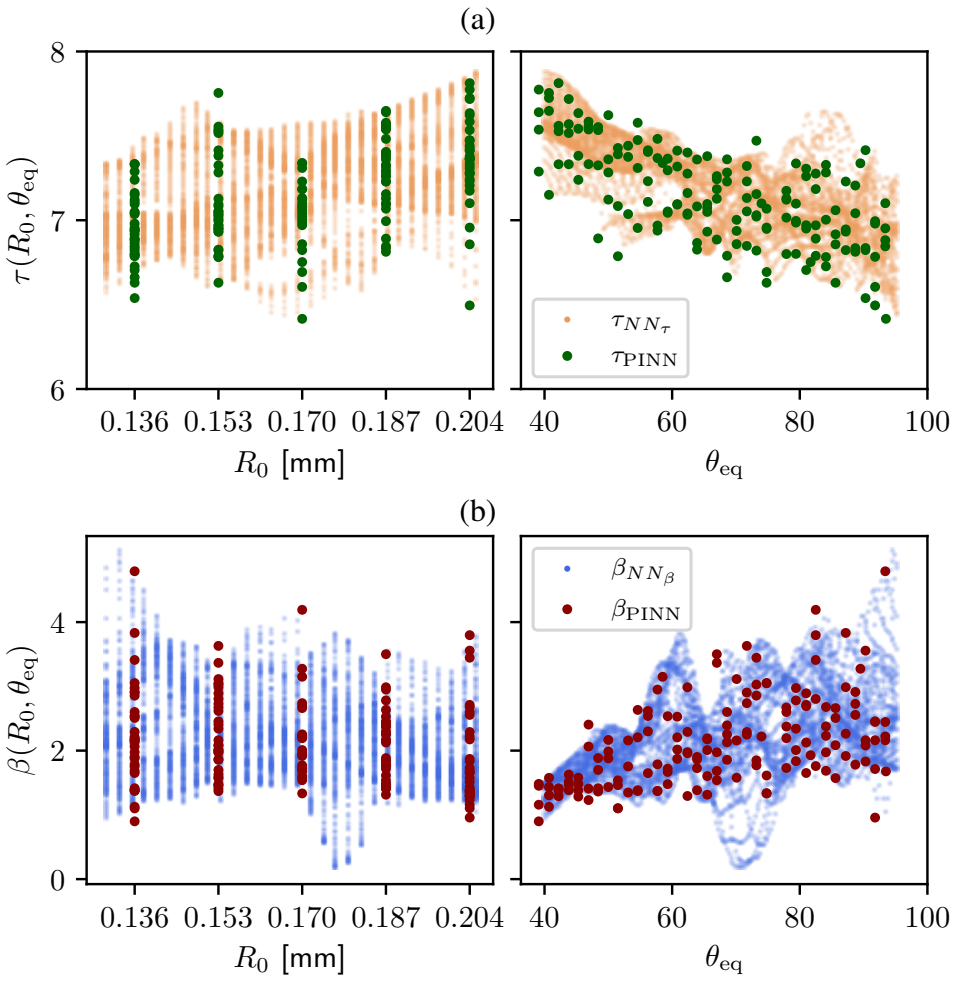


Figure 6.10: Predictions of the parameters (a) τ and (b) β using the trained neural networks NN_β and NN_τ. The horizontal axes show R_0 and θ_{eq}. The green and red circles correspond to the obtained values of τ and β using the PINNs that were used to train NN_β and NN_τ. Additionally, the orange and blue dots represent the predicted values for grid interpolations between $R_0 = 1.3$ to $R_0 = 2.05$ and $\theta_{eq} = 40°$ to $\theta_{eq} = 95°$.

guided by a fitness function that measures the goodness of fit between the equation and the data. The fitness function is optimized using various techniques such as genetic algorithms, gradient descent, or other optimization algorithms. The equations discovered by symbolic regression are expressed in terms of familiar (i.e., more common) mathematical functions and variables, which can be easily understood and interpreted by humans.

In this study, we used the Python library `gplearn` for symbolic regression [303]. As discussed in Section 6.4.2, in order to have more accurate formulation for the ODE parameters before using symbolic regression, we trained two networks, NN_β and NN_τ, using the discovered values $\beta(R_0, \theta_{eq})$, and $\tau(R_0, \theta_{eq})$ enabling us to predict the parameter values for grid interpolations where no corresponding data points were available. The predicted parameters from both PINNs and the NN_β and NN_τ networks are fed through the symbolic regression model to discover a mathematical formulation for each parameter. For this purpose, θ_{eq} and R_0 are fed as inputs, and η, β, and τ are the outputs. We set the population size to $5,000$ and evolve 20 generations until the error is close to 1%. Since the equation consists of basic operations such as addition, subtraction, multiplication, and division, we do not require any custom functions.

The following results of symbolic regression can be substituted in Equation (6.12),

$$\begin{aligned} \eta &= -0.255 \\ \beta &= 0.283 + 0.27\left(\frac{\theta_{eq}}{d}\right) \\ \tau &= 6.13\left(\frac{d}{\theta_{eq}} + 1\right), \end{aligned} \tag{6.14}$$

where d is the initial size of the droplet in mDPD units.

Figure 6.11 shows the history of $\alpha(t, R_0, \theta_{eq})$ using the the ODE (Equation (6.12)) with parameters from Equation (6.14). The conversion between d in mDPD units and R_0 in physical unit is $R_0 = d \times 1.701 \times 10^{-2}$ mm.

In Figure 6.12, the left panel depicts the values of α for an initial drop size of R_0 =

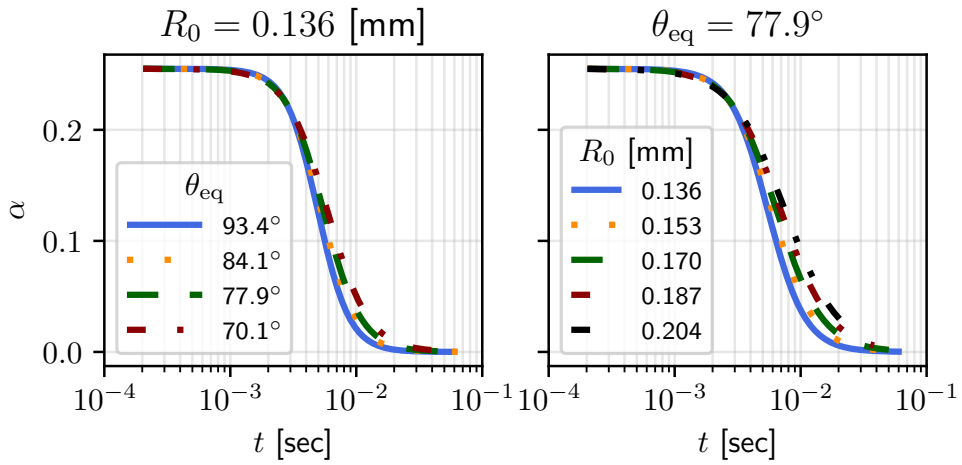


Figure 6.11: The behaviour of α from RHS of Equation (6.12) with parameters from Equation (6.14). Left: different contact angles with fixed initial radius $R_0 = 0.136\,\text{mm}$. Right: varying initial radii with fixed contact angle $\theta_{eq} = 77.9°$.

0.127 mm and contact angles $\theta_{eq} = 93.4°$ and $\theta_{eq} = 87.2°$. The right panel compares the solution of Equation (6.12) using the discovered parameters, Equation 6.14, with the simulation data. The figure demonstrates the agreement between the ODE solution and the actual simulation results for this particular, unseen data set. It is important to note that this particular drop size lies outside the training interval for initial drop sizes $[0.136, 0.204]$ mm.

6.6 Bayesian physics-informed neural network: B-PINN results

Bayesian Physics-Informed Neural Networks (B-PINNs) integrate the traditional PINN framework with Bayesian Neural Networks (BNNs) [351] to enable quantification of uncertainty in predictions [178]. This framework combines the advantages of BNNs [352] and PINNs to address both forward and inverse nonlinear problems. By choosing a prior over the ODE and network parameters, and by defining a likelihood function, one can find posterior distributions, using Bayes's theorem. B-PINNs offer a robust approach for handling problems containing uncorrelated noise, and they provide aleatoric and epistemic uncertainty quantification on the parameters of neural networks and ODEs.

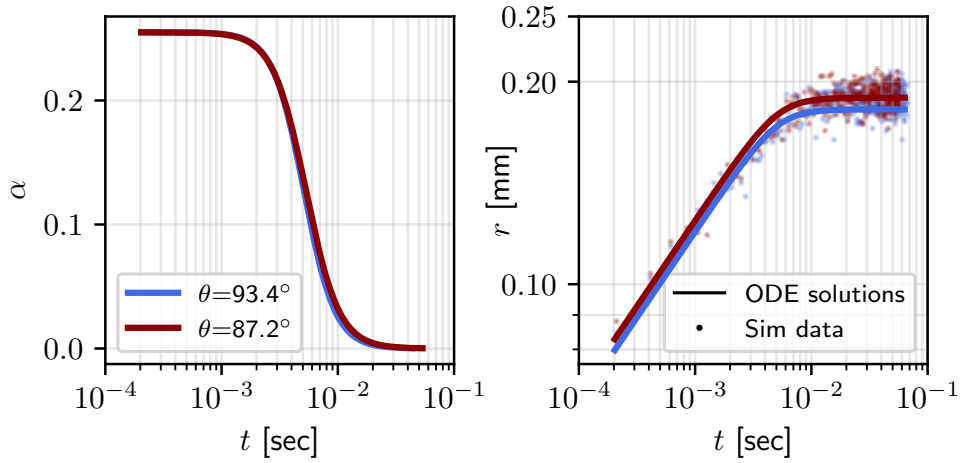


Figure 6.12: The left figure illustrates the behavior of the parameter α using Equation (6.12) for $R_0 = 0.127$ mm, which falls outside the range of the initial drop sizes used for training the networks. On the right panel, the simulation data and the solution obtained from solving the ODE (Equation (6.12)) with parameters from symbolic regression, Equation (6.14) are shown.

The BNN component of the prior adopts Bayesian principles by assigning probability distributions to the weights and biases of the neural network. To account for noise in the data, we add noise to the likelihood function. By applying Bayes' rule, we can estimate the posterior distribution of the model and the ODE parameters. This estimation process enables the propagation of uncertainty from the observed data to the predictions made by the model. We write Equation (6.12) as

$$\mathcal{N}_t(r;\boldsymbol{\lambda}) = f(t), \quad t \in \mathbb{R}^+ \tag{6.15a}$$

$$\mathcal{I}(r,\boldsymbol{\lambda}) = r_0, \quad t = 0, \tag{6.15b}$$

where $\boldsymbol{\lambda} = [\eta,\ \beta, \tau]^\top$ is a vector of the parameters of the ODE (Equation (6.12)), and $\mathcal{N}_t$ is a general differential operator. $f(t)$ is the forcing term, and $\mathcal{I}$ is the initial condition. This problem is an inverse problems, $\boldsymbol{\lambda}$ is inferred from the data with estimates on aleatoric and

epistemic uncertainties. The likelihoods of simulation data and ODE parameters are given as

$$
\begin{aligned}
P(\mathcal{D}\,|\,\boldsymbol{\theta},\boldsymbol{\lambda}) &= P\left(\mathcal{D}_r \mid \boldsymbol{\theta}\right) P\left(\mathcal{D}_f \mid \boldsymbol{\theta},\boldsymbol{\lambda}\right) P\left(\mathcal{D}_I \mid \boldsymbol{\theta},\boldsymbol{\lambda}\right), \text{ where} \\
P(\mathcal{D}_r\,|\,\boldsymbol{\theta},\boldsymbol{\lambda}) &= \prod_{i=1}^{N_r} \frac{1}{\sqrt{2\pi\sigma_r^{(i)^2}}} \exp\left[-\frac{\left(r(\boldsymbol{t}_r^{(i)};\boldsymbol{\theta},\boldsymbol{\lambda}) - \bar{r}^{(i)}\right)^2}{2\sigma_r^{(i)^2}}\right], \\
P\left(\mathcal{D}_f\,|\,\boldsymbol{\theta},\boldsymbol{\lambda}\right) &= \prod_{i=1}^{N_f} \frac{1}{\sqrt{2\pi\sigma_f^{(i)^2}}} \exp\left[-\frac{\left(f(\boldsymbol{t}_f^{(i)};\boldsymbol{\theta},\boldsymbol{\lambda}) - \bar{f}^{(i)}\right)^2}{2\sigma_f^{(i)^2}}\right], \\
P(\mathcal{D}_I\,|\,\boldsymbol{\theta},\boldsymbol{\lambda}) &= \prod_{i=1}^{N_I} \frac{1}{\sqrt{2\pi\sigma_I^{(i)^2}}} \exp\left[-\frac{\left(I(\boldsymbol{t}_i^{(i)};\boldsymbol{\theta},\boldsymbol{\lambda}) - \bar{I}^{(i)}\right)^2}{2\sigma_I^{(i)^2}}\right],
\end{aligned}
\tag{6.16}
$$

where $D = D_r \cup D_f \cup D_I$ with $\mathcal{D}_r = \left\{\left(\ln t_r^{(i)}, \ln \bar{r}^{(i)}\right)\right\}_{i=1}^{N_r}$, $\mathcal{D}_f = \left\{\left(t_f^{(i)}, f^{(i)}\right)\right\}_{i=1}^{N_f}$, $\mathcal{D}_I = \left\{\left(t_I^{(i)}, I^{(i)}\right)\right\}_{i=1}^{N_I}$ are scattered noisy measurements. The joint posterior of $[\boldsymbol{\theta},\boldsymbol{\lambda}]$ is given as

$$
\begin{aligned}
P(\boldsymbol{\theta},\boldsymbol{\lambda} \mid \mathcal{D}) &= \frac{P(\mathcal{D} \mid \boldsymbol{\theta},\boldsymbol{\lambda})P(\boldsymbol{\theta},\boldsymbol{\lambda})}{P(\mathcal{D})} \\
&\simeq P(\mathcal{D} \mid \boldsymbol{\theta},\boldsymbol{\lambda})P(\boldsymbol{\theta},\boldsymbol{\lambda}) \\
&= P(\mathcal{D} \mid \boldsymbol{\theta},\boldsymbol{\lambda})P(\boldsymbol{\theta})P(\boldsymbol{\lambda}).
\end{aligned}
\tag{6.17}
$$

To sample the parameters from the the posterior probability distribution defined by Equation (6.17), we utilized the Hamiltonian Monte Carlo (HMC) approach [353], which is an efficient Markov Chain Monte Carlo (MCMC) method [354]. For a detailed description of the method, please see, e.g., Refs. [355, 356, 357]. To sample the posterior probability distribution, however, variational inference [358] could be also used. In variational inference, the posterior density of the unknown parameter vector is approximated by another parameterized density function, which is restricted to a smaller family of distributions [178]. To compute the uncertainty in the ODE parameters by using B-PINN, a noise of 5% was added to the original data set. The noise was sampled from a normal distribution with a mean of 0 and standard deviation of ± 1.

