A Deep Learning Approach for Computational Electromagnetics

Mohammad Mushfiqur Rahman

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

MASTER OF SCIENCE

Department of Electrical & Computer Engineering

McGill University
Montréal, Québec, Canada

December 2023

Abstract

In the current status quo, electrical systems can be found in an ever-increasing range of products that we use in our daily lives. With the advances in technology, industries such as the automotive, communications and medical devices have been disrupted with new electrical and electronic systems. The innovation and development of such systems with increasing complexities over time has been supported by the increased use of Electromagnetic (EM) analysis software. Such software enables engineers to virtually design, analyze and optimize EM systems without the need for building physical prototypes, thus helping to shorten the development cycles, and consequently, cut costs. The purpose of this thesis is to develop surrogate models, using Deep Learning (DL), that can facilitate the application of EM analysis software.

The industry standard for simulating EM problems is using either the Finite Difference Method (FDM) or the Finite Element Method (FEM). Optimization of the design process using such methods requires significant computational resources and time. With the emergence of Artificial Intelligence (AI), along with specialized tools for Automatic Differentiation (AD), the use of Deep Learning (DL) has become computationally much more efficient and cheaper. These advances in Machine Learning (ML) have ushered in a new era in EM simulations where engineers can compute results much faster while maintaining a certain level of accuracy.

In this thesis, I have proposed two different models that can compute the magnetic field distribution in EM systems. The first model is based on a Recurrent Neural Network (RNN), which is trained through a data-driven supervised learning method. The second model is an extension to the first with the incorporation of additional physics-based information to our model. Such a DL model, which is constrained by the laws of physics, is known as a Physics-Informed Neural

Network (PINN). The solutions when compared with the ground truth, computed using FEM, show promising accuracy for our DL models while reducing the computation time and resources required, as compared to previous implementations in the literature.

This thesis presents a Neural Network (NN) architecture and is trained with two different learning methodologies, namely, supervised and physics-based. The working of the network along with the different learning methodologies is validated over several EM problems with varying levels of complexities. Furthermore, a comparative study is performed regarding performance accuracy and computational cost to establish the efficacy of different architectures and learning methodologies.

Abrégé

Dans le statu quo actuel, les systèmes électriques se retrouvent dans une gamme toujours croissante de produits que nous utilisons dans notre vie quotidienne. Avec les progrès technologiques, des secteurs tels que l'automobile, les communications et les appareils médicaux ont été perturbés par de nouveaux systèmes électriques et électroniques. L'innovation et le développement de tels systèmes, dont la complexités augmente au fil du temps, ont été soutenus par l'utilisation accrue de logiciels d'analyse électromagnétique (EM). Un tel logiciel permet aux ingénieurs de concevoir, d'analyser et d'optimiser virtuellement des systèmes EM sans avoir besoin de construire des prototypes physiques, contribuant ainsi à raccourcir les cycles de développement et, par conséquent, à réduire les coûts. Le but de cette thèse est de développer des modèles de substitution, utilisant le Deep Learning (DL), qui peuvent faciliter l'application de logiciels d'analyse EM.

La norme industrielle pour simuler les problèmes EM utilise soit la méthode des différences finies (FDM) ou la méthode des éléments finis (FEM). L'optimisation du processus de conception à l'aide de telles méthodes nécessite des ressources de calcul et du temps importants. Avec l'émergence de l'intelligence artificielle (IA), ainsi que des outils spécialisés de différenciation automatique (AD), l'utilisation du Deep Learning (DL) est devenue beaucoup plus efficace et moins coûteuse sur le plan informatique. Ces avancées en matière d'apprentissage automatique (ML) ont inauguré une nouvelle ère dans les simulations EM où les ingénieurs peuvent calculer les résultats beaucoup plus rapidement tout en maintenant un certain niveau de précision.

Dans cette thèse, j'ai proposé deux modèles différents permettant de calculer la distribution du champ magnétique dans les systèmes EM. Le premier modèle est basé sur un réseau neuronal récurrent (RNN), formé via une méthode d'apprentissage supervisé basée sur les données. Le deuxième

modèle est une extension du premier avec l'incorporation d'informations physiques supplémentaires à notre modèle. Un tel modèle DL, contraint par les lois de la physique, est connu sous le nom de réseau neuronal fondé sur la physique (PINN). Les solutions, comparées à la vérité terrain, calculées à l'aide de FEM, montrent une précision prometteuse pour nos modèles DL tout en réduisant le temps de calcul et les ressources requises, par rapport aux implémentations précédentes dans la littérature.

Cette thèse présente une architecture de réseau neuronal (NN) et est formée avec deux méthodologies d'apprentissage différentes, à savoir supervisée et basée sur la physique. Le fonctionnement du réseau ainsi que les différentes méthodologies d'apprentissage sont validés sur plusieurs problèmes EM avec différents niveaux de complexités. En outre, une étude comparative est réalisée concernant la précision des performances et le coût de calcul afin d'établir l'efficacité des différentes architectures et méthodologies d'apprentissage.

Related Publication

The Chapters 3 and 4 of this thesis have been published in *COMPEL — The international journal for computation and mathematics in electrical and electronic engineering* on 11 August 2023 [1]. I am the primary author of the paper, while Arbaaz Khan is the secondary author. Arbaaz Khan contributed towards about half of the data collection process. Arbaaz Khan along with the other authors — Prof. David A. Lowther and Prof. Dennis Giannacopoulos, contributed with the guidance on the research methodology, reviewing of the transcript, and analysis of the results.

Acknowledgements

Firstly, I would like to thank the Almighty for He is the reason for my existence. I am grateful for my health, my knowledge, and the opportunities I have been blessed with in my life.

I would like to express my sincere gratitude to my research supervisor, Prof. Dennis Giannacopoulos, for his unwavering guidance and support throughout the entire program. His expertise, patience, and invaluable insights were instrumental in shaping this thesis. I feel honoured to have got this opportunity to work under his supervision and learn from him.

I also want to thank Prof. David A. Lowther for his guidance and support in my research work. I would like to acknowledge the help and support received from my colleagues Arbaaz Khan, Amir Akbari, and the rest of the members of the Computational Electromagnetics (COMPEM) laboratory at McGill. Particularly Arbaaz, who has been like a friend and a mentor to me. The insightful discussions we had throughout the course of this thesis helped me become a better researcher.

I want to thank my parents, Mrs. Nayar Sultana Rahman and Engr. Md. Mokhlesur Rahman, my brothers, Taahmidur Rahman and Mahin Ar-Rahman, and my cousins, Aarpa Ibrahim, Raidah Ibrahim, Erfan Reza, and Kazi Shahul Hossain, for their unconditional love and support. Their encouragement and sacrifices have been the driving force behind my achievements.

Special thanks to all my friends, Samin Islam, Naaf Anowar, Samin Ihsan, and Tashreef Anowar, for their kind words and company, which made this journey a great experience.

Contribution of Authors

I, Mohammad Mushfiqur Rahman, the author of this thesis contributed to all the chapters (Chapters 1, 2, 3, 4, and 5) in it. Arbaaz Khan contributed towards about half of the data collection process discussed in Section 3.2 of Chapter 3.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AD Automatic Differentiation. ii, 77

AE Autoencoders. 44

AI Artificial Intelligence. ii, xii, 11, 12, 40, 42

AN Artificial Neuron. xii, 43, 45, 46, 70

ANN Artificial Neural Network. 45

ASHA Asynchronous Successive Halving Algorithm. 90

AWS Amazon Web Services. 44

BC Boundary Condition. xii, 23, 30, 31, 33, 35, 59, 76–80, 83, 90, 98

BN Batch Normalization. xii, 50–53

BNN Biological Neural Network. 45

BPTT Backpropagation Through Time. 66

CAD Computer-Aided Design. xii, 26, 61, 62

CEM Computational Electromagnetics. xii, 13, 14, 23–28, 57, 60, 84, 97–99

CFD Computational Fluid Dynamics. 5

CGI Computer-Generated Imagery. 8, 9

CNN Convolutional Neural Network. xiv, 43, 44, 54–59, 77, 85, 86, 89, 93, 98

COMPEM Computational Electromagnetics. vii, 25, 26

CPINN Conservative Physics-Informed Neural Network. 80

CPU Central Processing Unit. 8

CTE Coefficient of Thermal Expansion. 4

DC Direct Current. 64

DL Deep Learning. ii, iii, xii, 11–14, 42–45, 48–50, 57, 58, 60–63, 72, 75, 77, 80, 81, 84, 85, 89, 97, 98

DNN Deep Neural Network. 43–45, 50, 81, 84

DRM Design Reference Mission. 3

EDA Electronic Design Automation. 26

EM Electromagnetic. ii, iii, xii, 10, 13–19, 22–33, 35–37, 58–60, 64, 72, 75, 76, 80, 85, 97, 98

EMC Electromagnetic Compatibility. 26, 31

EMI Electromagnetic Interference. 26, 31

FC Fully Connected. 47, 54, 57, 73

FDFD Finite Difference Frequency-Domain. 24

FDM Finite Difference Method. ii, xii, 13, 24, 29–31, 34, 58, 60

FDTD Finite Difference Time-Domain. 24, 31

FE Finite Element. 33, 34

FEA Finite Element Analysis. 5, 14, 33, 64, 74, 97, 98

FEM Finite Element Method. ii, iii, xii, xiii, 13, 24–26, 33–35, 58–60, 84, 87–89, 91, 94–96, 98