Here, the neural network model architecture comprises of two hidden layers, each containing 50 neurons. The network takes time (t) as the input, and generates a droplet radius $r(t)$ as the output. Additionally, we include a total of 2,000 burn-in samples.

The computational expense of B-PINNs compared to traditional neural networks primarily arises from the iterative nature of Bayesian inference and the need to sample from the posterior distribution. B-PINNs involve iterative Bayesian inference, where the posterior distribution is updated iteratively based on observed data. This iterative process requires multiple iterations to converge to a stable solution, leading to increased computational cost compared to non-iterative methods. Moreover, B-PINNs employ sampling-based algorithms such as MCMC or variational inference to estimate the posterior distribution of the model parameters. These algorithms generate multiple samples from the posterior distribution, which are used to approximate uncertainty and infer calibrated parameters.

Sampling from the posterior distribution can be computationally expensive, particularly for high-dimensional parameter spaces or complex physics models. Furthermore, B-PINNs often require running multiple forward simulations of the physics-based model for different parameter samples. Each simulation represents a potential configuration of the model parameters. Since physics-based simulations can be computationally intensive, conducting multiple simulations significantly increases the computational cost of training B-PINNs. Achieving a high acceptance rate for posterior samples, especially for high-dimensional data, demands running a large number of simulations. This further adds to the computational complexity.

Due to the computational expense associated with B-PINNs, we opt to use the method selectively for a few cases only. Utilizing the insights gained from PINNs prediction and symbolic regression, specifically the known value of $\eta = -0.255$, we can leverage the power of B-PINNs to uncover and ascertain the values of the parameters β and τ. Figure 6.13 showcases the comparison between the mean values of the radii $\tilde{r}(t)$ predicted by B-PINNs represented by solid lines, the corresponding standard deviations denoted by highlighted regions, and the simulation data used for training is presented as stars, while the test data is indicated by colored

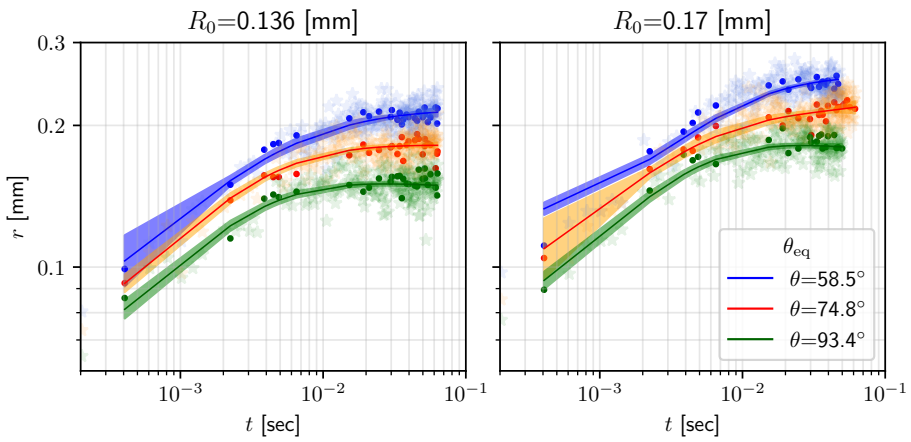


Figure 6.13: The mean and uncertainty (mean ± 2 standard deviation) of B-PINN predictions of the spreading radii history are given as solid lines and shaded regions, respectively. The test simulation data is depicted by solid circles and training data is indicated by stars. This analysis is carried out for two different initial drop sizes, namely $R_0 = 0.137$ mm and 0.170 mm, for three equilibrium contact angles.

circles. The horizontal axes represent time, while the vertical axes depict the spreading radii $\tilde{r}(t)$. This comparative analysis is conducted for two distinct initial drop sizes, namely $R_0 = 0.137$ mm and 0.170 mm, considering various equilibrium contact angles.

Figure 6.14 illustrates the mean values of the parameters β and τ obtained using B-PINNs along with their corresponding standard deviations. The solid lines represent the average values of the discovered parameters, while the highlighted regions indicate the standard deviations. The parameters discovered by PINNs are represented by the dashed lines. On the left side, the vertical axes represent the values of β, while the panels on the right side display the values of τ. The results are presented for two initial drop sizes: $R_0 = 0.136$ mm (top panels) and $R_0 = 0.17$ mm (bottom panels). From the figure, it can be observed that the parameter β exhibits a range of values between 1.0 and 3.0. On the other hand, the parameter τ fluctuates within the range of 5.0 to 7.0. These ranges provide insight into the variability and uncertainty associated with the estimated values of β and τ obtained through the B-PINN methodology.

By comparing Figures 6.9 and 6.14, it becomes evident that the discovered parameters β and τ using PINNs of B-PINNs frameworks exhibit remarkable similarity. This striking

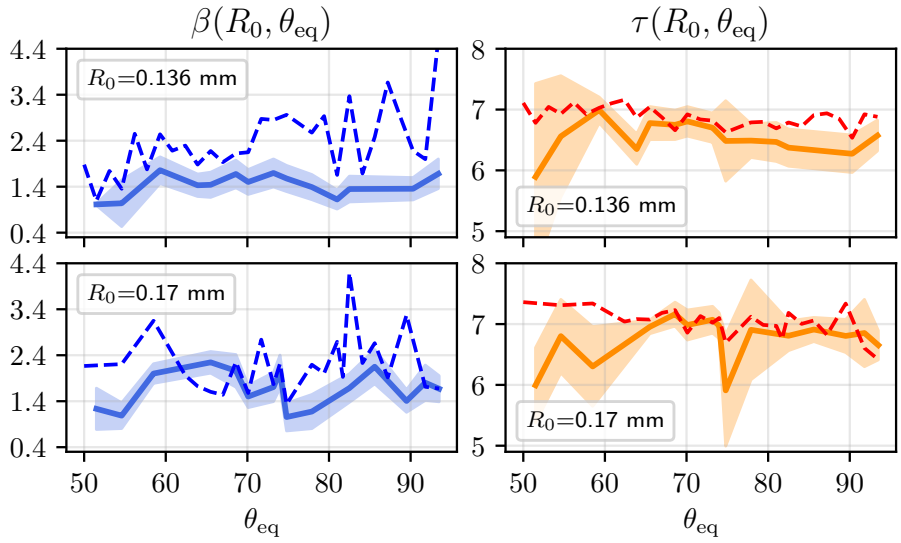


Figure 6.14: Comparison between B-PINNs and PINNs discovered parameters for range of equilibrium contact angles and two initial radii. The mean values (solid lines) and the standard deviations (mean values ± 2 standard deviations, shaded region) of β (left panels) and τ (right panels). The dashed lines represent the parameters discovered by PINNs.

similarity reinforces the efficacy and capability of our models in accurately identifying the parameters of the ODE described in Equation (6.12). The close alignment between the discovered parameters in both figures demonstrates the robustness and reliability of our models. It highlights their ability to effectively capture the underlying dynamics and characteristics of the spreading behavior of CMAS, leading to accurate parameter estimation. This consistency and agreement between the PINN and B-PINN results provide further validation of the power and effectiveness of our modeling approaches in uncovering the true values of the parameters β and τ in the ODE. Additionally, Equation (6.12) with anticipated parameters obtained using B-PINNs is solved using `Odeint`. The results are presented in Figure 6.15, which provides a comparison between the simulated spreading radii $r(t)$ (circles) and the solution of the ODE (solid lines). This comparison is conducted for initial drop sizes of $R_0 = 0.136$ mm and 0.17 mm considering different contact angles.

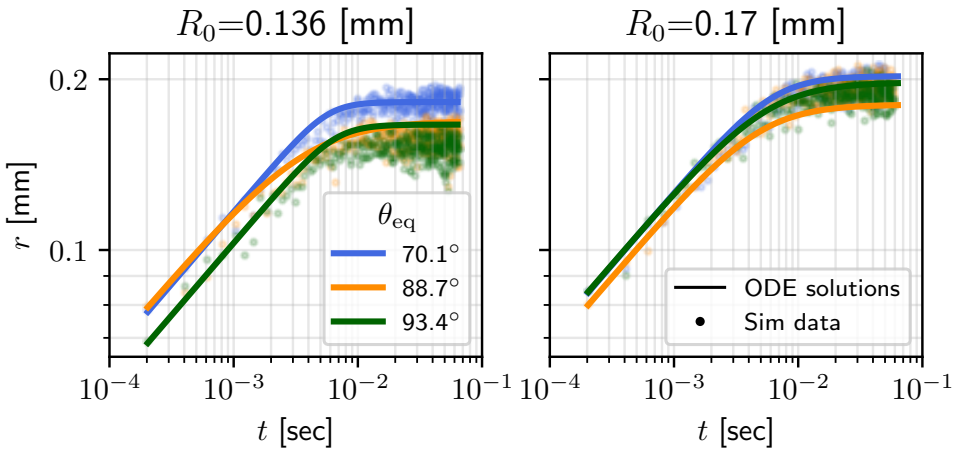


Figure 6.15: Comparison between the ODE solution with parameters found by B-PINNs (solid lines), and the simulation radii (circles). Two initial drop sizes R_0 = 0.137 mm and 0.170 mm and three equilibrium contact angles are shown.

6.7 Conclusions

This study introduces a new approach to model the spreading dynamics of molten CMAS droplets. In the liquid state, CMAS is characterized by high viscosity, density, and surface tension. The main objective is to achieve a comprehensive understanding of the spreading dynamics by integrating the underlying physics into the neural network architecture.

The study emphasizes the potential of PINNs in analyzing complex systems with intricate dynamics. To study the dynamics of CMAS droplets, we performed simulations using the mDPD method. By analyzing the simulation data and observing the droplet behavior, we proposed a coarse parametric equation (Equation (6.12)), which consists of three unknown parameters. This parametric equation aims to capture and describe the observed behavior of the CMAS droplets based on the simulation results. Using the data from the mDPD simulations, the study employed the PINNs framework to determine the parameters of the equation. Symbolic regression was then utilized to establish the relationship between the identified parameter values, and the initial droplet radii and contact angles. As a result, a simplified ODE model was developed, accurately capturing the spreading dynamics. The model's parameters were explicitly determined based on the droplet's geometry and surface properties. Furthermore, B-PINNs were employed to assess the uncertainty associated with the model predictions, pro-

viding a comprehensive analysis of the spreading behavior of CMAS droplets.

In reality, the CMAS attack involves a complex interplay of reaction kinetics between CMAS and the TBC, surface morphology (roughness, porosity) occurring under a highly non-uniform thermal conditions. However, our findings extend beyond the isothermal conditions used in this study and even the specific case of CMAS droplets. The relationships uncovered and methods developed in this study have broader applications in understanding the spreading dynamics of droplets in general. By leveraging the insights gained from this research, one can investigate and understand the behavior of droplets in diverse contexts, furthering our understanding of droplet spreading phenomena. This knowledge can potentially be used in developing strategies for effective droplet management and optimizing processes involving droplets in a wide range of practical applications.

Acknowledgments

EK thanks the Mitacs Globalink Research Award Abroad and Western University's Science International Engagement Fund Award. MK thanks the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs Program. The work was partially supported by DOE grant (DE-SC0023389). RBK would like to acknowledge the support received from the US Army Research Office Mathematical Sciences Division for this research through grant number W911NF-17-S-0002. LB and AG were supported by the US Army Research Laboratory 6.1 basic research program in propulsion sciences. ZL and GK acknowledge the support from the AIM for Composites, an Energy Frontier Research Center funded by the U.S. Department of Energy (DOE), Office of Science, Basic Energy Sciences (BES) under Award #DE-SC0023389. ZL also acknowledges support from the National Science Foundation (Grants OAC-2103967 and CDS&E-2204011). Computing facilities were provided by the Digital Research Alliance of Canada (https://alliancecan.ca) and the Center for Computation and Visualization, Brown University. The authors acknowledge the resources

and support provided by Department of Defense Supercomputing Resource Center (DSRC) through use of ”Narwhal” as part of the 2022 Frontier Project, Large-Scale Integrated Simulations of Transient Aerothermodynamics in Gas Turbine Engines.

Chapter 7

Conclusions and future work

7.1 Conclusions

In this thesis, we have presented a comprehensive exploration of data-driven approaches for the discovery of PDEs in the context of black-box in Chapter 4, gray-box in Chapter 5, and white-box modeling in Chapter 6. A wide range of methodologies and techniques to uncover the underlying PDEs based on available data and varying degrees of prior knowledge were studied. The thesis showcases the potential of data-driven approaches, such as MLP in Section 4.3.1, CNN in Section 4.4, CNN-LSTM in Section 4.3.2, X-PINNs in Section 5.3, PINNs in Section 6.4 , and symbolic regression in Section 5.4 in discovering and understanding the behavior of complex systems described by PDEs.