FEMM Finite Element Method Magnetics. 64

FNN Feedforward Neural Network. 47, 58

FVTD Finite Volume Time-Domain. 31

GAN Generative Adversarial Network. 44

GCP Google Cloud Platform. 44

GIGO Garbage In, Garbage Out. 62

GO Geometrical Optics. 29

GPU Graphics Processing Unit. 44, 98

GRU Gated Recurrent Unit. xii, 68–70, 73, 75

GTD Geometrical Theory of Diffraction. 29

HFSS High-Frequency Structure Simulator. 25

hp-VPINN hp-Variational Physics-Informed Neural Network. 80

I/FC Interface Condition. 33

IC Initial Condition. xii, 10, 30, 31, 76–79

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 57

JWST James Webb Space Telescope. 1–6, 36

L-BFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno. 79

LN Layer Normalization. xii, 52, 53, 75

LRN Local Response Normalization. 57

LSTM Long Short-Term Memory. xii, 66–68, 70

ML Machine Learning. ii, xii, 11–13, 25, 40–45, 47, 50, 57, 62, 97, 98

MMF magnetomotive force. 37

MoM Method of Moments. 24, 32, 33

MRI Magnetic Resonance Imaging. 27

MSE Mean Squared Error. xii, xiii, 46, 74, 77–79, 83, 87, 88, 91

NASA National Aeronautics and Space Administration. 1, 2, 4

NLP Natural Language Processing. 44, 70

NN Neural Network. iii, xii, 13, 42–50, 52, 54, 57, 59, 65, 68, 70, 77–81, 83

ODE Ordinary Differential Equation. 29, 75, 76, 80

ODTK Orbit Determination Tool Kit. 3

PDE Partial Differential Equation. xii, 23, 29, 34, 59, 64, 75–81, 83, 90, 93, 98

PI-RNN Physics-Informed Recurrent Neural Network. xii–xiv, 13, 59, 81–84, 89–96, 98, 99

PINN Physics-Informed Neural Network. ii, iii, xii, 75–81

PML Perfectly Matched Layer. 31

PO Physical Optics. 29

POC proof-of-concept. 39, 84

PTD Physical Theory of Diffraction. 29

RF Radio Frequency. 15, 23, 25

RL Reinforcement Learning. 41

RN Recurrent Neuron. xii, 65, 66, 68, 70, 73, 74

RNN Recurrent Neural Network. ii, xii–xiv, 13, 44, 52, 59, 65, 66, 70–74, 77, 81, 84, 86–90, 93, 98

SRP Solar Radiation Pressure. 3

SSD Solid-State Drive. 44

STK Systems Tool Kit. 3

SVM Support Vector Machine. 43

TPU Tensor Processing Unit. 44

UTD Uniform Geometrical Theory of Diffraction. 29

WWDC Worldwide Developers Conference. 1

1

Introduction

1.1 Simulations

Did you watch the latest Worldwide Developers Conference (WWDC) [6], or the Google I/O [7], or Tesla Investor Day [8, 9], or watch the latest Avatar film [10], or have you played the latest PlayStation game God of War [11], or perhaps you have watched the launch of the James Webb Space Telescope (JWST) by National Aeronautics and Space Administration (NASA) [12]. Now, what do a new product launched by Apple, Google or Tesla, one of the highest grossing movies of all time, one of the best selling video games and a space telescope could possibly have in common? Even though it might sound far fetched, but they all do share an integral component in their development and production phases. It is the application of *computer simulations*! Simulations are virtual or computational models that mimic real-world processes or systems to analyze their behavior, test hypotheses, or make predictions [13]. They have been in use ever since the introduction of computers

and have grown as computers became more powerful. However, one might argue that (analog) simulations have existed before the advent of digital computers, such as simulating a scaled-down airplane in a wind tunnel to test its aerodynamics using analog computers [14]. Nonetheless, in this thesis, I focus on digital simulations only because experimental based simulation is costly to scale.

Computer simulation originates from World War II [15–17]. If you have watched or heard of Christopher Nolan's latest movie, Oppenheimer [18], then you must have an idea of the Manhattan Project [19]. It was during the Manhattan Project when two mathematicians Jon Von Neumann and Stanislaw Ulam were trying to analyze the behaviour of neutrons [17]. They used the Monte Carlo method to create a computer simulation in order to solve problems associated with neutron diffusion during the design of the hydrogen bomb [15, 17]. Such problems were impossible to solve analytically then and even now [15].

1.1.1 Case Studies on Real-World Applications of Simulations

Ever since its advent in the 1940s, computer simulations have become a key component in the advancement of human civilization. Their applications have proliferated in various industries for various purposes. Simulations have allowed to break the shackles of physical limitations, and push research and development further than ever before. In this section, I would like to discuss a couple of such groundbreaking success stories among the many that have been there in the past few decades.

James Webb Space Telescope (JWST)

The first of the success stories is that of the James Webb Space Telescope (JWST) by National Aeronautics and Space Administration (NASA). It is what NASA refers to as the Next Generation Space Telescope that will allow astronomers all-over the world to conduct scientific observations and provide a profound understanding of the origin of the universe [12]. The JWST is the largest, most advanced and powerful space telescope ever built by NASA till date [20]. Engineers had to design a structure that could be folded and placed inside a rocket. Launching such a gigantic telescope into space that needs to travel about 1.5 *million km* from earth and operate at about $-234^{\circ}C$ presented some unprecedented engineering challenges [21]. Furthermore, it had to undergo six months of commissioning in space — unfolding its mirrors, sun-shield, and other smaller systems; cooling down; aligning; and calibrating [20]. Accounting for all these delicate maneuvers, the JWST

had 344 single-point failures, where each single-point of failure would compromise the whole mission, making it one of the riskiest missions in history [22]. To mitigate risk and minimize the chances of failure, engineers had to test the JWST comprehensively. Unfortunately, the sheer size of the JWST made it extremely difficult to test everything in person, and tests carried out on Earth would not be sufficient to determine the performance in space. Hence, the only way to ensure the proper operation of the JWST was through simulation.

Unlike other spacecrafts, the JWST needs to operate at extremely low temperatures. To achieve that it uses passive cooling system which involves 5 huge sun-shields, each the size of a tennis court [20, 23]. These shields needed to unfold into shape during the trajectory of the telescope. The light from the Sun carries momentum, which can potentially push the spacecraft. During the unfolding process the change in the surface area of the spacecraft will cause a change in the pressure from the solar radiation [24, 25]. Hence, engineers had to use different simulation models for specific segments in the mission [24]. For instance, Orbit Determination Tool Kit (ODTK) was used to determine the optimal orbital path for the JWST [22, 24, 25]. Additional plugin, Solar Radiation Pressure (SRP) was used to model the pressure on the sun-shields on the telescope from the solar rays [22, 24]. Once the JWST reached the destination point, which is the Libration-point (L2), engineers needed to simulate complex Design Reference Missions (DRMs) to ensure that the telescope remains in the correct position [22, 24, 26]. To create such DRMs engineers used Astrogator in Systems Tool Kit (STK) which simulated the various gravitational forces along with solar radiation pressure at L2 [22].

Furthermore, the JWST uses 18 hexagonal mirrors to serve as one big primary mirror [21]. The mirrors were segmented so that it could be folded up into the rocket. To determine the feasibility of connecting such segmented mirrors edgewise, while considering the natural vibrational frequencies of the components holding them together, engineers used Ansys Mechanical simulator [27]. Another simulation software, Ansys Zemax was used to design and test the process to precisely align all the mirror segments to function as a monolithic mirror [22]. Engineers built a physical test bed of the telescope that's one-seventh the real size [22]. They used Ansys Zemax to simulate the alignment process before implementing it on the real hardware. To simulate the on-orbit conditions for the test bed telescope flight models were incorporated in Ansys Zemax. Such models provided probabilistic state of the mirror segments at each step of the alignment procedure along with the best and worst

case scenarios, which allowed the engineers to design the mirror actuators accordingly [22, 24].

Moreover, the JWST is a very sophisticated spacecraft which required its different components built in specific specialized facilities. These delicate components then had to be brought together for assembly and testing. The final testing took place at a Northrop Grumman factory in Redondo Beach, California [28], before it was shipped to its launch-site in French Guiana [29]. Engineers had to make sure that all these sensitive components and the final structure remain safe during the various transportation phases. As a result, they had to examine the vibrations from the different vehicles used and how the spacecraft components would react to them [30, 31]. Furthermore, they needed to examine how the rocket would handle such a heavy payload and safely launch into space without damaging it [30]. To handle such a crucial task engineers used Siemens' Simcenter Femap software to simulate the various transportation environments [30, 31]. It allowed them to design the components such that the real-world vibrations are well within the safety margin.

Another challenge was the extremely low operating temperatures of the JWST in space. On one end the sun-shield reaches about $110°C$ and on the other end the instruments operate at around $-236°C$, creating a sharp temperature gradient in the telescope [23]. This possesses an extremely difficult challenge in the form of Coefficient of Thermal Expansion (CTE) [30]. CTE characterizes the expansion of a material upon heating and vice versa [32]. Different materials have specific CTE which means that they would expand or contract at different rates. This could cause stress and strain in the structure of the telescope comprised of various materials, where the best case scenario would be a distorted image and the worst case would be a component coming off from the spacecraft [30]. Engineers handled this problem by carefully selecting materials that have similar CTE and then using their properties to simulate how the whole structure would behave in space at all the possible operating temperatures [30, 31]. Femap simulation allowed to ensure that the different components of the telescope would stress and strain in harmony and not cause any structural damage throughout the entire mission [30, 31].

The colossal size of the JWST, the sheer length of its trajectory, the extreme operating temperatures, and the harsh conditions of space pushed simulations of all sorts to their limits. However, those simulations made it possible to predict and visualize the whole mission way ahead of its launch, allowing engineers and scientists to minimize the possibilities of failure. Furthermore, NASA has made an almost real-time simulation of the mission available to the public to appreciate the leap in

scientific and technological advancement the JWST has achieved [33].

Airbus A380

Another impactful example of computer simulation in a real-world engineering project is the case of the Airbus A380, the largest commercial passenger aircraft [34]. Simulation played a vital role in the aerodynamic design of the A380, particularly in optimizing its wings. The aircraft's size and weight required innovative design solutions to ensure fuel efficiency and performance. Computational Fluid Dynamics (CFD) simulations were extensively employed to model the airflow around various wing configurations, allowing engineers to analyze the aerodynamic performance and identify areas for improvement.