For black-box learning, two scenarios were investigated with a specific focus on understanding and analyzing phase-field dynamics in Chapter 4. These scenarios involved the utilization of well-known models such as Allen-Cahn, Cahn-Hilliard, and the phase-field crystal as test cases. MLP and CNN-LSTM architectures were employed to learn the PDEs when there is an unknown relationship between field evolution and its spatial derivatives. A specialized CNN architecture was proposed for cases where information regarding spatial dependence is unavailable. These algorithms provide a black-box approach for discovering PDEs without

relying on prior knowledge of the physical model's functional form.

We have created an MLP architecture (see Section 4.3.1) capable of learning PDEs by explicitly computing spatial derivatives using finite differences. In addition to spatial derivatives, we also feed time derivatives through the network to learn the right-hand side of the equation. MLP networks have the ability to adapt and work well with various types of data. This includes handling data with low resolution or granularity and their associated spatial derivatives (rates of change in the data with respect to spatial coordinates). MLP networks are flexible enough to effectively process and analyze such diverse data types, making them suitable for tasks involving PDE learning. However, it is extremely important to choose the architecture of the MLP network with care to achieve the best possible performance. The selection of the network's structure and parameters should be done thoughtfully and based on the specific requirements of the task at hand. Additionally, having accurate knowledge of the derivative orders (the order of spatial and temporal derivatives involved in the PDE) is crucial for successful training of the network. This knowledge allows for the appropriate design and configuration of the MLP architecture, ensuring that it can effectively learn and approximate the desired PDEs. However, the choice of architecture for an MLP network can significantly impact its performance. The architecture of an MLP refers to the arrangement and connectivity of its layers, including the number of hidden layers, the number of neurons in each layer, and the activation functions used. Selecting the appropriate architecture depends on various factors, including the specific requirements of the task, available data, computational resources, and desired accuracy. Different architectures can exhibit varying capabilities in handling different types of data and modeling complex relationships. For example, a shallow MLP with only one hidden layer may struggle to capture intricate patterns and relationships in the data, resulting in lower performance. On the other hand, a deeper MLP with multiple hidden layers can potentially learn more abstract and hierarchical representations, enabling it to capture complex dependencies and achieve better performance.

Training an MLP necessitates explicit calculations of spatial derivatives, which can be

computationally expensive and time-consuming. However, CNN networks offer a different approach. In a CNN, the initial convolutional layer is capable of directly estimating derivatives without the need for explicit calculations. This means that CNNs can approximate spatial derivatives in a way that is similar to finite-difference approximations. By estimating derivatives directly in the convolutional layer, CNNs are able to capture the spatial features present in the input data. This is because the convolutional layer applies filters to the input data, identifying patterns and features that are spatially distributed. The filters act as local receptive fields, analyzing the neighboring data points to identify gradients and spatial variations. This ability to estimate derivatives and capture spatial features makes CNNs particularly well-suited for modeling PDEs with complex spatial dependencies. They can learn and represent the underlying patterns and structures in the data without explicitly calculating spatial derivatives. This simplifies the learning process and enhances the network's capability to handle intricate spatial relationships. By eliminating the need for explicit spatial derivative calculations, CNNs offer a more efficient and streamlined approach to PDE learning, especially when compared to traditional methods. However, due to the nature of their architecture, CNNs often demand a significant amount of memory to store the network parameters and intermediate feature maps during training and inference. This can pose challenges, especially when working with limited computational resources or large-scale datasets. Moreover, CNNs typically require a substantial amount of data to achieve optimal performance and accurate results. Training a CNN on a small dataset may lead to overfitting, where the network becomes overly specialized to the training samples and fails to generalize well to new, unseen data. It is important to ensure that the dataset used for training is diverse and representative of the problem domain to avoid overfitting and improve the generalization ability of the network.

In the context of gray-box learning, the approach assumes that we have prior knowledge about certain components or aspects of the equation, while considering other parts as unknown. The primary objective of gray-box learning is to predict and uncover the unknown components of the equation. Gray-box learning allows us to make informed predictions and fill in the gaps

in our understanding by utilizing the known components and integrating them with observed data. we employed a modified version of X-PINNs for gray-box learning of the Allen-Cahn equation in Chapter 5. By dividing the computational domain into four subdomains and utilizing our understanding of the Laplacian part, we successfully uncovered the non-linear component $U-U^3$ of the equation. To reveal the closed mathematical form of the unknown component and its coefficients, we utilized symbolic regression. By applying symbolic regression to our dataset, we aimed to find a mathematical formula that represents the relationship between the known variables and the unknown component. The outcomes of this approach exhibited a strong alignment between the discovered form and the exact form of the equation. This framework proved to be highly effective in accurately identifying the unknown components based on the available data.

In the realm of white box modeling, we developed a novel approach to investigate the spreading dynamics of molten CMAS droplets in Chapter 6. Our method involved integrating the fundamental physics principles into the neural network architecture, allowing us to gain valuable insights into the behavior of CMAS droplets. To accurately capture the droplets' dynamics, we employed the mDPD model for simulations. However, rather than relying solely on the simulation data, we proposed a coarse parametric equation with three unknown parameters. This equation aimed to provide a concise representation of the droplets' behavior while allowing for flexibility in capturing their dynamics. To determine the values of these unknown parameters, we combined the power of PINNs with the available simulation data. By training the PINNs using the simulation data, we were able to infer the optimal parameter values that aligned with the observed behavior of the droplets. This integration of PINNs and simulation data enabled us to gain a comprehensive understanding of the droplet behavior and uncover insights into their spreading dynamics.

7.2 Future work

There are several interesting alleys for future work. For example, it would be interesting to focus on enhancing the performance and efficiency of data-driven models, such as MLPs and CNNs, for PDE discovery. This can be achieved by exploring advanced architectural designs, optimization techniques, and regularization methods that specifically target the unique challenges posed by PDE discovery tasks.

In the case of discovering equations using black-box modeling, extending the methods to more complex PDE systems presents additional challenges. One particularly challenging scenario involves predicting three-dimensional phase-field models. This presents difficulties due to limited computer memory, making efficient network training increasingly problematic. Consequently, it becomes essential to explore frameworks that can handle large datasets to address this issue. MLP requires explicit spatial derivatives, which significantly increases the computational cost when predicting three-dimensional equations. Additionally, the use of CNNs for three-dimensional phase-field models can be computationally expensive. Therefore, alternative approaches need to be explored to overcome these challenges and improve the efficiency of prediction for three-dimensional phase-field models.

In the case of discovering equations using gray-box modeling, the current study primarily focused on specific PDE systems, such as the Allen-Cahn equation. However, due to the properties of PINNs that require spatial derivatives to create residuals in the physics-informed part, dealing with higher-order derivatives like in the Cahn-Hilliard and the phase-field crystal models became either impossible or computationally expensive. In future research, it is crucial to broaden the scope by exploring more complex PDE systems that involve higher-order derivatives, additional variables, and intricate dynamics. To tackle the challenges posed by such systems, employing X-PINNs for both time and space decomposition can be a promising approach. By utilizing X-PINNs, it becomes feasible to effectively address the investigation of equations involving higher-order derivatives. This approach allows for the efficient management of the computational complexity associated with such equations. This advancement

would significantly contribute to advancing our understanding of complex physical phenomena described by PDEs.

By applying similar methodologies to droplet systems other than those presented in Chapter 6, the relationships between the parameters and dynamics can be further explored, providing valuable insights for managing and understanding droplet behavior in various contexts and applications. The power of PINNs, demonstrated in this thesis, opens up new possibilities for investigating and optimizing processes involving droplets and advancing our knowledge in droplet-related phenomena. Further research can involve conducting experimental studies to validate the predictions made by the developed model and refine its parameters. This can be accomplished through the design of controlled experiments specifically aimed at observing and measuring the spreading dynamics under different conditions. This can include different substrate materials, varying surface properties, and changes in temperature or environmental conditions. Through systematic observations and measurements, researchers can gather comprehensive datasets that encompass a range of scenarios, providing valuable insights into the behavior of droplets. By combining simulated and experimental data, a more comprehensive understanding of the dynamics of the system can be achieved, and the discovered PDEs/ODEs can be validated against real-world observations. This evaluation should include assessing the scalability, computational efficiency, and practical usability of the models.

The methodologies and techniques developed in this study have the potential to be applicable to a wide range of domains and applications beyond the scope of the specific systems investigated. In future research, it would be valuable to explore the application of data-driven approaches for discovering PDEs in diverse scientific disciplines, engineering, and industrial processes. Additionally, addressing the challenges and complexities that arise in different applications is essential. These challenges may include handling large-scale datasets, dealing with high-dimensional systems, incorporating domain-specific constraints, or considering specific boundary conditions. These conditions could be of the Dirichlet type (specifying values at boundaries) or the Neumann type (specifying derivatives at boundaries). Adapting the method-

ologies to address these challenges is critical for successfully applying data-driven approaches to PDE discovery in diverse domains.

An important aspect of PDE modeling is the quantification of uncertainties and the assessment of sensitivity in relation to different input parameters and data variations. Some of these issues were addressed in Section 6.6. Future research is needed for developing techniques that effectively quantify uncertainties in data-driven models. This would enable robust predictions and informed decision-making even in the presence of uncertain conditions. As mentioned in Section 6.6, the current implementation of B-PINNs and Bayesian methods in the context of PDE modeling can be computationally expensive. This is primarily due to the inherent complexity and computational requirements associated with Bayesian inference and the integration of uncertainty quantification techniques. Bayesian methods involve evaluating complex probabilistic computations, such as estimating posterior distributions and conducting MCMC sampling. These calculations often require a large number of iterations and computations, making them computationally intensive and time-consuming. By improving the computational efficiency of Bayesian methods, one can enhance the practical applicability and scalability of these techniques in real-world PDE modeling scenarios. This would enable broader adoption of uncertainty-aware modeling approaches, facilitating more reliable predictions and decision-making under uncertain conditions.

The potential future directions discussed above would contribute to more accurate and reliable modeling of complex systems governed by partial differential equations, enabling better predictions and informed decision-making in a wide range of scientific and engineering applications.

In this study, we utilized artificially generated data to evaluate the effectiveness of an inverse method. To delve deeper, we focused on three key objectives: black box modeling, grey box modeling, and white box modeling. These objectives have direct applications in uncovering mathematical equations through the utilization of experimental data in both application phase fields and drop spreading phenomena. Experimental data can stem from a diverse range

of time-dependent experiments. By integrating data from both microscopic and macroscopic sources, the process of equation discovery can be further refined. This approach entails capturing intricate behaviors at the microscopic level, facilitated by techniques such as atomic force microscopy and optical microscopy. Simultaneously, it includes incorporating broader trends observed in macroscopic data, obtained through methods like scanning electron microscopy. Integrating these authentic data sources has the potential to significantly improve the accuracy and comprehensiveness of the equation discovery process. Moreover, it enables us to thoroughly assess the effectiveness of our approaches.

Based on the above discussion, future works can be categorized as follows:

- Performance and efficiency enhancement: Focus on improving the performance and efficiency of data-driven models for PDE discovery. Explore advanced architectural designs and innovative network architectures that are well-suited for PDE discovery. These architectures may include specialized designs that can effectively capture the underlying dynamics and patterns of PDE systems. Develop data-driven models that are capable of efficiently handling large-scale PDE datasets, optimizing computational resources, and improving the accuracy and reliability of PDE predictions.

- Handling higher-dimensional equations: Address the challenges associated with higher-dimensional equations by developing frameworks and methodologies capable of effectively handling and predicting complex PDE systems in three or more dimensions. This involves managing computational complexity and memory requirements while ensuring practical and scalable solutions.

- Handling equations with higher-order Derivatives: Explore techniques to handle equations involving higher-order derivatives. Develop methods that can efficiently learn and predict PDEs with multiple derivatives.

- Integration of experimental data: Enhance the accuracy and practical usability of data-driven models by combining simulated and experimental data. Integrate experimental

observations with simulation data to achieve comprehensive understanding and validation of the discovered PDEs/ODEs.

- Application to diverse domains: Extend the application of data-driven approaches for PDE discovery to diverse scientific disciplines, engineering fields, and industrial processes. Customize and adapt the methodologies to effectively capture and model the unique characteristics of each domain.

- Scalability and generalization: Address challenges related to scalability and generalization of data-driven models. Develop techniques to train and generalize data-driven models effectively with limited data, avoid overfitting, and enhance their performance in diverse scenarios.

- Uncertainty quantification and sensitivity analysis: Develop techniques to accurately measure and propagate uncertainties in data-driven models, specifically in scenarios where we are modeling the impact of environmental factors. Give priority to robust uncertainty quantification and sensitivity analysis to facilitate reliable predictions and informed decision-making, particularly when dealing with uncertain conditions.

Bibliography

[1] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An overview of machine learning," *Mach. Learn.*, pp. 3–23, 1983.

[2] K. Lindorff-Larsen, S. Piana, R. O. Dror, and D. E. Shaw, "How fast-folding proteins fold," *Science*, vol. 334, no. 6055, pp. 517–520, 2011.

[3] M. Karplus, "Molecular dynamics of biological macromolecules: A brief history and perspective," *Biopolymers*, vol. 68, no. 3, pp. 350–358, 2003.

[4] T. Murtola, A. Bunker, I. Vattulainen, M. Deserno, and M. Karttunen, "Multiscale modeling of emergent materials: biological and soft matter," *Phys. Chem. Chem. Phys.*, vol. 11, no. 12, pp. 1869–1892, 2009.

[5] K. R. Elder, M. Katakowski, M. Haataja, and M. Grant, "Modeling elasticity in crystal growth," *Phys. Rev. Lett.*, vol. 88, p. 245701, 2002.

[6] S. A. Silber and M. Karttunen, "SymPhas–General purpose software for phase-field, phase-field crystal, and reaction-diffusion simulations," *Adv. Theory Simul.*, vol. 5, p. 2100351, 2022.

[7] W. B. Hardy, "III. the spreading of fluids on glass," *Lond. Edinb. Dublin philos. mag. j. sci.*, vol. 38, no. 223, pp. 49–55, 1919.

[8] J. A. Nieminen, D. B. Abraham, M. Karttunen, and K. Kaski, "Molecular dynamics of a microscopic droplet on solid surface," *Phys. Rev. Lett.*, vol. 69, no. 1, pp. 124–127, 1992.

[9] M. N. Popescu, G. Oshanin, S. Dietrich, and A.-M. Cazabat, "Precursor films in wetting phenomena," *J. Phys. Condens. Matter*, vol. 24, no. 24, p. 243102, 2012.

[10] R. B. Koneru, A. Flatau, Z. Li, L. Bravo, M. Murugan, A. Ghoshal, and G. E. Karniadakis, "Quantifying the dynamic spreading of a molten sand droplet using multiphase mesoscopic simulations," *Phys. Rev. Fluids*, vol. 7, no. 10, p. 103602, 2022.

[11] ISO, *ISO/IEC 14882:2017: Programming languages—C++*. Geneva, Switzerland: International Organization for Standardization, 5 ed., 2017.

[12] K. Diederik, B. Jimmy, *et al.*, "Adam: A method for stochastic optimization," *ArXiv*, vol. 1412.6980, pp. 273–297, 2014.

[13] A. Kaplan and M. Haenlein, "Siri, siri, in my hand: Who's the fairest in the land? on the interpretations, illustrations, and implications of artificial intelligence," *Bus. Horiz.*, vol. 62, no. 1, pp. 15–25, 2019.

[14] J. F. Allen, "Ai growing up: The changes and opportunities," *AI magazine*, vol. 19, no. 4, pp. 13–13, 1998.

[15] D. Kirsh, "Foundations of ai: the big issues," *Artif. Intell.*, vol. 47, no. 1-3, pp. 3–30, 1991.

[16] A. Newell, J. C. Shaw, and H. A. Simon, "The processes of creative thinking.," in *Contemporary Approaches to Creative Thinking, 1958, University of Colorado, CO, US; This paper was presented at the aforementioned symposium.*, Atherton Press, 1962.

[17] A. Turing, "Programmers," *Handbook for Manchester Electronic Computer', University of Manchester Computing Laboratory. A digital facsimile of the original may be viewed*

in The Turing Archive for the History of Computing document.¡ http://www. AlanTuring. net/programmers_handbook, 1950.

[18] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.

[19] A. M. Turing, "Computing machinery and intelligence," *Creative Computing*, vol. 6, no. 1, pp. 44–53, 1980.

[20] A. M. Turing, "Lecture to the london mathematical society on 20 february 1947," *MD COMPUTING*, vol. 12, pp. 390–390, 1995.

[21] A. Turing, "Intelligent machinery (1948)," *B. Jack Copeland*, p. 395, 2004.

[22] N. Muthukrishnan, F. Maleki, K. Ovens, C. Reinhold, B. Forghani, and R. Forghani, "Brief history of artificial intelligence," vol. 30, pp. 393–399, Elsevier, 2020.

[23] M. Minsky and S. A. Papert, "Proposal to arpa for research on artificial intelligence at mit, 1970-1971," 1970.

[24] C. S. Navinchandra, "Neural networks and physical systems with emergent collective computational abilities," *Education*, vol. 3, no. 6, 2012.

[25] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *PNAS*, vol. 81, no. 10, pp. 3088–3092, 1984.

[26] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biol. Cybern.*, vol. 52, no. 3, pp. 141–152, 1985.

[27] H. S. Seung, "Continuous attractors and oculomotor control," *Neural Netw.*, vol. 11, no. 7-8, pp. 1253–1258, 1998.

[28] J. Šíma and P. Orponen, "Continuous-time symmetric hopfield nets are computationally universal," *Neural Comput.*, vol. 15, no. 3, pp. 693–733, 2003.

[29] T. Kohonen, "Self-organized formation of topologically correct feature maps," *Biol. Cybern.*, vol. 43, no. 1, pp. 59–69, 1982.

[30] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev*, vol. 3, no. 3, pp. 210–229, 1959.

[31] K. Santosh, S. Antani, D. Guru, and N. Dey, *Medical Imaging: Artificial Intelligence, Image Recognition, and Machine Learning Techniques*. New York, USA: CRC Press, 2019.

[32] A. Kulkarni and A. Shivananda, *Natural language processing recipes*. New York, USA: Springer, 2019.

[33] J. Mayr, C. Unger, and F. Tombari, "Self-supervised learning of the drivable area for autonomous vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 362–369, IEEE, Madrid, Spain, 2018.

[34] I. El Naqa and M. J. Murphy, *What Is Machine Learning?*, pp. 3–11. Cham: Springer, 2015.

[35] K. Wagstaff, "Machine learning that matters," *arXiv preprint arXiv:1206.4656*, 2012.

[36] A. Famili, W.-M. Shen, R. Weber, and E. Simoudis, "Data preprocessing and intelligent data analysis," *Intell. Data Anal.*, vol. 1, no. 1, pp. 3–23, 1997.

[37] P. Benardos and G.-C. Vosniakos, "Optimizing feedforward artificial neural network architecture," *Eng. Appl. Artif. Intell.*, vol. 20, no. 3, pp. 365–382, 2007.

[38] P. Cunningham, M. Cord, and S. J. Delany, *Supervised Learning*, pp. 21–49. Berlin, Heidelberg, Germany: Springer, 2008.

[39] V. Nasteski, "An overview of the supervised machine learning methods," *Horizons. b*, vol. 4, pp. 51–62, 2017.

[40] F. Ponulak and A. Kasiński, "Supervised learning in spiking neural networks with resume: sequence learning, classification, and spike shifting," *Neural Comput.*, vol. 22, no. 2, pp. 467–510, 2010.

[41] S. Loussaief and A. Abdelkrim, "Machine learning framework for image classification," in *7th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, pp. 58–61, IEEE, 2016.

[42] H. B. Barlow, "Unsupervised learning," *Neural Comput.*, vol. 1, no. 3, pp. 295–311, 1989.

[43] K. J. Cios, R. W. Swiniarski, W. Pedrycz, L. A. Kurgan, K. J. Cios, R. W. Swiniarski, W. Pedrycz, and L. A. Kurgan, "Unsupervised learning: association rules," *Data Mining: A Knowledge Discovery Approach*, pp. 289–306, 2007.

[44] I. Krol, F. D. Schwab, R. Carbone, M. Ritter, S. Picocci, M. L. De Marni, G. Stepien, G. M. Franchi, A. Zanardi, M. D. Rissoglio, *et al.*, "Detection of clustered circulating tumour cells in early breast cancer," *Br. J. Cancer.*, vol. 125, pp. 23–27, 2021.

[45] N. A. Mat-Isa, M. Y. Mashor, and N. H. Othman, "An automated cervical pre-cancerous diagnostic system," *Artif. Intell. Med.*, vol. 42, no. 1, pp. 1–11, 2008.

[46] T. S. Madhulatha, "An overview on clustering methods," *arXiv preprint arXiv:1205.1117*, 2012.

[47] F. Nielsen and F. Nielsen, "Hierarchical clustering," *Introduction to HPC with MPI for Data Science*, pp. 195–211, 2016.

[48] M. E. Celebi, *Partitional clustering algorithms*. Springer, New York, USA, 2014.

[49] K. Krishna and M. N. Murty, "Genetic k-means algorithm," *IEEE Trans. Syst. Man. Cybern.*, vol. 29, no. 3, pp. 433–439, 1999.

[50] N. K. Kaur, U. Kaur, and D. Singh, "K-medoid clustering algorithm-a review," *Int. J. Comput. Appl. Technol*, vol. 1, no. 1, pp. 42–45, 2014.

[51] M. Afzali and S. Kumar, "Text document clustering: issues and challenges," in *International conference on machine learning, big data, cloud and parallel computing (COMIT-Con)*, pp. 263–268, IEEE, Faridabad, India, 2019.

[52] M. Karthikeyan and P. Aruna, "Probability based document clustering and image clustering using content-based image retrieval," *Appl. Soft Comput.*, vol. 13, no. 2, pp. 959–966, 2013.

[53] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, USA), pp. 1928–1937, PMLR, 2016.

[54] B. Heidenreich, "An introduction to positive reinforcement training and its benefits," *J. Exot. Pet Med.*, vol. 16, no. 1, pp. 19–23, 2007.

[55] L. L. Thompson, E. D. Claus, S. K. Mikulich-Gilbertson, M. T. Banich, T. Crowley, T. Krmpotich, D. Miller, and J. Tanabe, "Negative reinforcement learning is affected in substance dependence," *Drug and alcohol dependence*, vol. 123, no. 1-3, pp. 84–90, 2012.

[56] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013.

[57] A. Coronato, M. Naeem, G. De Pietro, and G. Paragliola, "Reinforcement learning for intelligent healthcare applications: A survey," *Artif. Intell. Med.*, vol. 109, p. 101964, 2020.

[58] S. Aradi, "Survey of deep reinforcement learning for motion planning of autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 2, pp. 740–759, 2020.

[59] S. R. Granter, A. H. Beck, and D. J. Papke Jr, "Alphago, deep learning, and the future of the human microscopist," *Arch. Path. Lab.*, vol. 141, no. 5, pp. 619–621, 2017.

[60] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[61] C. O. S. Sorzano, J. Vargas, and A. P. Montano, "A survey of dimensionality reduction techniques," *arXiv preprint arXiv:1403.2877*, 2014.

[62] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 1–45, 2017.

[63] T. R. Reed and J. H. Dubuf, "A review of recent texture segmentation and feature extraction techniques," *CVGIP: Image understanding*, vol. 57, no. 3, pp. 359–372, 1993.

[64] L. Meneghetti, N. Demo, and G. Rozza, "A dimensionality reduction approach for convolutional neural networks," *Appl. Intell.*, pp. 1–16, 2023.

[65] S. Hijazi, R. Kumar, C. Rowen, *et al.*, "Using convolutional neural networks for image recognition," *Cadence Design Systems Inc.: San Jose, CA, USA*, vol. 9, p. 1, 2015.

[66] Y. Goldberg, "A primer on neural network models for natural language processing," *Artif. Intell. Res.*, vol. 57, pp. 345–420, 2016.

[67] C.-X. Feng and X.-F. Wang, "Surface roughness predictive modeling: neural networks versus regression," *IIE Trans* ., vol. 35, no. 1, pp. 11–27, 2003.

[68] D. Chung, "Machine learning for predictive model in entrepreneurship research: predicting entrepreneurial action," *Small Enterp. Res.*, pp. 1–18, 2023.

[69] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, 2015.

[70] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, (Munich, Germany), pp. 234–241, Springer, 2015.

[71] J. Ker, L. Wang, J. Rao, and T. Lim, "Deep learning applications in medical image analysis," *Ieee Access*, vol. 6, pp. 9375–9389, 2017.

[72] T. Epelbaum, F. Gamboa, J.-M. Loubes, and J. Martin, "Deep learning applied to road traffic speed forecasting," *arXiv preprint arXiv:1710.08266*, 2017.

[73] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[74] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[75] A. Kag, Z. Zhang, and V. Saligrama, "Rnns incrementally evolving on an equilibrium manifold: A panacea for vanishing and exploding gradients?," in *International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30, 2020.*

[76] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci.*, vol. 6, no. 12, pp. 310–316, 2017.

[77] H. Pratiwi, A. P. Windarto, S. Susliansyah, R. R. Aria, S. Susilowati, L. K. Rahayu, Y. Fitriani, A. Merdekawati, and I. R. Rahadjeng, "Sigmoid activation function in selecting the best model of artificial neural networks," in *J. Phys. Conf. Ser.*, vol. 1471, p. 012010, IOP Publishing, 2020.

[78] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *Int. J. Artif. Intell.*, vol. 1, no. 4, pp. 111–122, 2011.

[79] K. Eckle and J. Schmidt-Hieber, "A comparison of deep networks with relu activation function and linear spline-type methods," *Neural Netw.*, vol. 110, pp. 232–242, 2019.

[80] B. Paria, C.-K. Yeh, I. E. Yen, N. Xu, P. Ravikumar, and B. Póczos, "Minimizing flops to learn efficient sparse representations," *arXiv preprint arXiv:2004.05665*, 2020.

[81] N. N. Prasad and J. N. Rao, "The estimation of the mean squared error of small-area estimators," *Am. Stat. Assoc. Bull.*, vol. 85, no. 409, pp. 163–171, 1990.

[82] D. P. Mandic, "A generalized normalized gradient descent algorithm," *IEEE Signal Process. Lett.*, vol. 11, no. 2, pp. 115–118, 2004.

[83] P. Murugan, "Feed forward and backward run in deep convolution neural network," *arXiv preprint arXiv:1711.03278*, 2017.

[84] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE winter conference on applications of computer vision (WACV)*, pp. 464–472, IEEE, 2017.

[85] Y. Bengio, *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pp. 437–478. Berlin, Heidelberg, Germany: Springer, 2012.

[86] F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of," *Brain Mechanisms*, pp. 555–559, 1962.

[87] M. Braun and M. Golubitsky, *Differential equations and their applications*, vol. 2. Springer, New York, USA, 1983.

[88] E. L. Ince, "Ordinary differential equations," (New York, USA), Courier Corporation, 1956.