1. **Fuel Efficiency Improvement:** The aerodynamic refinements achieved through simulations contributed to a remarkable improvement in fuel efficiency [35]. Airbus reported that the A380 consumes up to $8 - 20\%$ less fuel per seat compared to its closest competitor, making it one of the most fuel-efficient aircraft of its size [34, 36].

2. **Noise Reduction:** Simulations also assisted in reducing the aircraft's noise emissions during takeoff and landing. The A380 incorporates advanced wing designs and other aerodynamic features, leading to a 50% reduction in perceived noise at airports compared to previous-generation aircraft [34].

3. **Increased Payload and Range:** The aerodynamic enhancements allowed the A380 to carry more passengers and cargo over longer distances. With a typical seating capacity of around 550 passengers, the A380 has a range of approximately $8,000$ $nautical\ miles$ ($14,800$ $kilometers$) [34, 37].

4. **Structural Optimization:** In addition to aerodynamics, simulations were employed in structural analysis to optimize the aircraft's design. Finite Element Analysis (FEA) simulations helped ensure the structural integrity and safety of critical components, reducing overall weight while maintaining structural strength [38].

By leveraging simulations in the design process, Airbus successfully developed an aircraft that revolutionized long-haul air travel. The A380's fuel efficiency, range, and passenger capacity made it

a game-changer in the aviation industry. The impact of simulations in this engineering project led to significant improvements in aircraft performance, cost-effectiveness, and environmental sustainability.

1.1.2 Classification of Simulations

As we have seen for the JWST, there are various simulations that are deployed to solve real-world problems. As such, simulations can be categorized in many different ways. Sometimes they are classified based on the different fields of application that they are being used in, such as the aerospace industry, film industry, gaming industry, service industry, economic studies, scientific studies, etc. However, they can be broadly categorized based on the type of the model and the way it progresses through time that the simulation is implementing [14, 39]. The different types of simulations are illustrated in Figure 1.1 and discussed below:

- **Deterministic vs Stochastic Simulation:** If the model consists of all the input and output variables that are deterministic, i.e., without any randomness then it's a *deterministic simulation* [14, 39]. Exact mathematical functions are used to describe the models. Otherwise, if the model consists of at least one variable that is random then it's a *stochastic simulation* [14, 39]. A probabilistic function is used to describe the randomness in the variable.

- **Static vs Dynamic Simulation:** If the model doesn't change with time, i.e., the model's state remains constant then it's a *static simulation* [14, 39]. Monte Carlo Model is an example of a static simulation. Otherwise, if the model changes with time, i.e., the model's state varies with time then it's a *dynamic simulation* [14, 39]. For instance, a conveyor system in an airport.

- **Continuous vs Discrete Simulation:** If the model's state change can occur continuously with time then it's a *continuous simulation* [14, 39]. The simulation advances time with a constant rate, i.e., a fixed time slice. Otherwise, if the model's state change can occur only at discrete time points then it's a *discrete simulation* [14, 39]. Usually the change is event-based, which is why it's also called *discrete-event simulation*.

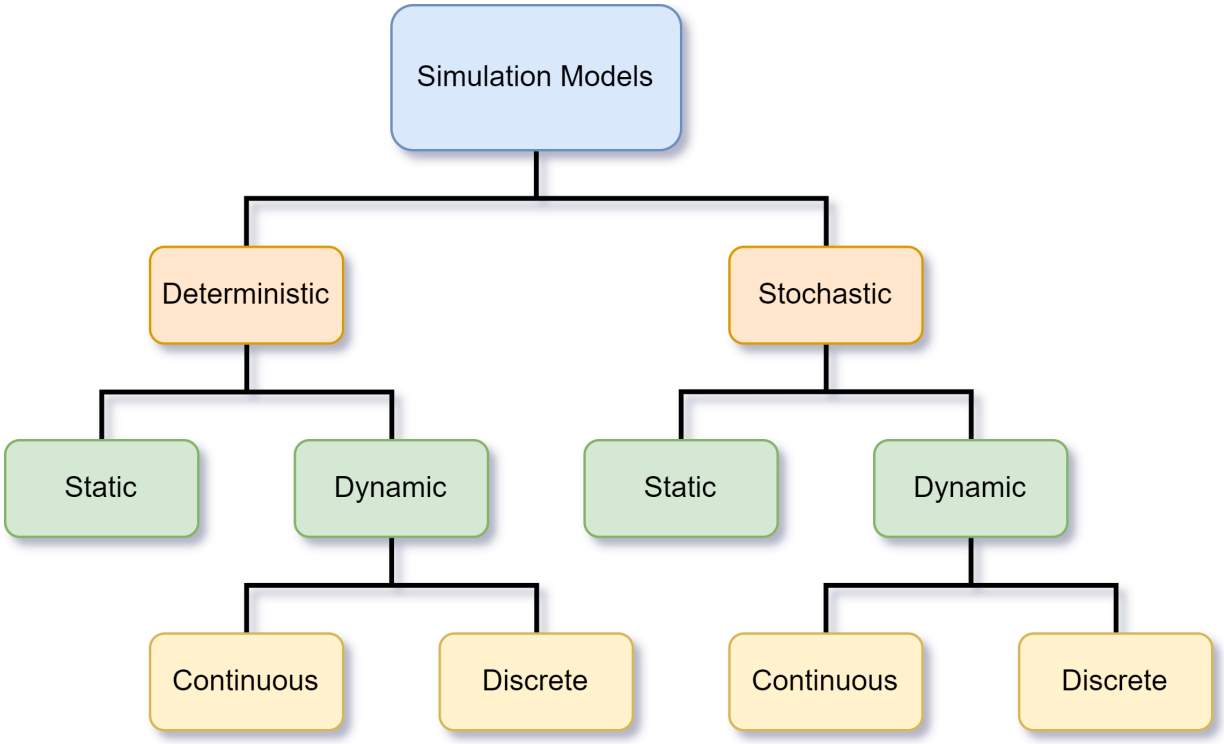

Figure 1.1: Classification of simulations

1.1.3 Why is Simulation Necessary?

Simulations serve as indispensable tools across diverse fields and industries, playing a pivotal role in expediting research and development. Their significance lies in enhancing the pace of progress and innovation within these domains:

1. **Predicting and Planning:** Simulations provide the means to model intricate systems and processes, thereby facilitating the prediction of their behavior under diverse conditions. This valuable capability proves instrumental in strategizing future actions, making well-informed decisions, and mitigating potential risks [40, 41].

2. **Cost-Effectiveness:** Performing real-world experiments entails significant expenses, consumes considerable time, and occasionally poses safety hazards. Simulations present a cost-effective alternative, empowering researchers and engineers to examine hypotheses and scenarios within a virtual environment, thereby circumventing the necessity for physical resources. For example, simulating crash tests for vehicles or testing the durability of materials can save time and resources [40, 41].

3. **Safety and Risk Assessment:** In fields like aviation, medicine, and engineering, simulations are used to evaluate potential risks and safety concerns without endangering human lives or causing damage to the equipment. Furthermore, some environments are challenging or impossible to access physically, such as outer space, the deep ocean, or microscopic scales. Simulations enable researchers to study and understand phenomena in such environments [40, 41].

4. **Training and Education:** Simulations are widely used for education and training purposes for various applications such as medical procedures, aviation maneuvers, and military operations. They offer a safe and controlled environment for learners to gain practical experience, refine their skills, and develop confidence before dealing with real-world situations [40].

5. **Design and Optimization:** Simulations offer a multifaceted advantage in engineering and research: they facilitate iterative improvement through immediate feedback, enable the creation of virtual prototypes, reducing reliance on physical ones and accelerating the design process, and aid in designing and optimizing products, processes, and systems by allowing engineers and designers to explore numerous virtual configurations to identify the most efficient and effective solutions [42, 43].

6. **Entertainment:** Simulations are the foundation of modern video games, interactive entertainment, and the mesmerizing visual effects in movies. They create immersive and realistic virtual worlds for users to enjoy and/or engage with. No experiment or place in the world could replace them, making simulations indispensable. For instance, the latest Avatar movie required about 18.5 *petabytes* of data, weeks of simulation, and millions of Central Processing Unit (CPU) hours for the Computer-Generated Imagery (CGI) [44].

Overall, simulation offers a controlled and versatile tool to understand and analyze complex systems, and predict outcomes in a wide range of fields, making it an indispensable component of modern research, development, and decision-making processes.

1.1.4 Applications of Simulations

Given the variety of types of simulation, the range of its applications is vast. The fundamental reason behind this is the fact that simulation is not just a specific application of a technique or algorithm, such as a linear program, but rather an approach for solving any real-life problem.

1. **Manufacturing Applications:** Used to simulate various manufacturing processes. These could help to analyze the impact of failure at different points in the whole process, and minimize synchronization delays of prefabricated parts before assembly [40].

2. **Automobile Applications:** Used to simulate the production process of automobiles. Design and analyze different automobile models [40].

3. **Military Applications:** Used to train soldiers under various conditions. Design and analyze new weapons [40].

4. **Aerospace Applications:** Used to simulate different models of plane, helicopters, etc. Flight simulators allow training new pilots without the use of actual planes [40].

5. **Financial Applications:** Used to to analyze how changes in policies, market conditions, or other factors might impact the economy [40].

6. **Gaming Applications:** Used to create virtual worlds with their own rules and physics, allowing players to interact with them [45].

7. **Medical Applications:** Used to enhance the efficiency of hospital management. Helps to analyze various diseases and allow early preventive measures [40].

8. **Scientific Applications:** Used to analyze various natural phenomena and design scientific experiments [16].

9. **Film Applications:** Used to create various visually pleasing scenes. Simulations used in movies are often referred to as CGI [46].

1.2 Different Approaches for Simulations

Simulations can be a very powerful tool for understanding complex systems, making predictions, and testing hypotheses in a controlled environment. To perform a simulation, we usually need to follow the procedures given below:

1. Define the *system* we are simulating.
2. Create a *mathematical* or *computational model*.
3. Set *Initial Conditions (ICs)*.
4. Use *algorithms* to iterate through time steps, observing how the *system* evolves.