[89] K. Omori and J. Kotera, "Overview of pdes and their regulation," *Circ. Res.*, vol. 100, no. 3, pp. 309–327, 2007.

[90] L. F. Shampine, "Conservation laws and the numerical solution of odes," *Comput. Math. with Appl.*, vol. 12, no. 5-6, pp. 1287–1296, 1986.

[91] D.-E. Gratie, B. Iancu, and I. Petre, "Ode analysis of biological systems," *Formal Methods for Dynamical Systems: 13th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2013, Bertinoro, Italy, June 17-22, 2013. Advanced Lectures*, pp. 29–62, 2013.

[92] D. A. Charlebois and G. Balázsi, "Modeling cell population dynamics," *In Silico Biol.*, vol. 13, no. 1-2, pp. 21–39, 2019.

[93] D. J. Higham, "Modeling and simulating chemical reactions," *SIAM review*, vol. 50, no. 2, pp. 347–368, 2008.

[94] K. Nakkeeran, "Mathematical description of differential equation solving electrical circuits," *J. Circuits Syst. Comput.*, vol. 18, no. 05, pp. 985–991, 2009.

[95] M. H. Zawawi, A. Saleha, A. Salwa, N. Hassan, N. M. Zahari, M. Z. Ramli, and Z. C. Muda, "A review: Fundamentals of computational fluid dynamics (cfd)," in *AIP conference proceedings*, vol. 2030, p. 020252, AIP Publishing LLC, 2018.

[96] T. Rabczuk, H. Ren, and X. Zhuang, "A nonlocal operator method for partial differential equations with application to electromagnetic waveguide problem," *Comput. Mater. Contin.*, pp. 31–55, 2019.

[97] D. Heath and M. Schweizer, "Martingales versus pdes in finance: an equivalence result with examples," *J. Appl. Probab.*, vol. 37, no. 4, pp. 947–957, 2000.

[98] A. D. Polyanin and A. I. Zhurov, *Separation of variables and exact solutions to nonlinear PDEs*. Boca Raton, FL, USA: CRC Press, 2021.

[99] J. C. Strikwerda, "Finite difference schemes and partial differential equations," SIAM, Philadelphia, PA, 2004.

[100] J. W. Thomas, *Numerical partial differential equations: finite difference methods*, vol. 22. Springer, New York, USA, 2013.

[101] K.-J. Bathe, "Finite element method," *Wiley encyclopedia of computer science and engineering*, pp. 1–12, 2007.

[102] J. Shen, T. Tang, and L.-L. Wang, *Spectral methods: algorithms, analysis and applications*, vol. 41. Springer Berlin, Heidelberg, Germany, 2011.

[103] D. W. Zingg and T. T. Chisholm, "Runge–kutta methods for linear ordinary differential equations," *Appl. Numer. Math*, vol. 31, no. 2, pp. 227–238, 1999.

[104] S. Yoon, D. Jeong, C. Lee, H. Kim, S. Kim, H. G. Lee, and J. Kim, "Fourier-spectral method for the phase-field equations," *Mathematics*, vol. 8, 2020.

[105] L. Q. Chen and J. Shen, "Applications of semi-implicit Fourier-spectral method to phase field equations," *Comput. Phys. Commun.*, vol. 108, pp. 147–158, 1998.

[106] M. Heinonen, C. Yildiz, H. Mannerström, J. Intosalmi, and H. Lähdesmäki, "Learning unknown ode models with gaussian processes," in *International conference on machine learning*, pp. 1959–1968, PMLR, 2018.

[107] Y. Chen, B. Hosseini, H. Owhadi, and A. M. Stuart, "Solving and learning nonlinear pdes with gaussian processes," *J. Comput. Phys.*, vol. 447, p. 110668, 2021.

[108] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proc. Natl. Acad. Sci. USA*, vol. 115, pp. 8505–8510, 2018.

[109] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans. Neural Netw.*, vol. 9, pp. 987–1000, 1998.

[110] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *J. Comput. Phys.*, vol. 375, pp. 1339–1364, 2018.

[111] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE trans. neural netw.*, vol. 9, no. 5, pp. 987–1000, 1998.

[112] M. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *Commun. numer. methods eng.*, vol. 10, pp. 195–201, 1994.

[113] Z. Liu, Y. Yang, and Q. Cai, "Neural network as a function approximator and its application in solving differential equations," *Appl. Math. Mech.*, vol. 40, pp. 237–248, 2019.

[114] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, 2019.

[115] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proc. Natl. Acad. Sci.*, vol. 113, pp. 3932–3937, 2016.

[116] H. Schaeffer, "Learning partial differential equations via data discovery and sparse optimization," *Proc. R. Soc. Math. Phys. Eng. Sci.*, vol. 473, p. 20160446, 2017.

[117] S. Lee, M. Kooshkbaghi, K. Spiliotis, C. I. Siettos, and I. G. Kevrekidis, "Coarse-scale PDEs from fine-scale observations via machine learning," *Chaos Interdiscip. J. Nonlinear Sci.*, vol. 30, p. 013141, 2020.

[118] J. Bakarji and D. M. Tartakovsky, "Data-driven discovery of coarse-grained equations," *J. Comput. Phys.*, vol. 434, p. 110219, 2021.

[119] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Data-driven discovery of partial differential equations," *Sci. Adv.*, vol. 3, p. e1602614, 2017.

[120] Z. Long, Y. Lu, X. Ma, and B. Dong, "Pde-net: Learning pdes from data," in *International conference on machine learning*, pp. 3208–3216, PMLR, 2018.

[121] Z. Long, Y. Lu, and B. Dong, "PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network," *J. Comp. Phys.*, vol. 399, p. 108925, 2019.

[122] J. Qu, W. Cai, and Y. Zhao, "Learning time-dependent PDEs with a linear and nonlinear separate convolutional neural network," *J. Comp. Phys.*, vol. 453, p. 110928, 2022.

[123] U. Fasel, E. Kaiser, J. N. Kutz, B. W. Brunton, and S. L. Brunton, "Sindy with control: A tutorial," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 16–21, IEEE, 2021.

[124] D. E. Shea, S. L. Brunton, and J. N. Kutz, "Sindy-bvp: Sparse identification of nonlinear dynamics for boundary value problems," *Phys. rev. res.*, vol. 3, no. 2, p. 023255, 2021.

[125] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nat. Rev. Phys.*, vol. 3, pp. 422–440, 2021.

[126] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Comput. Methods Appl. Mech. Eng.*, vol. 365, p. 113028, 2020.

[127] A. D. J. Karniadakis and G. Em, "Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations.," *Commun. Comput. Phys.*, vol. 28, no. 5, pp. 2002–2041, 2020.

[128] K. Shukla, A. D. Jagtap, and G. E. Karniadakis, "Parallel physics-informed neural networks via domain decomposition," *J. Comput. Phys.*, vol. 447, p. 110683, 2021.

[129] T. Qin, Z. Chen, J. D. Jakeman, and D. Xiu, "Deep learning of parameterized equations with applications to uncertainty quantification," *Int. J. Uncertain. Quantif.*, vol. 11, no. 2, 2021.

[130] Y. Sun, L. Zhang, and H. Schaeffer, "NeuPDE: Neural network based ordinary and partial differential equations for modeling time-dependent data," in *Proceedings of The First Mathematical and Scientific Machine Learning Conference* (J. Lu and R. Ward, eds.), vol. 107 of *Proceedings of Machine Learning Research*, pp. 352–372, PMLR, 20–24 Jul 2020.

[131] H. Xu, H. Chang, and D. Zhang, "Dlga-pde: Discovery of pdes with incomplete candidate library via combination of deep learning and genetic algorithm," *J. Comput. Phys.*, vol. 418, p. 109584, 2020.

[132] P. Fannin, C. Marin, I. Malaescu, and N. Stefu, "An investigation of the microscopic and macroscopic properties of magnetic fluids," *Phys. B: Condens.*, vol. 388, no. 1-2, pp. 87–92, 2007.

[133] D. Givon, R. Kupferman, and A. Stuart, "Extracting macroscopic dynamics: model problems and algorithms," *Nonlinearity*, vol. 17, no. 6, p. R55, 2004.

[134] A. N. Gorban, N. K. Kazantzis, I. G. Kevrekidis, H. C. Öttinger, and C. Theodoropoulos, *Model reduction and coarse-graining approaches for multiscale phenomena*. Springer, 2006.

[135] M. G. Saunders and G. A. Voth, "Coarse-graining methods for computational biology," *Annu. Rev. Biophys.*, vol. 42, pp. 73–93, 2013.

[136] S. O. Nielsen, C. F. Lopez, G. Srinivas, and M. L. Klein, "Coarse grain models and the computer simulation of soft materials," *J. Condens. Matter Phys.*, vol. 16, no. 15, p. R481, 2004.

[137] J. Jin, A. J. Pak, A. E. Durumeric, T. D. Loose, and G. A. Voth, "Bottom-up coarse-graining: Principles and perspectives," *J. Chem. Theory Comput.*, vol. 18, no. 10, pp. 5759–5791, 2022.

[138] F. Müller-Plathe, "Coarse-graining in polymer simulation: from the atomistic to the mesoscopic scale and back," *ChemPhysChem*, vol. 3, no. 9, pp. 754–769, 2002.

[139] S. Dhamankar and M. A. Webb, "Chemically specific coarse-graining of polymers: methods and prospects," *J. Polym. Sci.*, vol. 59, no. 22, pp. 2613–2643, 2021.

[140] Y. Mori and M. Sakai, "Visualization study on the coarse graining dem for large-scale gas–solid flow systems," *Particuology*, vol. 59, pp. 24–33, 2021.

[141] N. Provatas and K. Elder, *Phase-Field Methods in Materials Science and Engineering*. Weiheim, Germany: John Wiley & Sons, Inc., 2010.

[142] R. L. Harrison, "Introduction to monte carlo simulation," in *AIP conference proceedings*, vol. 1204, pp. 17–21, American Institute of Physics, 2010.

[143] H. Wang, F.-B. Tian, and X.-D. Liu, "Lattice boltzmann model for interface capturing of multiphase flows based on allen–cahn equation," *Chin. Phys. B*, vol. 31, no. 2, p. 024701, 2022.

[144] R. E. Rudd and J. Q. Broughton, "Coarse-grained molecular dynamics and the atomic limit of finite elements," *Phys. Rev. B*, vol. 58, no. 10, p. R5893, 1998.

[145] H. H. Nax, M. N. Burton-Chellew, S. A. West, and H. P. Young, "Learning in a black box," *J. Econ. Behav. Organ.*, vol. 127, pp. 1–15, 2016.

[146] E. Elkind, B. Genest, D. Peled, and H. Qu, "Grey-box checking," in *Formal Techniques for Networked and Distributed Systems-FORTE 2006: 26th IFIP WG 6.1 International Conference, Paris, France, September 26-29, 2006. Proceedings 26*, pp. 420–435, Springer, 2006.

[147] O. Loyola-Gonzalez, "Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view," *IEEE access*, vol. 7, pp. 154096–154113, 2019.

[148] F. P. Kemeth, S. Alonso, B. Echebarria, T. Moldenhawer, C. Beta, and I. G. Kevrekidis, "Black and gray box learning of amplitude equations: Application to phase field systems," *Phys. Rev. E* ., vol. 107, no. 2, p. 025305, 2023.

[149] S. Yaghoubi and G. Fainekos, "Gray-box adversarial testing for control systems with machine learning components," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 179–184, 2019.

[150] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *Ieee Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[151] T. Barlow, "Feed-forward neural networks for secondary structure prediction," *J. Mol. Graph.*, vol. 13, pp. 175–183, 1995.

[152] S. I. Gallant *et al.*, "Perceptron-based learning algorithms," *IEEE Trans. Neural Netw.*, vol. 1, no. 2, pp. 179–191, 1990.

[153] A. G. Ivakhnenko and V. G. Lapa, "Cybernetic predicting devices," tech. rep., PURDUE UNIV LAFAYETTE IND SCHOOL OF ELECTRICAL ENGINEERING, New York, USA, 1966.

[154] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, pp. 65–93, Elsevier, 1992.

[155] B. J. Wythoff, "Backpropagation neural networks: a tutorial," *Chemom. Intell. Lab. Syst.*, vol. 18, no. 2, pp. 115–155, 1993.

[156] R. J. Erb, "Introduction to backpropagation neural network computation," *Pharm. Res.*, vol. 10, pp. 165–170, 1993.

[157] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001.

[158] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Phys. D: Nonlinear Phenom.*, vol. 404, p. 132306, 2020.

[159] G. Saon, Z. Tüske, D. Bolanos, and B. Kingsbury, "Advancing rnn transducer technology for speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5654–5658, IEEE, 2021.

[160] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain, "Machine translation using deep learning: An overview," in *International conference on computer, communications and electronics (comptelix)*, pp. 162–167, IEEE, 2017.

[161] H. A. Al-Muzaini, T. N. Al-Yahya, and H. Benhidour, "Automatic arabic image captioning using rnn-lstm-based language model and cnn," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 6, 2018.

[162] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, 1997.

[163] A. Graves, "Long short-term memory," in *Supervised Sequence Labelling with Recurrent Neural Networks*, pp. 37–45, Berlin, Heidelberg, Germany: Springer, 2012.

[164] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pp. 1597–1600, IEEE, 2017.

[165] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[166] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[167] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Comput.*, vol. 29, pp. 2352–2449, 2017.

[168] S. Miao, Z. J. Wang, and R. Liao, "A cnn regression approach for real-time 2d/3d registration," *IEEE Trans. Med.*, vol. 35, no. 5, pp. 1352–1363, 2016.

[169] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language*, (Doha, Qatar), pp. 1746—1751, Association for Computational Linguistics, 2014.

[170] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (Boston, MA , USA), pp. 1–9, 2015.

[171] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," *arXiv preprint arXiv:1703.04691*, 2017.

[172] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations," *ArXiv*, vol. abs/1711.10561, 2017.

[173] A. D. Jagtap, Z. Mao, N. Adams, and G. E. Karniadakis, "Physics-informed neural networks for inverse problems in supersonic flows," *J. Comput. Phys.*, vol. 466, p. 111402, 2022.

[174] S. Wang, X. Yu, and P. Perdikaris, "When and why PINNs fail to train: A neural tangent kernel perspective," *J. Comput. Phys.*, vol. 449, p. 110768, 2022.

[175] R. Rodriguez-Torrado, P. Ruiz, L. Cueto-Felgueroso, M. C. Green, T. Friesen, S. Matringe, and J. Togelius, "Physics-informed attention-based neural network for solving non-linear partial differential equations," *arXiv preprint arXiv:2105.07898*, 2021.

[176] W. Li, C. Zhang, C. Wang, H. Guan, and D. Tao, "Revisiting pinns: Generative adversarial physics-informed neural networks and point-weighting method," *arXiv preprint arXiv:2205.08754*, 2022.

[177] H. Gao, M. J. Zahr, and J.-X. Wang, "Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems," *Comput. Methods Appl. Mech. Eng.*, vol. 390, p. 114502, 2022.

[178] L. Yang, X. Meng, and G. E. Karniadakis, "B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data," *J. Comput. Phys.*, vol. 425, p. 109913, 2021.

[179] W. La Cava, P. Orzechowski, B. Burlacu, F. O. de França, M. Virgolin, Y. Jin, M. Kommenda, and J. H. Moore, "Contemporary symbolic regression methods and their relative performance," *arXiv preprint arXiv:2107.14351*, 2021.

[180] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *science*, vol. 324, no. 5923, pp. 81–85, 2009.

[181] M. F. Korns, *Accuracy in Symbolic Regression*, pp. 129–151. New York, USA: Springer, 2011.

[182] M. Kotanchek, G. Smits, and E. Vladislavleva, *Trustable symbolic regression models: using ensembles, interval arithmetic and pareto fronts to develop robust and trust-aware models*, pp. 201–220. Boston, MA: Springer US, 2008.

[183] L. Kammerer, G. Kronberger, B. Burlacu, S. M. Winkler, M. Kommenda, and M. Affenzeller, *Symbolic Regression by Exhaustive Search: Reducing the Search Space Using Syntactical Constraints and Efficient Semantic Structure Deduplication*, pp. 79–99. Cham: Springer International Publishing, 2020.

[184] B. Can and C. Heavey, "Comparison of experimental designs for simulation-based symbolic regression of manufacturing systems," *Comput. Ind. Eng.*, vol. 61, no. 3, pp. 447–462, 2011.

[185] V. Aryadoust, "Application of evolutionary algorithm-based symbolic regression to language assessment: Toward nonlinear modeling," *Psychological Test and Assessment Modeling*, vol. 57, no. 3, p. 301, 2015.

[186] P. D. Truscott and M. F. Korns, *Detecting Shadow Economy Sizes with Symbolic Regression*, pp. 195–210. New York, USA: Springer, 2011.

[187] H. Vaddireddy, A. Rasheed, A. E. Staples, and O. San, "Feature engineering and symbolic regression methods for detecting hidden physics from sparse sensor observation data," *Phys. Fluids*, vol. 32, p. 015113, 2020.

[188] B. Babu and S. Karthik, "Genetic programming for symbolic regression of chemical process systems.," *Eng. Lett.*, vol. 14, no. 2, pp. 42–55, 2007.

[189] W. B. Langdon, *Genetic Programming — Computers Using "Natural Selection" to Generate Programs*, pp. 9–42. Boston, MA: Springer, 1998.

[190] H. Tuan-Hao, R. I. McKay, D. Essam, and N. X. Hoai, "Solving symbolic regression problems using incremental evaluation in genetic programming," in *2006 IEEE International Conference on Evolutionary Computation*, pp. 2134–2141, IEEE, 2006.

[191] J. J. Schnur and N. V. Chawla, "Information fusion via symbolic regression: A tutorial in the context of human health," *Inf. Fusion.*, 2022.

[192] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong, "Crossover and mutation operators of genetic algorithms," *Int. J. Mach. Learn. Comput.*, vol. 7, no. 1, pp. 9–12, 2017.

[193] S.-M. Udrescu and M. Tegmark, "Ai feynman: A physics-inspired method for symbolic regression," *Sci. Adv.*, vol. 6, no. 16, p. eaay2631, 2020.

[194] S. M. Allen and J. W. Cahn, "Ground state structures in ordered binary alloys with second neighbor interactions," *Acta Metall.*, vol. 20, pp. 423–433, 1972.

[195] J. W. Cahn and J. E. Hilliard, "Free energy of a nonuniform system. I. Interfacial free energy," *J. Chem. Phys.*, vol. 28, pp. 258–267, 1958.

[196] P. C. Hohenberg and B. I. Halperin, "Theory of dynamic critical phenomena," *Rev. Mod. Phys.*, vol. 49, p. 435, 1977.

[197] K. R. Elder and M. Grant, "Modeling elastic and plastic deformations in nonequilibrium processing using phase field crystals," *Phys. Rev. E*, vol. 70, p. 051605, 2004.

[198] D.-H. Yeon, Z.-F. Huang, K. Elder, and K. Thornton, "Density-amplitude formulation of the phase-field crystal model for two-phase coexistence in two and three dimensions," *Philos. Mag.*, vol. 90, no. 1-4, pp. 237–263, 2010.

[199] H. Zhang, X. Jiang, F. Zeng, and G. E. Karniadakis, "A stabilized semi-implicit fourier spectral method for nonlinear space-fractional reaction-diffusion equations," *J. Comput. Phys.*, vol. 405, p. 109141, 2020.

[200] T. Young, "An essay on the cohesion of fluids," *Philos. Trans. R. Soc. Lond.*, vol. 95, no. 0, pp. 65–87, 1805.

[201] P.-S. d. Laplace, "Supplément au livre X du traité de mécanique céleste. sur l'action capillaire," in *Traité de mécanique céleste*, Paris, France: Gauthier-Vilars, 1805.

[202] N. Adam, "Use of the term 'young's equation'for contact angles," *Nature*, vol. 180, no. 4590, pp. 809–810, 1957.

[203] L. H. Tanner, "The spreading of silicone oil drops on horizontal surfaces," *J. Phys. D: Appl. Phys.*, vol. 12, no. 9, p. 1473, 1979.

[204] D. Bonn, J. Eggers, J. Indekeu, J. Meunier, and E. Rolley, "Wetting and spreading," *Rev. Mod. Phys.*, vol. 81, no. 2, pp. 739–805, 2009.

[205] G. McHale, N. Shirtcliffe, S. Aqil, C. Perry, and M. Newton, "Topography driven spreading," *Phys. Rev. Lett.*, vol. 93, no. 3, p. 036102, 2004.

[206] S. L. Cormier, J. D. McGraw, T. Salez, E. Raphaël, and K. Dalnoki-Veress, "Beyond tanner's law: Crossover between spreading regimes of a viscous droplet on an identical film," *Phys. Rev. Lett.*, vol. 109, no. 15, p. 154501, 2012.

[207] K. G. Winkels, J. H. Weijs, A. Eddi, and J. H. Snoeijer, "Initial spreading of low-viscosity drops on partially wetting surfaces," *Phys. Rev. E*, vol. 85, p. 055301, 2012.

[208] Q. Rao, Y. Xia, J. Li, J. McConnell, J. Sutherland, and Z. Li, "A modified many-body dissipative particle dynamics model for mesoscopic fluid simulation: methodology, calibration, and application for hydrocarbon and water," *Mol. Sim.*, vol. 47, no. 4, pp. 363–375, 2021.

[209] Y. Xia, J. Goral, H. Huang, I. Miskovic, P. Meakin, and M. Deo, "Many-body dissipative particle dynamics modeling of fluid flow in fine-grained nanoporous shales," *Phys. Fluids*, vol. 29, no. 5, p. 056601, 2017.

[210] Z. Li, G.-H. Hu, Z.-L. Wang, Y.-B. Ma, and Z.-W. Zhou, "Three dimensional flow structures in a moving droplet on substrate: A dissipative particle dynamics study," *Phys. Fluids*, vol. 25, p. 072103, 2013.

[211] P. Español and P. Warren, "Statistical mechanics of dissipative particle dynamics," *EPL*, vol. 30, no. 4, p. 191, 1995.

[212] R. D. Groot, "Applications of dissipative particle dynamics," in *Novel Methods in Soft Matter Simulations* (M. Karttunen, A. Lukkarinen, and I. Vattulainen, eds.), pp. 5–38, Berlin, Heidelberg, Germany: Springer, 2004.

[213] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis," *Astron. J.*, vol. 82, p. 1013, 1977.

[214] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, *et al.*, "Lammps-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comput. Phys. Comms.*, vol. 271, p. 108171, 2022.

[215] C. Huré, H. Pham, and X. Warin, "Some machine learning schemes for high-dimensional nonlinear PDEs," *Math. Comput.*, vol. 89, pp. 1547–1579, 2020.

[216] W. E, J. Han, and A. Jentzen, "Algorithms for solving high dimensional PDEs: From nonlinear Monte Carlo to machine learning," *Nonlinearity*, vol. 35, pp. 278–310, 2021.

[217] R. Ranade, C. Hill, and J. Pathak, "DiscretizationNet: A machine-learning based solver for Navier–Stokes equations using finite volume discretization," *Comput. Methods Appl. Mech. Eng.*, vol. 378, p. 113722, 2021.

[218] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nat. Mach. Intell.*, vol. 3, pp. 218–229, 2021.

[219] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "DeepXDE: A deep learning library for solving differential equations," *SIAM Rev.*, vol. 63, pp. 208–228, 2021.

[220] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos, "Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks," *Proc. R. Soc. Math. Phys. Eng. Sci.*, vol. 474, p. 20170844, 2018.

[221] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying LSTM to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*, pp. 193–200, Vienna, Austria: Springer, 2002.

[222] J. del Águila Ferrandis, M. S. Triantafyllou, C. Chryssostomidis, and G. E. Karniadakis, "Learning functionals via LSTM neural networks for predicting vessel dynamics in extreme sea states," *Proc. R. Soc. A*, vol. 477, p. 20190897, 2021.

[223] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," *Proc. Natl. Acad. Sci.*, vol. 116, pp. 15344–15349, 2019.

[224] C. Theodoropoulos and E. Luna-Ortiz, "A reduced input/output dynamic optimisation method for macroscopic and microscopic systems," in *Model reduction and coarse-graining approaches for multiscale phenomena*, pp. 535–560, Berlin, Heidelberg, Germany: Springer, 2006.

[225] T. N. Thiem, M. Kooshkbaghi, T. Bertalan, C. R. Laing, and I. G. Kevrekidis, "Emergent spaces for coupled oscillators," *Front. Comput. Neurosci.*, vol. 14, p. 36, 2020.

[226] D. Qin, J. Yu, G. Zou, R. Yong, Q. Zhao, and B. Zhang, "A novel combined prediction scheme based on CNN and LSTM for urban PM 2.5 concentration," *IEEE Access*, vol. 7, pp. 20050–20059, 2019.

[227] A. M. Turing, "The chemical basis of morphogenesis," *Bull. Math. Biol.*, vol. 52, pp. 153–197, 1990.

[228] T. Leppänen, M. Karttunen, K. Kaski, R. A. Barrio, and L. Zhang, "A new dimension to Turing patterns," *Phys. D: Nonlinear Phenom.*, vol. 168–169, pp. 35–44, 2002.

[229] P. Gray and S. Scott, "Sustained oscillations and other exotic patterns of behavior in isothermal reactions," *J. Phys. Chem.*, vol. 89, pp. 22–32, 1985.

[230] B. Grossmann, K. R. Elder, M. Grant, and J. M. Kosterlitz, "Directional solidification in two and three dimensions," *Phys. Rev. Lett.*, vol. 71, pp. 3323–3326, 1993.

[231] W. J. Boettinger, J. A. Warren, C. Beckermann, and A. Karma, "Phase-field simulation of solidification," *Annu. Rev. Mater. Sci.*, vol. 32, pp. 163–194, 2002.

[232] B. Nestler, H. Garcke, and B. Stinner, "Multicomponent alloy solidification: Phase-field modeling and simulations," *Phys. Rev. E*, vol. 71, p. 041609, 2005.

[233] V. Heinonen, C. Achim, J. Kosterlitz, S.-C. Ying, J. Lowengrub, and T. Ala-Nissila, "Consistent hydrodynamics for phase field crystals," *Phys. Rev. Lett.*, vol. 116, p. 024303, 2016.

[234] E. Alster, K. Elder, and P. W. Voorhees, "Displacive phase-field crystal model," *Phys. Rev. Mater.*, vol. 4, p. 013802, 2020.

[235] N. Provatas, J. Dantzig, B. Athreya, P. Chan, P. Stefanovic, N. Goldenfeld, and K. Elder, "Using the phase-field crystal method in the multi-scale modeling of microstructure evolution," *JOM*, vol. 59, pp. 83–90, 2007.

[236] N. Faghihi, S. Mkhonta, K. Elder, and M. Grant, "Phase-field crystal for an antiferromagnet with elastic interactions," *Phys. Rev. E*, vol. 100, p. 022128, 2019.

[237] I. Aranson, V. Kalatsky, and V. Vinokur, "Continuum field description of crack propagation," *Phys. Rev. Lett.*, vol. 85, p. 118, 2000.

[238] R. Spatschek, E. Brener, and A. Karma, "Phase field modeling of crack propagation," *Philos. Mag.*, vol. 91, pp. 75–95, 2011.