Now, the *model*, in step 2, that is used to capture the essence of the real-world system, can be developed using more than just one approach. These techniques play a crucial role in determining the overall performance, evaluation, and interpretability of the simulation. Essentially, there are two different ways of constructing a model, and therefore, the simulation — using the first-principle or the data-driven approach.

1.2.1 First-Principle Simulation

A *first-principle model*, often referred to as a *first-principle simulation* or *calculation*, is a computational approach used in various scientific and engineering fields to understand, predict, and simulate the behavior of complex systems based on fundamental physical or mathematical principles [47–49]. This approach is also known as ab initio modeling or physics-based modeling. The term *first principle* refers to the fundamental laws, equations, and principles that govern the behavior of a system. In the context of physics and engineering, these principles typically include the laws of physics, such as Newton's laws of motion, Maxwell's equations for Electromagnetics (EMs), the laws of thermodynamics, and quantum mechanics, among others. The first-principle models for EMs are discussed in details in Section 2.3. The pros and cons of first-principle approach are discussed below [47, 48]:

Advantages:

- **Accuracy:** In situations where the underlying physics are well understood and controlled, first-principle models can be highly accurate.

- **Generalization:** First-principle models can extrapolate well beyond the range of test data, unlike empirical models that should be limited within the ranges of data used in their development.
- **Transparency:** First-principle models are highly interpretable because they are based on well-established physical laws and principles, and can be easily inspected for accuracy or completeness.

Disadvantages:

- **Complexity and Computational Cost:** First-principle models can be complex and may require specialized knowledge to develop and maintain. Furthermore, they can be computationally expensive to solve, especially for large-scale systems.
- **Data Requirements:** In practice, it's often necessary to combine first-principle models with empirical data to improve accuracy, which can be complex and time-consuming. Furthermore, estimating parameters in first-principle models can be challenging, as accurate values may not always be readily available, requiring additional experimentation or calibration.
- **Initial Development Time:** Creating a first-principle model from scratch can be time-consuming, especially when detailed knowledge of the system's physics is needed.

1.2.2 Data-Driven Simulation

A *data-driven model*, also known as an *empirical model* or *black-box model*, is a type of mathematical or computational model that is built primarily based on observed data, without explicit consideration of the underlying physical or mechanistic principles governing the system [47, 49]. Instead of relying on fundamental equations or first-principles, data-driven models use statistical and Machine Learning (ML) techniques to analyze and make predictions from data patterns. They are valuable when the underlying mechanisms governing a system are not well understood or when empirical data is the primary source of information. They are widely used in industries and research areas where large datasets are available and accurate predictions are needed. In this era of Big Data and Artificial Intelligence (AI), data-driven models form the core of ML, especially Deep Learning

(DL), which are discussed in depth in Section 2.5. The pros and cons of data-driven approach are discussed below [50]:

Advantages:

- **Efficiency:** Data-driven models can be very efficient at processing large amounts of data and extracting valuable insights, often outperforming traditional models in complex systems.
- **Flexibility:** Data-driven models can be flexible and adaptable to changing conditions, as they can be updated with new data as it becomes available.
- **Improved Decision-making:** By providing accurate and timely information, data-driven models can help organizations make better decisions.

Disadvantages:

- **Data Quality and Quantity:** Data-driven models require a substantial amount of high-quality data to perform well. If the data is inaccurate or incomplete, the model's predictions may be unreliable.
- **Complexity and Overfitting:** Data-driven models can be complex and may require specialized knowledge to develop and maintain. Furthermore, they can be prone to overfitting, where they memorize noise in the training data rather than capturing the underlying patterns.
- **Transparency:** Data-driven models are seen as "black boxes," i.e., they lack interpretability, making it challenging to understand how they arrive at their predictions.

1.2.3 Hybrid Simulation

In the recent years, there has been a focus towards a third type — the *hybrid approach*. Hybrid modeling refers to the integration of ML and AI technologies with physical modeling, based on first principles [51]. This approach combines first-principle-based models with data-driven models into a joint architecture, supporting enhanced model qualities, such as robustness and explainability. Figure 1.2 illustrates all the three approaches for simulations.

Hybrid modeling can be used to improve the accuracy and efficiency of simulations by combining the strengths of both first-principle and data-driven approaches [52, 53]. For example, a hybrid model might use ML to predict the behavior of a complex system in situations where it is difficult to derive an accurate physics-based model, while still relying on first principles to ensure that the predictions are physically plausible. Hybrid modeling is an active area of research, and new techniques and applications are being developed all the time [52, 53]. It has the potential to revolutionize many fields by enabling more accurate and efficient simulations, and by providing new insights into complex systems.

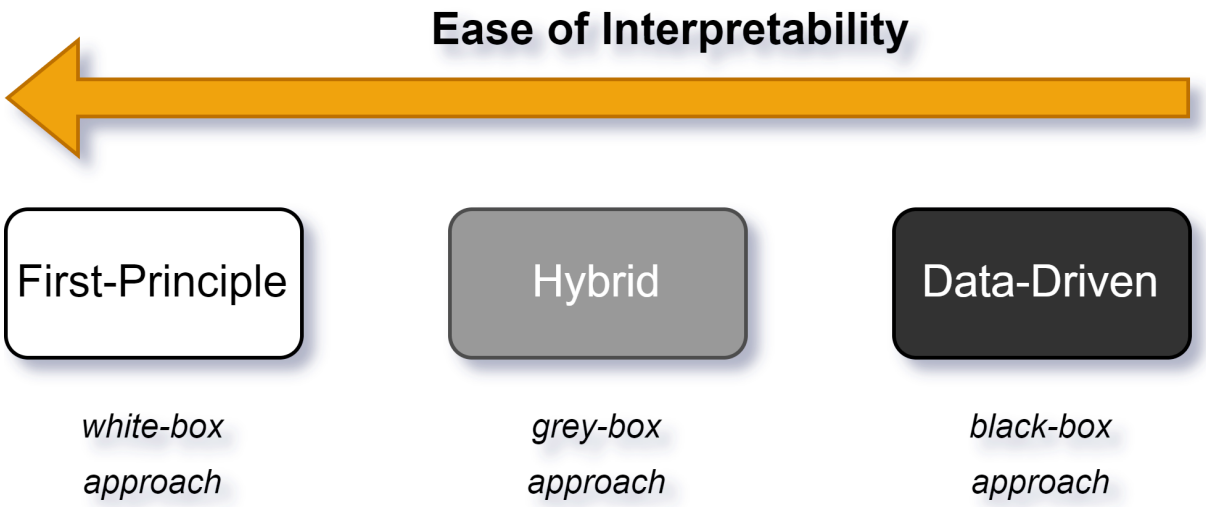

Figure 1.2: Different approaches for simulations

1.3 A Deep Learning (DL) Approach for Simulation

The focus of this thesis is to explore both the data-driven and the hybrid techniques for Computational Electromagnetics (CEM), i.e., simulation for EMs. Here, I have implemented both the approaches using Deep Learning (DL) to solve EM problems. The aim is to combine the strengths of both the first-principle methods, such as the Finite Element Method (FEM), the Finite Difference Method (FDM), etc., and the data-driven methods, like Neural Network (NN), Recurrent Neural Network (RNN), etc., to create a hybrid model, Physics-Informed Recurrent Neural Network (PI-RNN). Both the data-driven and the hybrid models are intended to reduce the computation costs

associated with conventional FEA solvers, and act as surrogates for them. Both the approaches are rigorously tested using different EM problems of varying complexities to examine their robustness and performance.

1.4 Thesis Overview

The rest of the thesis is organized in four chapters. Chapter 2 talks about all the necessary background knowledge required to assess and analyze the methodologies used in this work. Here the basics of EMs, CEM, and DL are discussed in details. Chapter 3 forms the core of this thesis as it illustrates the methodologies used. Here both the data-driven and the hybrid approaches using DL are described in depth. Chapter 4 talks about the experimental results of both the types of techniques, which are evaluated against the first-principle-based model solutions. The performances are then compared with that of a previous data-driven model in the literature, used as a benchmark. Finally, I conclude the thesis with Chapter 5.

2

Background

2.1 Electromagnetics (EMs)

In today's modern world, the most widely used electronic device is a mobile phone (also known as a cell phone). According to the data from 2022, about 96.1% of the global consumers own one [54]. However, the first cellular mobile phone was built only less than 50 years ago by Dr. Martin Cooper and his team in Motorola in 1973 [55]. These devices communicate using Radio Frequency (RF) waves. The foundation for this long-distance wireless communication was laid out by Nobel Prize laureate Guglielmo Marconi. In 1895, he built a system to send the first message using radio waves over a distance of several kilometers [56, 57]. Just as Marconi's telegraph system, most of the electronic gadgets encompassing our daily lives from television, computers, washing machines to electric cars, satellites and all forms of wireless communication are all predicated on the fundamentals of the *Electromagnetic (EM) theory.*

2.1.1 Evolution of the EM Theory

The evolution of the EM theory could be traced back to as early as 1785 with the publication of Coulomb's law [58]. Charles Augustin de Coulomb, a colonel in the Engineering Corps of the French Army and an experimentalist in electricity and magnetism, formulated the law of electric forces on charges [58]. It states that the magnitude of the forces between two charged bodies are directly proportional to the product of their charges, and inversely proportional to the square of the distance between them. It further mentions that like charges experience forces of attraction and opposite charges experience forces of repulsion. Coulomb's law led to the quantification of charges, with the unit being named in his honour (**C** for *coulomb*), which was crucial to the development of the EM theory [59].