[239] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units," *ArXiv*, vol. 1611.01491, pp. 1–17, 2016.

[240] W. L. Ziegler, "Computational method to compute the derivative and antiderivative; with concern for terminating a converging iterative process and considering accuracy, round-off error, approximation, and extrapolation," *Math. Model.*, vol. 8, pp. 77–84, 1987.

[241] W. H. Press and S. A. Teukolsky, "Numerical calculation of derivatives," *Comput. Phys.*, vol. 5, pp. 68–69, 1991.

[242] H. Arbabi and I. G. Kevrekidis, "Particles to partial differential equations parsimoniously," *Chaos Interdiscip. J. Nonlinear Sci.*, vol. 31, p. 033137, 2021.

[243] H. Arbabi, J. E. Bunder, G. Samaey, A. J. Roberts, and I. G. Kevrekidis, "Linking machine learning with multiscale numerics: Data-driven discovery of homogenized equations," *JOM*, vol. 72, pp. 4444–4457, 2020.

[244] J.-F. Cai, B. Dong, S. Osher, and Z. Shen, "Image restoration: Total variation, wavelet frames, and beyond," *J. Am. Math. Soc.*, vol. 25, pp. 1033–1089, 2012.

[245] B. Dong, Q. Jiang, and Z. Shen, "Image restoration: Wavelet frame shrinkage, nonlinear evolution pdes, and beyond," *Multiscale Model. Simul.*, vol. 15, pp. 606–660, 2017.

[246] V. Oommen, K. Shukla, S. Goswami, R. Dingreville, and G. E. Karniadakis, "Learning two-phase microstructure evolution using neural operators and autoencoder architectures," *arXiv preprint arXiv:2204.07230*, 2022.

[247] L. Prechelt, "Early stopping-but when?," in *Neural Networks: Tricks of the Trade*, pp. 55–69, Berlin, Heidelberg, Germany: Springer, 1998.

[248] C. J. Lapeyre, A. Misdariis, N. Cazard, D. Veynante, and T. Poinsot, "Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates," *Combust. Flame*, vol. 203, pp. 255–264, 2019.

[249] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa,

P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental algorithms for scientific computing in python," *Nat. Methods*, vol. 17, pp. 261–272, 2020.

[250] A. C. Hindmarsh, "ODEPACK, a systematized collection of ODE solvers," *Sci. Comput.*, vol. 1, pp. 55–64, 1983.

[251] R. Rico-Martinez, J. Anderson, and I. Kevrekidis, "Continuous-time nonlinear signal processing: a neural network based approach for gray box identification," in *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, (Ermioni, Greece), pp. 596–605, IEEE, 1994.

[252] J. Fan and Q. Yao, *Nonlinear time series: nonparametric and parametric methods*. New York, USA: Springer, 2003.

[253] E. Kiyani, S. Silber, M. Kooshkbaghi, and M. Karttunen, "Machine-learning-based data-driven discovery of nonlinear phase-field dynamics," *Phys. Rev. E* ., vol. 106, p. 065303, 2022.

[254] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proc. R. Soc.*, vol. 474, no. 2219, p. 20180335, 2018.

[255] K. Fukami, T. Murata, K. Zhang, and K. Fukagata, "Sparse identification of nonlinear dynamics with low-dimensionalized flow representations," *J. Fluid Mech.*, vol. 926, p. A10, 2021.

[256] M. Hoffmann, C. Fröhner, and F. Noé, "Reactive sindy: Discovering governing reactions from concentration data," *J. Chem. Phys.*, vol. 150, no. 2, p. 025101, 2019.

[257] Q. Lou, X. Meng, and G. E. Karniadakis, "Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation," *J. Comp. Phys.*, vol. 447, p. 110676, 2021.

[258] K. Shukla, P. C. Di Leoni, J. Blackshire, D. Sparkman, and G. E. Karniadakis, "Physics-Informed neural network for ultrasound nondestructive quantification of surface breaking cracks," *J. Nondestr. Eval.*, vol. 39, no. 3, p. 61, 2020.

[259] A. D. J. Karniadakis and G. Em, "Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations," *Commun. Comput. Phys.*, vol. 28, no. 5, pp. 2002–2041, 2020.

[260] K. Shukla, A. D. Jagtap, and G. E. Karniadakis, "Parallel physics-informed neural networks via domain decomposition," *J. Comput. Phys.*, vol. 447, p. 110683, 2021.

[261] L. Billard and E. Diday, "Symbolic regression analysis," in *Classification, Clustering, and Data Analysis* (K. Jajuga, A. Sokołowski, and H.-H. Bock, eds.), (Berlin, Heidelberg, Germany), pp. 281–288, Springer, 2002.

[262] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[263] P. C. Hohenberg and B. I. Halperin, "Theory of dynamic critical phenomena," *Rev. Mod. Phys.*, vol. 49, no. 3, pp. 435–479, 1977.

[264] P. Colli, C. Verdi, and A. Visintin, *Free boundary problems: theory and applications*, vol. 147. Birkhäuser, 2012.

[265] J. E. Pearson, "Complex patterns in a simple system," *Science*, vol. 261, no. 5118, pp. 189–192, 1993.

[266] T. Leppänen, M. Karttunen, R. A. Barrio, and K. Kaski, "Morphological transitions and bistability in turing systems," *Phys. Rev. E*, vol. 70, no. 6 Pt 2, p. 066202, 2004.

[267] N. Faghihi, N. Provatas, K. R. Elder, M. Grant, and M. Karttunen, "Phase-field-crystal model for magnetocrystalline interactions in isotropic ferromagnetic solids," *Phys. Rev. E*, vol. 88, p. 032407, 2013.

[268] S. Najem and M. Grant, "Phase-field model for collective cell migration," *Phys. Rev. E*, vol. 93, no. 5, p. 052405, 2016.

[269] Z. Hong and V. Viswanathan, "Open-Sourcing Phase-Field simulations for accelerating energy materials design and optimization," *ACS Energy Lett.*, vol. 5, no. 10, pp. 3254–3259, 2020.

[270] S. A. Silber and M. Karttunen, "SymPhas —general purpose software for phase-field, phase-field crystal, and reaction-diffusion simulations," *Adv. Theory Simul.*, vol. 5, no. 1, p. 2100351, 2022.

[271] L.-Q. Chen, "Phase-Field models for microstructure evolution," *Annu. Rev. Mater. Res.*, vol. 32, no. 1, pp. 113–140, 2002.

[272] Y. Li, R. Shi, C. Wang, X. Liu, and Y. Wang, "Phase-field simulation of thermally induced spinodal decomposition in polymer blends," *Model. Simul. Mater. Sci. Eng.*, vol. 20, p. 075002, 2012.

[273] A. A. Nepomnyashchy, "Coarsening versus pattern formation," *Comptes Rendus Physique*, vol. 16, no. 3, pp. 267–279, 2015.

[274] R. Mattey and S. Ghosh, "A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations," *Comput. Methods Appl. Mech. Eng.*, vol. 390, p. 114474, 2022.

[275] C. L. Wight and J. Zhao, "Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks," *Commun. Comput. Phys.*, vol. 29, pp. 930–954, 2021.

[276] Y. Shin, J. Darbon, and G. E. Karniadakis, "On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs," *Commun. Comput. Phys.*, vol. 28, no. 5, pp. 2042–2074, 2020.

[277] W. Li, M. Z. Bazant, and J. Zhu, "Phase-Field DeepONet: Physics-informed deep operator neural network for fast simulations of pattern formation governed by gradient flows of free-energy functionals," *arXiv preprint*, 2023.

[278] S. Cai, Z. Wang, S. Wang, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks for heat transfer problems," *J. Heat Transf.*, vol. 143, 2021.

[279] J. Stiasny, G. S. Misyris, and S. Chatzivasileiadis, "Physics-informed neural networks for non-linear system identification for power system dynamics," in *2021 IEEE Madrid PowerTech*, pp. 1–6, IEEE, 2021.

[280] N. Fatima, "Enhancing performance of a deep neural network: A comparative analysis of optimization algorithms," *ADCAIJ: Adv. Distrib. Comput. Artif. Intell. J.*, vol. 9, no. 2, pp. 79–90, 2020.

[281] A. Mustapha, L. Mohamed, and K. Ali, "Comparative study of optimization techniques in deep learning: Application in the ophthalmology field," *J. Phys.: Conf. Ser.*, vol. 1743, no. 1, p. 012002, 2021.

[282] Y. Ma, X. Xu, S. Yan, and Z. Ren, "A preliminary study on the resolution of electro-thermal multi-physics coupling problem using physics-informed neural network (pinn)," *Algorithms*, vol. 15, p. 53, 2022.

[283] Q. Zhu, Z. Liu, and J. Yan, "Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks," *Comput. Mech.*, vol. 67, pp. 619–635, 2021.

[284] S. Choi, I. Jung, H. Kim, J. Na, and J. M. Lee, "Physics-informed deep learning for data-driven solutions of computational fluid dynamics," *Korean J. Chem. Eng.*, vol. 39, pp. 515–528, 2022.

[285] D. My Ha, C. Pao-Hsiung, W. Jian Cheng, and O. Chin Chun, "Physics-informed neural network with numerical differentiation for modelling complex fluid dynamic problems," in *International Conference on Offshore Mechanics and Arctic Engineering*, vol. 85925, p. V007T08A001, American Society of Mechanical Engineers, 2022.

[286] C. Broeckhoven and A. du Plessis, "Has snake fang evolution lost its bite? new insights from a structural mechanics viewpoint," *Biol. Lett.*, vol. 13, no. 8, p. 20170293, 2017.

[287] J.-H. Bastek and D. M. Kochmann, "Physics-informed neural networks for shell structures," *Eur. J. Mech. A/Solids*, vol. 97, p. 104849, 2023.

[288] R. Laubscher, "Simulation of multi-species flow and heat transfer using physics-informed neural networks," *Phys. Fluids*, vol. 33, p. 087101, 2021.

[289] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Commun. ACM.*, vol. 64, no. 3, pp. 107–115, 2021.

[290] B. Neyshabur, S. Bhojanapalli, D. Mcallester, and N. Srebro, "Exploring generalization in deep learning," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[291] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge: MIT press, 2016.

[292] G. Swirszcz, W. M. Czarnecki, and R. Pascanu, "Local minima in training of neural networks," *arXiv preprint arXiv:1611.06310*, 2016.

[293] N. Landro, I. Gallo, and R. La Grassa, "Mixing adam and sgd: a combined optimization method," *arXiv preprint arXiv:2011.08042*, 2020.

[294] L. Billard and E. Diday, "Symbolic regression analysis," in *Classification, Clustering, and Data Analysis* (K. Jajuga, A. Sokołowski, and H.-H. Bock, eds.), (Berlin, Heidelberg, Germany), pp. 281–288, Springer, 2002.

[295] Z. Hu, A. D. Jagtap, G. E. Karniadakis, and K. Kawaguchi, "When do extended physics-informed neural networks (xpinns) improve generalization?," *arXiv preprint arXiv:2109.09444*, 2021.

[296] T. De Ryck, A. D. Jagtap, and S. Mishra, "Error estimates for physics informed neural networks approximating the navier-stokes equations," *arXiv preprint arXiv:2203.09346*, 2022.

[297] G. Farhani, A. Kazachek, and B. Wang, "Momentum diminishes the effect of spectral bias in physics-informed neural networks," *arXiv preprint arXiv:2206.14862*, 2022.

[298] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, "Characterizing possible failure modes in physics-informed neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 34, pp. 26548–26560, 2021.

[299] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *International Conference on Machine Learning*, pp. 5301–5310, PMLR, 2019.

[300] L. N. Trefethen and D. Bau, *Numerical linear algebra*, vol. 181. Siam, Philadelphia, 2022.

[301] O. Claveria, E. Monte, and S. Torra, "Assessment of the effect of the financial crisis on agents' expectations through symbolic regression," *Appl. Econ. Lett.*, vol. 24, pp. 648–652, 2017.

[302] J. Fitzsimmons and P. Moscato, "Symbolic regression modeling of drug responses," in *First International Conference on Artificial Intelligence for Industries (AI4I)*, pp. 52–59, IEEE, 2018.

[303] T. Stephens, "Genetic programming in python, with a scikit-learn inspired api: gplearn," 2016.

[304] M. Cranmer, "Interpretable machine learning for science with pysr and symbolicregression. jl," *arXiv preprint arXiv:2305.01582*, 2023.

[305] R. Feynman, R. Leighton, and M. Sands, *The Feynman Lectures on Physics, Vol. I: The New Millennium Edition: Mainly Mechanics, Radiation, and Heat*. Basic Books, 2015.

[306] R. Feynman, R. Leighton, and M. Sands, *The Feynman Lectures on Physics*. No. v. 2 in The Feynman Lectures on Physics, Pearson/Addison-Wesley, 2006.

[307] R. Feynman, *Lectures on Physics: The Definitive Edition*. Pearson/ Addison Wesley, 2006.

[308] J. Weiss, "A tutorial on the proper orthogonal decomposition," in *AIAA Aviation 2019 Forum*, (Reston, Virginia), American Institute of Aeronautics and Astronautics, 2019.

[309] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. New York, USA: Springer, 2009.

[310] M. E. Wall, A. Rechtsteiner, and L. M. Rocha, "Singular value decomposition and principal component analysis," in *A Practical Approach to Microarray Data Analysis* (D. P.

Berrar, W. Dubitzky, and M. Granzow, eds.), pp. 91–109, Boston: Kluwer Academic Publishers, 2005.

[311] J. G. Proakis, *Digital signal processing: principles, algorithms, and applications, 4/E*. Pearson Education, India, 2007.

[312] Y. Zhang, D. Zhao, J. Zhang, R. Xiong, and W. Gao, "Interpolation-dependent image downsampling," *IEEE Trans. Image Process.*, vol. 20, no. 11, pp. 3291–3296, 2011.