Originally, electricity and magnetism were considered independent fields of study. In April of 1820, it was a Danish physicist, Hans Christian Oersted who first discovered the connection between these two forces [58]. During the preparation for a classroom demonstration Oersted observed that the current in a wire would deflect the needle of a nearby compass. Later, upon further investigation he concluded that an electric current produces a magnetic field, giving rise to the concept of electromagnetism. Furthermore, he illustrated that not only a current deflects a magnetic needle, but a magnetic field deflects a current carrying wire. Oersted's demonstration of this connection between electricity and magnetism is considered pioneering to the modern study of electromagnetism.

Inspired by Oersted's findings, by the end of September 1820, French mathematician and physicist Andre-Marie Ampere, made some ground breaking discoveries on magnetic forces on current carrying wires [58]. He demonstrated that two parallel wires carrying current in the same direction attract each other, and repel if in the opposite direction. His experiments conceptualized the science of magnetic fields generated through electric current, showing how the magnetic fields encircle a wire with current. In November of 1820, he introduced his circuital law of addition of magnetic forces which was the basis of the general equation known today as Ampere's (circuital) law [58]. It states that the line integral of the magnetic flux density vector around a closed loop is directly proportional to the total electric current flowing through that enclosed loop. Ampere is honoured by the use of *ampere* (**A**) as the unit for electric current.

In September of 1821, Micheal Faraday, an English physicist and chemist, repeated Oersted's experiments [58]. In the process, he revealed that the magnetic field around a straight current wire is circular. Influenced by observing the patterns formed by iron fillings around a magnet, Faraday introduced the concepts of electric and magnetic field lines which provided a new perspective of research on electricity and magnetism. One of his most popular experiments was the one conducted in August 29, 1831, where he wound two coils on a single iron ring [58]. He observed that changing the current in one coil induced a current in the other. He, therefore, concluded that the change in magnetic field with time leads to the creation of electric current, hence, discovered EM induction. Faraday's induction ring was essentially the world's first electric transformer. Further experiments led to the invention of Faraday's wheel, the first dynamo, i.e., electric generator. Faraday used his concept of field lines to explain his experimental observations. He proposed that the magnitude of the induced current in a conductor is directly proportional to the number of magnetic field lines being cut by the conductor per unit time. This hypothesis is consistent with the more mathematical formulation of the phenomenon, currently known as Faraday's law of EM induction. It was this discovery that played a pivotal role in converting electricity from merely a scientific fascination to a powerful technology in the remainder of the 19th century. The unit of capacitance is named $farad$ (**F**) in his honour.

Finally, collecting all the discoveries in the field of electricity and magnetism up until his time, it was James Clerk Maxwell, a Scottish physicist, who connected all the dots and conceptualized the EM theory [58]. It all began with his fascination of Faraday's experiments and theoretical speculations. He published two of his famous papers — *"On Faraday's Lines of Forces"* (1856) and *"On Physical Lines of Force"* (1861), where he formulated the mathematical model of Faraday's hypothesis of field lines [58]. Later in 1873, in his famous book *"A Treatise on Electricity and Magnetism"*, he introduced the complete classical EM theory [58]. Here he summarized and synthesized all the research and experiments performed by his predecessors such as Coulomb, Oersted, Ampere, Faraday and many others, into a unified mathematical framework to explain the fundamental laws governing electricity and magnetism. The framework consists of four fundamental equations which are known today as Maxwell's equations, discussed in Section 2.1.2, and is used as the basis of EM theory. He is considered the greatest name in EM theory, and often referred to as the father of electromagnetism. Even Albert Einstein mentioned, *"the formulation of these equations is the most*

important event in physics since Newton's time" [58]. Through his mathematical model, Maxwell derived that a changing electric field with time must always be accompanied by a changing magnetic field with time. This was achieved through the introduction of displacement current, which was one of Maxwell's biggest contributions. Furthermore, it led to the conceptualization of EM waves, and the hypothesis of a possible family of them with varying frequency and wavelength. Furthermore, he identified light as an EM radiation, and calculated its speed from his equations.

In his publications, Maxwell's mathematical model for EM theory was defined in Cartesian coordinates. This led to long complex equations in order to express certain phenomena such as the curl and the divergence of a flux. Few years later, it was an English electrical engineer, mathematician, and physicist, Oliver Heaviside who introduced the vector notation in Maxwell's equations [58, 60]. Heaviside used scalar and vector products along with other operators such as the *grad*, *div*, and *curl* to re-write Maxwell's model with a compact set of four equations leading to its modern version, discussed in Section 2.1.2 [60]. Furthermore, Heaviside laid out the mathematical foundation of the guided EM waves theory in his series of publications in the 1880s and in his book *"Electromagnetic Theory"* in 1893 [58].

However, it was only after Maxwell's death that his EM theory was confirmed by the German physicist Heinrich Rudolf Hertz in 1887 [58]. Hertz was the first to experimentally prove and showcase EM waves and EM radiation. In his famous laboratory experiment in 1887, Hertz created the first radio (wireless) link, and demonstrated the transmission and reception of radio (EM) waves over several meters [58]. Thus, verifying Maxwell's formulation of electromagnetism. The unit of frequency, measured in cycles per second, of waves is named *hertz* (**Hz**) to honour his contributions.

After the groundbreaking discovery of radio waves, Guglielmo Marconi, in 1895, took this technology and other subsequent radio engineering inventions to develop the wireless communication over several kilometers. Marconi kept on improving his pioneering wireless system, and his contributions in laying the foundation for wireless telegraphy led him to his Nobel Prize in physics in 1909 [56, 57]. Ever since, the radio science has evolved through the decades and has become an essential component of modern technology for wireless communication used by systems ranging from satellites in the orbit to smart phones in our hands.

2.1.2 Maxwell's Equations

Modern EM theory is essentially predicated on *Maxwell's equations*. Research on electricity and magnetism culminated in Maxwell's book *"A Treatise on Electricity and Magnetism"*, in 1873, leading to a coherent set of equations [58]. Maxwell summarized the fundamental laws governing electromagnetism with his four equations. However, they are also simultaneously referred to by the name of the scientist who had discovered the individual phenomenon in the first place. Maxwell's equations can be applied to any scenario of electromagnetism, with the most general form being for a rapidly time-varying EM field in arbitrary EM media, which is discussed as follows:

Maxwell's 1^{st} Equation — Faraday's Law

Maxwell's 1^{st} equation originates from *Faraday's law of EM induction*. Faraday's law states that the rate of change of a magnetic flux induces a directly proportional electromotive force, leading to a flow of electric current. Maxwell's equation further elaborates that a time-varying magnetic field gives rise to an electric field circulating around it. The integral form is given by Equation 2.1, and the differential form is given by Equation 2.2:

$$\oint_C \mathbf{E} \cdot d\mathbf{l} = -\int_S \frac{\partial \mathbf{B}}{\partial t} \cdot d\mathbf{S}, \tag{2.1}$$

$$curl\ \mathbf{E} = \nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}, \tag{2.2}$$

where $\mathbf{E}$ is the electric field vector and $\mathbf{B}$ is the magnetic flux density vector. Here, the *curl* of $\mathbf{E}$ is proportional to the rate of change of the surrounding $\mathbf{B}$. Equation 2.3 describes the general relation between the magnetic flux density vector, $\mathbf{B}$ and the magnetic field vector, $\mathbf{H}$:

$$\mathbf{B} = \mathbf{B}(\mathbf{H}) \tag{2.3}$$

Furthermore, Equation 2.4 describes the relation for a linear material, based on the magnetic permeability, μ, as shown below:

$$\mathbf{B} = \mu\mathbf{H} \tag{2.4}$$

Maxwell's 2^{nd} Equation — Ampere's Law

Maxwell's 2^{nd} equation is predicated on *Ampere's Circuital law.* Ampere's law states that the sum of the magnetic field along an enclosed arbitrary path, C is numerically equal to the electric current flowing through the surface, S enclosed by the path. However, the law couldn't explain the phenomenon for rapidly time-varying fields in certain scenarios, such as a capacitor with high-frequency current. Maxwell corrected and completed Ampere's law with his hypothetical entity known as displacement current. The revised equation now not only takes the electric current but also the change in the electric flux density into consideration, as shown in Equations 2.5 (integral form) and 2.6 (differential form):

$$\oint_C \mathbf{H} \cdot d\mathbf{l} = \int_S \left(\frac{\partial \mathbf{D}}{\partial t} + \mathbf{J} \right) \cdot d\mathbf{S}, \tag{2.5}$$

$$curl\ \mathbf{H} = \nabla \times \mathbf{H} = \frac{\partial \mathbf{D}}{\partial t} + \mathbf{J}, \tag{2.6}$$

where $\mathbf{H}$ is the magnetic field vector, $\mathbf{D}$ is the electric flux density vector (due to displacement current), and $\mathbf{J}$ is the electric current density vector. Equations 2.7 and 2.8 show the general relation of $\mathbf{D}$ and $\mathbf{J}$ with $\mathbf{E}$, the electric field vector, respectively:

$$\mathbf{D} = \mathbf{D}(\mathbf{E}) \tag{2.7}$$

$$\mathbf{J} = \mathbf{J}(\mathbf{E}) \tag{2.8}$$

Furthermore, for linear materials the relation is described using Equations 2.9 and 2.10:

$$\mathbf{D} = \epsilon \mathbf{E}, \tag{2.9}$$

$$\mathbf{J} = \sigma \mathbf{E}, \tag{2.10}$$

where ϵ is the permittivity, and σ is the conductivity of linear materials.

Maxwell's 3^{rd} Equation — Gauss' Law

Maxwell's 3^{rd} equation, also known as *Gauss' law*, states that the total amount of electric charges present within the volume of an enclosed surface is equal to the total electric flux flowing out of the enclosed surface, and is given by Equation 2.11. In terms of vector notation, it can be stated that the divergence of the electric flux density vector, $\mathbf{D}$ at any given point is equal to the volume electric charge density, ρ_v, as shown in Equation 2.12.

$$\oint_S \mathbf{D} \cdot d\mathbf{S} = \int_v \rho_v dv \tag{2.11}$$

$$div\ \mathbf{D} = \nabla \cdot \mathbf{D} = \rho_v \tag{2.12}$$

It can be inferred from Gauss' law that positive electric charges act as sources of an electric field, and negative charges as sinks. Furthermore, an electric field always starts from a positive charge and diverges away from it, and converges toward a negative charge and stops at it.