[313] C. A. De Moura and C. S. Kubrusly, "The courant–friedrichs–lewy (cfl) condition," *AMC*, vol. 10, no. 12, 2013.

[314] T. S. Edwards, "Effects of aliasing on numerical integration," *Mech. Syst. Signal Process.*, vol. 21, no. 1, pp. 165–176, 2007.

[315] J. Ren and J. Duan, "Identifying stochastic governing equations from data of the most probable transition trajectories," *arXiv preprint arXiv:2002.10251*, 2020.

[316] C. B. Delahunt and J. N. Kutz, "A toolkit for data-driven discovery of governing equations in high-noise regimes," *IEEE Access*, vol. 10, pp. 31210–31234, 2022.

[317] K. Meidani and A. B. Farimani, "Data-driven identification of 2d partial differential equations using extracted physical features," *Comput. Methods Appl. Mech. Eng.*, vol. 381, p. 113831, 2021.

[318] P. G. de Gennes, "Wetting: statics and dynamics," *Rev. Mod. Phys.*, vol. 57, no. 3, pp. 827–863, 1985.

[319] S. Nishimoto and B. Bhushan, "Bioinspired self-cleaning surfaces with superhydrophobicity, superoleophobicity, and superhydrophilicity," *RSC Adv.*, vol. 3, no. 3, pp. 671–690, 2013.

[320] N. Jain, A. Lemoine, G. Chaussonnet, A. Flatau, L. Bravo, A. Ghoshal, M. Walock, and M. Murugan, "A critical review of physical models in high temperature multiphase fluid dynamics: Turbulent transport and particle-wall interactions," *Appl. Mech. Rev.*, 2021.

[321] E. Dussan, "On the spreading of liquids on solid surfaces: static and dynamic contact lines," *Ann. Rev. Fluid Mech.*, vol. 11, no. 1, pp. 371–400, 1979.

[322] J. D. McGraw, T. S. Chan, S. Maurer, T. Salez, M. Benzaquen, E. Raphaël, M. Brinkmann, and K. Jacobs, "Slip-mediated dewetting of polymer microdroplets," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 113, no. 5, pp. 1168–1173, 2016.

[323] A. M. J. Edwards, R. Ledesma-Aguilar, M. I. Newton, C. V. Brown, and G. McHale, "A viscous switch for liquid-liquid dewetting," *Commun. Phys.*, vol. 3, no. 1, pp. 1–6, 2020.

[324] D. L. Poerschke and C. G. Levi, "Effects of cation substitution and temperature on the interaction between thermal barrier oxides and molten CMAS," *J. Eur. Ceram. Soc.*, vol. 35, no. 2, pp. 681–691, 2015.

[325] V. L. Wiesner, U. K. Vempati, and N. P. Bansal, "High temperature viscosity of calcium-magnesium-aluminosilicate glass from synthetic sand," *Scr. Mater.*, vol. 124, pp. 189–192, 2016.

[326] D. R. Clarke, M. Oechsner, and N. P. Padture, "Thermal-barrier coatings for more efficient gas-turbine engines," *MRS Bull.*, vol. 37, no. 10, pp. 891–898, 2012.

[327] N. L. Ndamka, R. G. Wellman, and J. R. Nicholls, "The degradation of thermal barrier coatings by molten deposits: introducing the concept of basicity," *Mater. High Temp.*, vol. 33, no. 1, pp. 44–50, 2016.

[328] A. Nieto, R. Agrawal, L. Bravo, C. Hofmeister-Mock, M. Pepi, and A. Ghoshal, "Calcia–magnesia–alumina–silicate (CMAS) attack mechanisms and roadmap towards

sandphobic thermal and environmental barrier coatings," *Int. Mat. Rev.*, vol. 66, no. 7, pp. 451–492, 2021.

[329] K. M. Grant, S. Krämer, J. P. Löfvander, and C. G. Levi, "Cmas degradation of environmental barrier coatings," *Surface and Coatings Technology*, vol. 202, no. 4-7, pp. 653–657, 2007.

[330] M. H. Vidal-Setif, N. Chellah, C. Rio, C. Sanchez, and O. Lavigne, "Calcium–magnesium–alumino-silicate (CMAS) degradation of EB-PVD thermal barrier coatings: Characterization of CMAS damage on ex-service high pressure blade TBCs," *Surf. Coat. Technol.*, vol. 208, pp. 39–45, 2012.

[331] W. Song, Y. Lavallée, K.-U. Hess, U. Kueppers, C. Cimarelli, and D. B. Dingwell, "Volcanic ash melting under conditions relevant to ash turbine interactions," *Nat. Commun.*, vol. 7, p. 10795, 2016.

[332] J. Zhao, S. Chen, K. Zhang, and Y. Liu, "A review of many-body dissipative particle dynamics (mdpd): Theoretical models and its applications," *Phys. Fluids*, vol. 33, no. 11, p. 112002, 2021.

[333] L. Lei, E. L. Bertevas, B. C. Khoo, and N. Phan-Thien, "Many-body dissipative particle dynamics (MDPD) simulation of a pseudoplastic yield-stress fluid with surface tension in some flow processes," *J. Non-Newtonian Fluid Mech.*, vol. 260, pp. 163–174, 2018.

[334] A. Ghoufi and P. Malfreyt, "Coarse grained simulations of the electrolytes at the water–air interface from many body dissipative particle dynamics," *J. Chem. Theory Comput.*, vol. 8, no. 3, pp. 787–791, 2012.

[335] Z. Li, X. Bian, X. Yang, and G. E. Karniadakis, "A comparative study of coarse-graining methods for polymeric fluids: Mori-Zwanzig vs. iterative Boltzmann inversion vs. stochastic parametric optimization," *J. Chem. Phys.*, vol. 145, no. 4, p. 044102, 2016.

[336] K. C. Chan, Z. Li, and W. Wenzel, "A Mori-Zwanzig dissipative particle dynamics approach for anisotropic coarse grained molecular dynamics," *J. Chem. Theory Comput.*, vol. 19, no. 3, pp. 910–923, 2023.

[337] P. Español and P. B. Warren, "Perspective: Dissipative particle dynamics," *J. Chem. Phys.*, vol. 146, no. 15, p. 150901, 2017.

[338] P. B. Warren, "Hydrodynamic bubble coarsening in Off-Critical Vapor-Liquid phase separation," *Phys. Rev. Lett.*, vol. 87, no. 22, p. 225702, 2001.

[339] P. B. Warren, "Vapor-liquid coexistence in many-body dissipative particle dynamics," *Phys. Rev. E*, vol. 68, no. 6 Pt 2, p. 066702, 2003.

[340] R. Naraparaju, J. J. Gomez Chavez, P. Niemeyer, K.-U. Hess, W. Song, D. B. Dingwell, S. Lokachari, C. V. Ramana, and U. Schulz, "Estimation of CMAS infiltration depth in EB-PVD TBCs: A new constraint model supported with experimental approach," *J. Eur. Ceram. Soc.*, vol. 39, no. 9, pp. 2936–2945, 2019.

[341] N. P. Bansal and S. R. Choi, "Properties of desert sand and CMAS glass," Tech. Rep. NASA/TM-2014-218365, NASA Glenn Research Center Cleveland, Ohio, Aug. 2014.

[342] Z. Li, X. Bian, Y.-H. Tang, and G. E. Karniadakis, "A dissipative particle dynamics method for arbitrarily complex geometries," *J. Comput. Phys.*, vol. 355, pp. 534–547, 2018.

[343] O. Pitois and B. François, "Crystallization of condensation droplets on a liquid surface," *Coll. Polym. Sci.*, vol. 277, pp. 574–578, 1999.

[344] L. Chen, E. Bonaccurso, P. Deng, and H. Zhang, "Droplet impact on soft viscoelastic surfaces," *Phys. Rev. E*, vol. 94, p. 063117, 2016.

[345] G. Hassan, B. S. Yilbas, A. Al-Sharafi, and H. Al-Qahtani, "Self-cleaning of a hydrophobic surface by a rolling water droplet," *Sci. Rep.*, vol. 9, pp. 1–14, 2019.

[346] A. Eddi, K. G. Winkels, and J. H. Snoeijer, "Short time dynamics of viscous drop spreading," *Phys. Fluids*, vol. 25, no. 1, p. 013102, 2013.

[347] K. Shukla, P. C. Di Leoni, J. Blackshire, D. Sparkman, and G. E. Karniadakis, "Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks," *J. Nondestruct. Eval.*, vol. 39, pp. 1–20, 2020.

[348] S. Mishra and R. Molinaro, "Estimates on the generalization error of physics informed neural networks (pinns) for approximating pdes ii: A class of inverse problems," *arXiv preprint arXiv:2007.01138*, vol. 640, pp. 1–35, 2020.

[349] Y. Chen, L. Lu, G. E. Karniadakis, and L. Dal Negro, "Physics-informed neural networks for inverse problems in nano-optics and metamaterials," *Opt. Express*, vol. 28, pp. 11618–11633, 2020.

[350] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, "Physics-informed neural networks for high-speed flows," *Comput. Methods Appl. Mech. Eng.*, vol. 360, p. 112789, 2020.

[351] K. Bykov, M. M.-C. Höhne, A. Creosteanu, K.-R. Müller, F. Klauschen, S. Nakajima, and M. Kloft, "Explaining bayesian neural networks," *arXiv preprint arXiv:2108.10346*, 2021.

[352] C. M. Bishop, "Bayesian neural networks," *Journal of the Brazilian Computer Society*, vol. 4, pp. 61–68, 1997.

[353] T. Radivojević and E. Akhmatskaya, "Modified Hamiltonian Monte Carlo for bayesian inference," *Stat. Comput.*, vol. 30, no. 2, pp. 377–404, 2020.

[354] S. Brooks, "Markov Chain Monte Carlo method and its application," *J. Roy. Stat. Soc. D*, vol. 47, pp. 69–100, 1998.

[355] R. M. Neal, "MCMC using Hamiltonian dynamics," in *Handbook of Markov Chain Monte Carlo* (S. Brooks, A. Gelman, G. Jones, and X.-L. Meng, eds.), pp. 113–162, New York, USA: Chapman and Hall/CRC, 2011.

[356] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer New York, USA, 2012.

[357] A. Graves, "Practical variational inference for neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 24, 2011.

[358] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *Am. Stat. Assoc. Bull.*, vol. 112, no. 518, pp. 859–877, 2017.

Curriculum Vitae

Publications:

1. E. Kiyani, K. Shukla, G. Em Karniadakis, M. Karttunen, A Framework Based on Symbolic Regression Coupled with eXtended Physics-Informed Neural Networks for Gray-Box Learning of Equations of Motion from Data, Computer Methods in Applied Mechanics and Engineering, 415, p.116258, 2023.

2. E. Kiyani, H. Yazdani Sarvestani, H. Ravanbakhsh, R. Behbahani, B. Ashrafi, M. Rahmat, M. Karttunen, Designing architectured ceramics for transient thermal applications using finite element and deep learning, accepted for publication in Modelling and Simulation in Materials Science and Engineering (2023).

3. E. Kiyani, S. Silber, M. Kooshkbaghi, M. Karttunen, Data-driven discovery of nonlinear phase field dynamics, Physical Review E, 106, p.065303, 2022.

4. H. Ravanbakhsh, R. Behbahani, H. Yazdani Sarvestani, E. Kiyani, B. Ashrafi, M. Karttunen, and M. Rahmat, Architectured interlocked ceramics under thermal shock: finite elementand machine learning methods, Advanced Engineering Materials, 25, p.2201408, 2023. Cover article.

5. R. Behbahani, H. Yazdani Sarvestani, E. Fatehi, E. Kiyani, B. Ashrafi, M. Karttunen, and M. Rahmat, Machine learning-driven programming of alumina ceramics laser machining, Physica Scripta, 98, p.015834, 2022.

6. E. Kiyani, S.M. Vaezpour, and J. Tavakoli, Nonconvex vector optimization and optimality conditions for proper efficiency, International Journal of Analysis and Applications, 20, p.11-11, 2022.

7. E. Kiyani, S. M. Vaezpour, and J. Tavakoli, Optimality conditions for approximate solutions of real linear spaces, TWMS Journal of Applied and Engineering Mathematics, 407, p.395, 2021.

8. E. Kiyani, M. Soleimani-damaneh, Separation theorems and approximate proper efficiency on real linear vector spaces, Pacific Journal of Optimization, 10, p.715-734, 2014.

9. E. Kiyani, M. Soleimani-damaneh, Algebraic interior and separation on linear vector spaces: Some comments, Journal of Optimization Theory and Applications 161, p.994-998, 2014.

10. E. Kiyani, M. Soleimani-damaneh, Algebraic (relative) interior on linear vector spaces, Farhang va Andisheh Riazi, 53, p.55-71, 2013 (in Persian language).

Name: Elham KianiHarchegani

Post-Secondary Education and

University of Tehran
Tehran, Iran
2009 - 2011, M.Sc. of Applied Mathematics

Amirkabir University of Technology
Tehran, Iran
2013 - 2020, Ph.D. of Mathematics

University of Western Ontario
London, Canada
2018 - 2023, Ph.D. of Applied Mathematics and Computer Science

Related Work Experience:

Teaching Assistant and Research Assistant
The University of Western Ontario
2018 - 2023

Honours and Awards:

Ontario Graduate Scholarship (OGS)
2020

Ontario Graduate Scholarship (OGS)
2021

Mitacs Globalink Research Award Abroad
At Division of Applied Mathematics, Brown University, USA
2022

Western University Science International Engagement Fund Award
At Department of Mathematics, ETH Zürich, Switzerland
2023

Flight 752 Memorial Graduate Scholarship in Engineering and Science
Western University
2023