Maxwell's 4^{th} Equation — Gauss' Law for Magnetism

Maxwell's 4^{th} equation, also known as *Gauss' law for magnetism*, is analogous to his 3^{rd} law except that it describes the magnetic field instead of the electric field. It states that the net amount of magnetic flux entering a region of space is equal to the net amount leaving, as given by Equation 2.13. Therefore, the divergence of the magnetic flux density vector, $\mathbf{B}$ at any given point is always equal to zero, as shown in Equation 2.14.

$$\oint_S \mathbf{B} \cdot d\mathbf{S} = 0 \tag{2.13}$$

$$div\ \mathbf{B} = \nabla \cdot \mathbf{B} = 0 \tag{2.14}$$

The inference from Gauss' law for magnetism is that there are no equivalent "magnetic charges", for magnetic fields, to the electric charges for electric fields. In fact, magnets always occur in dipoles — north and south. Furthermore, magnetic fields, away from magnetic dipoles, always flow in closed

loops.

2.1.3 Poisson's Equation for Magnetic Field

The goal of this thesis is to evaluate the magnetic flux density distribution in various slowly time-varying (low-frequency) EM systems. Here, I would like to focus on what these systems are and how we can conceptually perceive the magnetic field distributions in them. In order to realize how the magnetic flux is generated in a low-frequency EM system we need to focus on Maxwell's 2^{nd} equation (also known as Ampere's law), previously discussed in Section 2.1.2. For a slowly time-varying field, the displacement current can be ignored, resulting in Equation 2.15:

$$\oint_C \mathbf{H} \cdot d\mathbf{l} = \int_S \mathbf{J} \cdot d\mathbf{S} \tag{2.15}$$

We can use the right-hand rule to find the relative directions — the thumb pointing the current flow and the other fingers pointing the contour.

For the problems addressed in this work, I prefer to use the differential form of Ampere's law which relates the field vectors at a point to the corresponding field sources at the same point, given by Equation 2.16:

$$curl\ \mathbf{B} = \nabla \times \mathbf{B} = \mu \mathbf{J}, \tag{2.16}$$

where the magnetic field vector, $\mathbf{H}$ has been replaced by the magnetic flux density vector, $\mathbf{B}$ using the relation given by Equation 2.4. We can further simplify the equation by using the relationship between magnetic flux density vector, $\mathbf{B}$ and magnetic vector potential, $\mathbf{A}$ given by Equation 2.17. This leads to the final form of the equation, also known as Poisson's equation for magnetic field, relating $\mathbf{A}$ and $\mathbf{J}$ as shown in Equation 2.18.

$$curl\ \mathbf{A} = \nabla \times \mathbf{A} = \mathbf{B} \tag{2.17}$$

$$-\nabla^2 \mathbf{A} = \mu \mathbf{J} \tag{2.18}$$

When dealing with any magnetic material, we need to consider the relative permeability of the

material, μ_r along with the permeability of vacuum, μ_0, which results in Equation 2.19 shown below:

$$-\nabla^2 \frac{1}{\mu_r \mu_0} \mathbf{A} = \mathbf{J} \tag{2.19}$$

2.2 Computational Electromagnetics (CEM)

Computational Electromagnetics (CEM) is essentially a branch of EMs that focuses on using computational methods and numerical techniques to solve EM field problems. It involves the application of mathematics, physics, and computer science to simulate and analyze EM phenomena. Generally, any EM problem can be defined by the EM theory, discussed in Section 2.1.1, i.e., through Maxwell's equations. The most accurate solution of those equations can be attained through analytic methods such as separation of variables, series expansion, conformal mapping, etc. [3]. Unfortunately, in most real-world scenarios it's extremely difficult to impossible to find the closed-form solutions of Maxwell's equations analytically due to a multitude of reasons, as mentioned below [61]:

- The Partial Differential Equation (PDE) is non-linear and cannot be linearized without compromising the result.
- The solution region is too complex.
- The Boundary Conditions (BCs) are of mixed types.
- The BCs depend on time.
- The medium is inhomogeneous or anisotropic.

However, computational methods used to solve the EM problems can overcome these issues. Hence, CEM plays a crucial role in the design, analysis, and optimization of various EM devices, systems, and structures.

The foundation of CEM lies in Maxwell's equations, a set of four partial differential equations that describe how electric and magnetic fields interact, as discussed in Section 2.1.2. CEM involves simulating the behavior of EM fields, including electric and magnetic fields, EM waves, and radiation patterns. This can be done in various frequency domains such as static, low-frequency, RF,

microwave, and optical. However, the focus of this thesis will be on low-frequency EM simulations, as mentioned in Section 2.1.3. CEM relies on numerical methods to discretize and solve Maxwell's equations. Common numerical techniques include Finite Difference Method (FDM), Finite Difference Time-Domain (FDTD), Finite Difference Frequency-Domain (FDFD), Method of Moments (MoM) and Finite Element Method (FEM), among others [62–64].

2.2.1 A Brief History of CEM

The use of simulations in EMs, also known as Computational Electromagnetics (CEM), can be traced back to the development of computers and computational methods. The field of electromagnetism itself dates back to the 19^{th} century, as discussed in Section 2.1.1. However, the application of computational simulations to solve EM problems started to become more prominent in the latter half of the 20^{th} century. Here are some key milestones:

- **1940s and 1950s:** Until the 1940s, people used to find ingenius ways to solve complex EM problems using the analytic methods [3]. The earliest attempts at simulating EM fields were made using analog computers [17]. Engineers and scientists used physical models and electronic circuits to represent EM phenomena. These early simulations were limited in scope and accuracy.

- **1960s:** With the advent of digital computers, numerical methods for solving EM field equations started to emerge [3]. The FDM [2] and the FEM [65, 66] were among the early techniques used for EM simulations. These methods allowed for the approximate solution of Maxwell's equations in various practical applications.

- **1970s:** As computer technology improved, simulations in electromagnetism began to address more complex problems. Early applications included modeling microwave propagation and antenna design. One notable development was the MoM [67], which was particularly well-suited for problems involving wire antennas and scattering from conducting bodies.

- **1980s:** The FDTD method was introduced [68], revolutionizing EM simulations. FDTD allowed for the direct numerical solution of Maxwell's equations in both time and space domains.

This method quickly gained popularity and is still commonly used today for a wide range of EM problems.

- **1990s:** Commercial software packages dedicated to EM simulations, such as Infolytica MAGNET [69] and Ansys Maxwell [70] became available. These tools provided engineers and researchers with user-friendly interfaces and powerful simulation capabilities for designing and optimizing microwave and RF components, antennas, and more.

- **2000s:** The use of EM simulations continued to grow in various industries, including telecommunications, aerospace, automotive, and electronics. Simulation tools such as COMSOL [71] played a crucial role in the development of advanced technologies like metamaterials, photonic devices, and wearable electronics.

- **2010s and Present:** Today, EM simulation is an essential tool in the design and analysis of a wide range of devices and systems, from integrated circuits and antennas to radar systems and optical devices. The simulations have become more accurate and efficient, thanks to advancements in computational techniques and hardware. Furthermore, the advent of ML has ushered in a new era in EM simulations [72–74].

Simulations in EMs have evolved from basic analog models to sophisticated numerical methods running on high-performance computers. The development of specialized software and tools dedicated to EM simulations, like the widely used application software packages such as Infolytica MAGNET [69], Ansys Maxwell [70], Ansys High-Frequency Structure Simulator (HFSS) [75], COMSOL Multiphysics [76] and CST Studio Suite [77], has significantly advanced the field. These tools have enabled engineers and researchers to tackle complex EM problems, optimize designs, and develop innovative technologies that have transformed various industries.

Computational Electromagnetics (COMPEM) Laboratory at McGill

The history of EM simulations cannot be discussed without mentioning my Computational Electromagnetics (COMPEM) lab at McGill, the birthplace of CEM in Canada [78]. It was established by Prof. Peter Peet Silvester, who began his academic career in Electrical Engineering department at McGill in 1964, shortly after receiving his PhD [78]. Prof. Silvester was the first to apply FEM

on EMs [65], consequently earning him the global recognition as a pioneer in the simulation of EM fields. He co-authored the seminal book *"Finite Elements for Electrical Engineers"* in 1983 [79], along with Prof. Ronald Leslie Ferrari, which is the first textbook dealing specifically with the application of the FEM to problems in electrical engineering. This book, in its third edition, is still considered the Bible of FEM for electrical engineers [80]. Prof. Silvester's contributions in the field were later summarized by Prof. Ronald L. Ferrari in [66]. Furthermore, the first textbook on Computer-Aided Design (CAD) for magnetics, published by Prof. Silvester and Prof. David A. Lowther [81], is another example of contribution from my lab. Over the past decades, the professors and students from the COMPEM lab have carried the torch of Prof. Silvester's legacy, and thrived in the field of CEM. One such notable achievements is that of Zoltan J. Cendes who founded Ansoft Corporation that developed the Electronic Design Automation (EDA) software[82].

2.2.2 Applications of CEM

Even though the fundamental purpose of CEM still remains to solve a given EM problem for any irregular geometry, which makes it a very strong and powerful tool, CEM is used for a wide range of applications:

1. **Design and Analysis:** It is used as the primary tool for the design and analysis of complex EM structures and systems, such as antennas, microwave circuits, integrated circuits, printed circuit boards, metamaterials, and photonic devices [62–64].

2. **Optimization:** It is used for optimizing the performance of EM devices and systems by iterating through various design parameters to achieve desired specifications [62–64]. This is crucial in industries like telecommunications, aerospace, and electronics.

3. **Electromagnetic Compatibility (EMC) Analysis:** EMC is the ability of an electrical or electronic system to operate as intended without either being affected by its EM environment or causing EM pollution itself [83].

4. **Electromagnetic Interference (EMI) Analysis:** EMI is the unwanted noise/interference/disturbance in an EM system originating from an external source [83].

5. **Signal Propagation Studies:** It is used to study scattering and radiation patterns of objects, which is essential in fields like radar, remote sensing, and satellite communication [62–64].

6. **Material Characterization:** It is employed in determining the EM properties of materials, which is crucial for designing and analyzing EM devices [62–64]. This includes modeling material properties like permittivity, permeability, and conductivity.

7. **Coupled Physics Problems:** It can be extended to solve coupled physics problems, where EM fields interact with other physical phenomena such as thermal, mechanical, or acoustic effects [63].

8. **Biomedical Applications:** It is used to model the interaction of EM waves with different parts of human body. Use cases include microwave breast tumor detection and monitoring, light propagation through retinal photoreceptors, absorption of electromagnetic power from the cellular telephone by the human head tissues, and many others [63, 84]. Furthermore, it facilitates the design of medical devices such as the Magnetic Resonance Imaging (MRI) scanners [63, 85].

9. **Education and Research:** It is a vital tool for researchers and educators in EMs and related fields. It allows students and scientists to explore and understand complex EM phenomena and develop new solutions and technologies [84].

CEM has become increasingly important in modern technological development and research, enabling engineers and scientists to tackle complex EM problems and design innovative devices and systems with greater accuracy and efficiency. It is heavily used in numerous industries, including telecommunications, electronics, aerospace, defense, and photonics, among others [62–64].

2.3 Classification of Computational Electromagnetics (CEM) Methods

Ever since the initiation of CEM in the 1960s, there have been various software applications developed for solving a variety of real-world EM problems. However, these software are essentially based on certain specific techniques to tackle the problems. These techniques can be broadly categorized

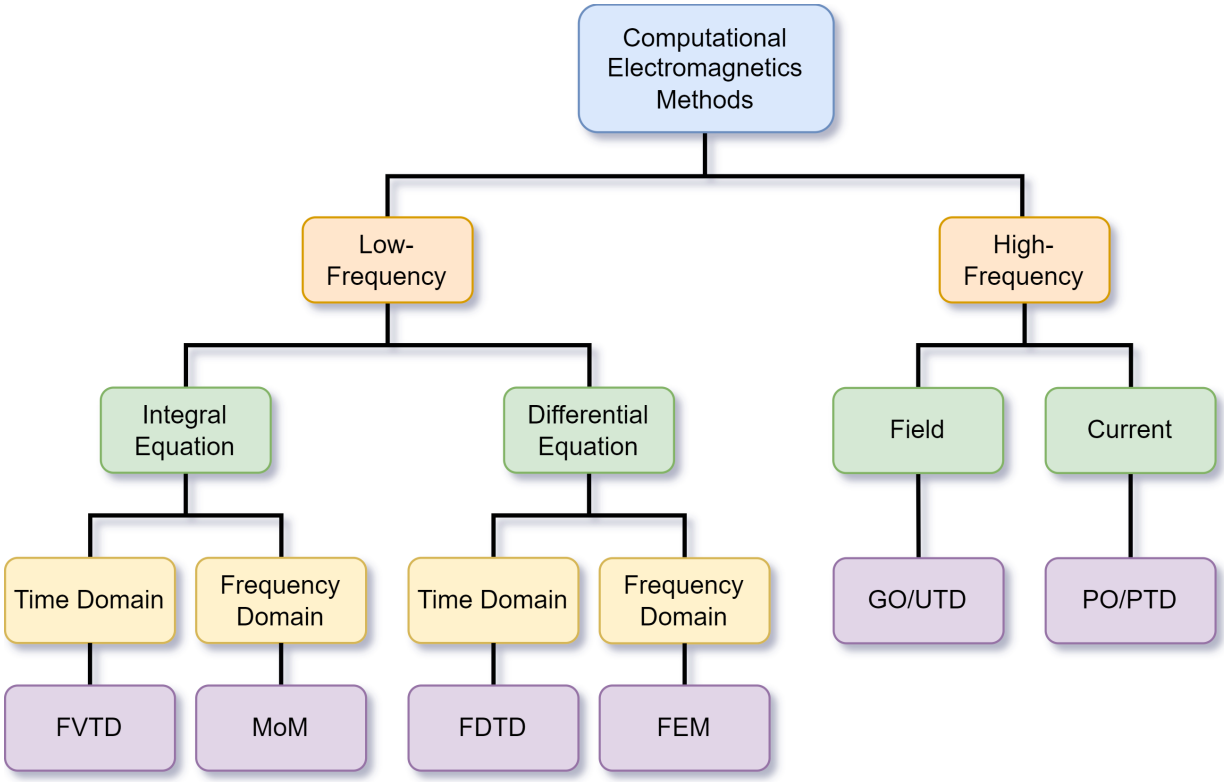

Figure 2.1: Classification of Computational Electromagnetics (CEM) methods [2]

based on the type of the EM problem under consideration. Classification of the CEM methods is illustrated in Figure 2.1 and discussed below:

- **High vs Low Frequency:** High frequency methods solve Maxwell's equations through simplifying hypothesis, which is only applicable if the wavelength is much smaller than the size of the object being modeled [2]. On the other hand, low-frequency methods, more commonly known as full-wave methods, are used where the wavelength is comparable to the geometrical characteristics of the object being modeled [2].

- **Field Based vs Current Based:** Field based techniques evaluate the scattered EM field given the incident field and a complex object. On the other hand, current based techniques try to approximate the currents induced on the object by the impinging field [2].

- **Time vs Frequency Domain:** Time-domain methods focus on how a signal changes over time, while frequency-domain methods focus on how the signal is distributed over a range of frequencies [2].

For high-frequency field based EM problems, methods like Geometrical Optics (GO) [2], Geometrical Theory of Diffraction (GTD) [86], or the upgraded Uniform Geometrical Theory of Diffraction (UTD) [86] frameworks are used. And for current-based problems, methods like Physical Optics (PO) [87] or the enhanced Physical Theory of Diffraction (PTD) [88] are used. However, as mentioned earlier in Sections 2.1.3 and 2.2, in this thesis my focus is on low-frequency EM problems. Hence, the low-frequency or full-wave methods have been discussed in further details.

2.3.1 Finite Difference Method (FDM)

The *FDM* was first introduced by A. Thom in the 1920s as "the method of squares" to solve nonlinear hydrodynamic equations [89]. However, it wasn't until around 1962, when FDM had its initial applications in EM field by Frederick C. Trutt [90].

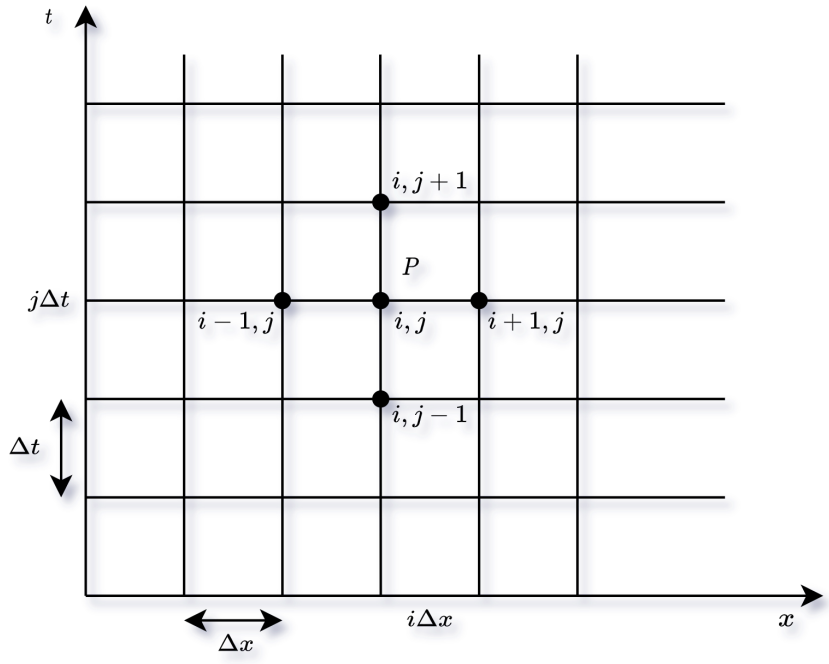

Figure 2.2: Finite Difference Method (FDM) mesh for two independent variables — x and t [3].

The FDM is a numerical technique used for solving differential equations, particularly PDEs and Ordinary Differential Equations (ODEs). The method owes its name to the process of approximating the derivatives of a function by replacing them by finite difference equations [3]. By doing so, it transforms the continuous differential equation into a system of algebraic equations that can be

solved numerically on a computer (or manually) [3]. The FDM connects the value of the dependent variable at any given point to its neighbouring points in the solution space, as shown in Figure 2.2. The steps required to solve any given EM problem using FDM are as follows [3]:

1. **Discretization:** The first step is to discretize the domain of the problem. For example, in a one-dimensional case, the interval of interest is divided into a series of discrete points. Similarly, for two or three-dimensional problems, a grid of nodes is created.

2. **Approximation of Derivatives:** The derivatives in the differential equation are approximated using finite difference formulas. This leads to a system of algebraic equations involving the values of the function at discrete points. The most common approximations include [3]:

   (a) **Forward-Difference:** Approximates the derivative at a point using values at that point and a nearby point ahead, given by Equation 2.20:

   $$f^{'}(x_o) \simeq \frac{f(x_o + \Delta x) - f(x_o)}{\Delta x} \tag{2.20}$$

   (b) **Backward-Difference:** Approximates the derivative at a point using values at that point and a nearby point behind, given by Equation 2.21:

   $$f^{'}(x_o) \simeq \frac{f(x_o) - f(x_o - \Delta x)}{\Delta x} \tag{2.21}$$

   (c) **Central-Difference:** Approximates the derivative at a point using values at that point and points on both sides, given by Equation 2.22:

   $$f^{'}(x_o) \simeq \frac{f(x_o + \Delta x) - f(x_o - \Delta x)}{2\Delta x} \tag{2.22}$$

3. **Solving the Algebraic System:** The resulting algebraic system can be solved using various numerical methods, such as Gaussian elimination, iterative methods like the Jacobi or Gauss-Seidel method, or specialized solvers for certain types of problems. While solving the equations, the BCs and/or ICs of the given problem need to be taken into account.

The specifics in the all three steps are determined based on the nature of the problem, the solution

region, the BCs and the ICs. The choice of discretization scheme and grid size can significantly impact the accuracy of the results, and care must be taken to ensure stability and convergence.

2.3.2 Finite Difference Time-Domain (FDTD)

The *FDTD* was first introduced in 1966 by Kane S. Yee [91] as Yee' Finite Difference Algorithm [3]. However, the acronym FDTD was coined by A. Taflove in 1980 when he published the first validated FDTD models of sinusoidal steady-state EM wave penetration into a 3D metal cavity [68]. Yee's algorithm was further developed by A. Taflove and others over the subsequent decades leading to the modern version of FDTD as we know today [68, 92–97]. One notable upgrade to FDTD was with the introduction of the Perfectly Matched Layer (PML) in 1994 by Jean-Pierre Berenger [98] for 2D problems, which allowed the simulation of open, infinite domains. Afterwards, the PML was extended to handle 3D problems by D.S. Katz and others [99]. A. Taflove has summarized the evolution of FDTD in his journal in 2022 [100].

The FDTD is a variant of the FDM which is used to solve Maxwell's time-varying field equations [3]. The distinction in the way the mesh, called the Yee's lattice, is generated in FDTD can be seen in Figure 2.3. The figure shows the positions of the field components in a unit cell of the Yee's lattice. Due to its ability to solve EM problems in both space and time domains FDTD is crucial for the study of EM waves. It is used for various applications in the EM field such as designing antenna, waveguides and optical devices, analyzing EMC/EMI, studying EM wave scattering and so on.

2.3.3 Finite Volume Time-Domain (FVTD)

The *Finite Volume Time-Domain (FVTD)* method is a computational simulation technique that was first applied to EM problems in the early 1990s [101–103]. It's a derivation from the FDTD method, and is based on Maxwell's curl equations in their conservative form [103]. The FVTD method solves Maxwell's equations numerically by integration over small elementary volumes. Because there are no limitations for selecting the shape of the elementary volumes, the FVTD is well suited for implementation with unstructured meshes [103]. It has become a powerful alternative to the FDTD method for EM problems where conformal meshing is advantageous.

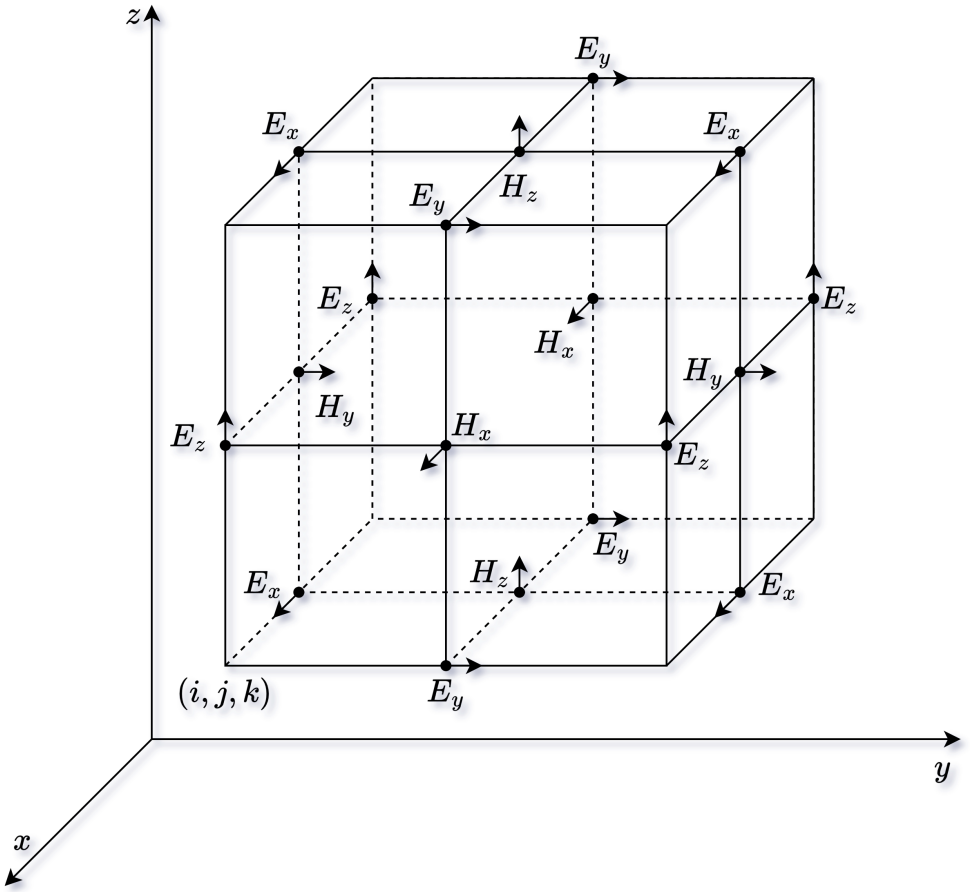

Figure 2.3: Positions of the field components in a unit cell of the Yee's lattice [3].

2.3.4 Method of Moments (MoM)

Although the name *MoM* was first used by R. F. Harrington in the western literature [104], its origin can be traced back to the Russian literature [105, 106]. The details of the MoM's advent and its subsequent evolution are fully documented by R. F. Harrington [107]. However, it was only after the works of Richmond in 1965 [108] and R. F. Harrington in 1967 [109], that the application of MoM in the field of EMs became widespread [3]. A history of its development can be found in [107, 110].

The MoM is a technique used for solving problems related to systems of equations, especially when dealing with integral equations [3]. It involves taking "moments" by multiplying with appropriate weighing functions and integrating, and thus the name. Essentially, it converts integral equations into algebraic equations, which can then be solved numerically. The application of the

MoM essentially involves the following steps [3]:

1. **Integral Equation Formulation:** Most EM problems can be described by integral equations, where the unknown function or distribution appears within integrals. These equations often arise when dealing with fields or quantities that depend on the values over an entire domain.

2. **Moment Representation:** The MoM involves approximating the unknown function or distribution by a set of *basis functions* or a finite set of discrete points (called *collocation points*). These basis functions or collocation points are chosen in a way that simplifies the integral equation.

3. **Approximation of Unknown Function:** The integral equation is then transformed into a system of algebraic equations by applying the chosen approximation. This system of equations relates the coefficients (or values) of the basis functions or collocation points to the values of the integral equation at those points.

4. **Solving for Coefficients:** Solve the resulting system of algebraic equations to determine the coefficients of the basis functions or the values at the collocation points. These coefficients represent the approximate solution to the integral equation.

The obtained coefficients or values can be used to compute other relevant quantities or make predictions about the physical system being studied. Post-processing may include evaluating fields, finding fluxes, or estimating other derived parameters. The MoM is particularly suitable for problems that involve BCs and Interface Conditions (I/FCs), where integral formulations are more natural than differential equations. It's often employed in EM simulations, such as Finite Element Analysis (FEA) of antennas and scattering problems.

2.3.5 Finite Element Method (FEM)

The *FEM* was formally formulated in 1943 by Richard L. Courant [111], the father of Finite Elements (FEs) [112]. Although it was initially developed for the field of structural analysis [2, 3, 111], FEM is widely applied in various fields such as engineering, physics, and computer science. However, it was first applied to solve EM problems by Peter P. Silvester in 1968-1969 [2, 3, 65, 80, 112]. Although

FEM was developed to solve problems in the frequency domain, it expanded to the time domain in the 80s [113]. The detailed history of its evolution can be found in [112, 114].

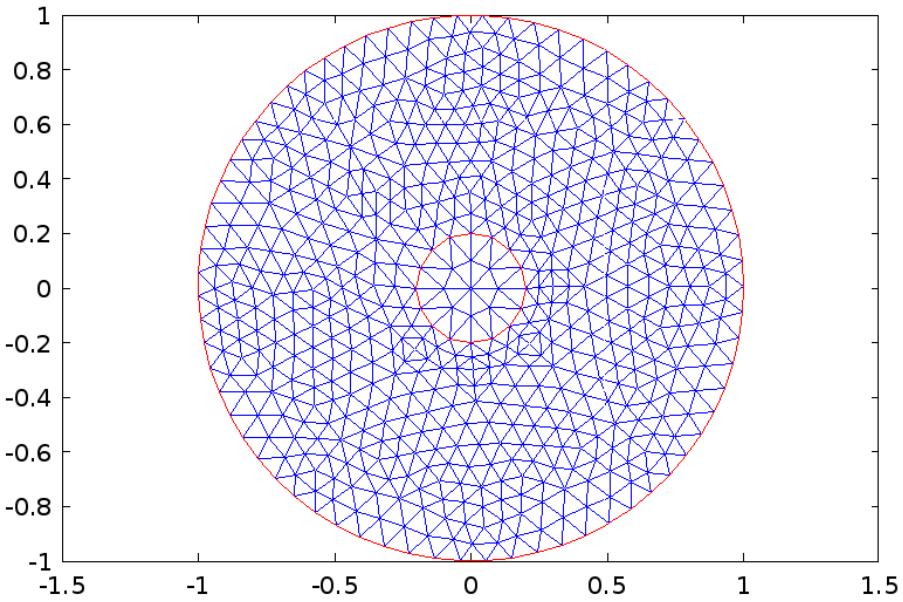

Figure 2.4: Finite Element Method (FEM) mesh, using triangular elements, for a current carrying coil inside a cylindrical domain.

The FEM is a numerical technique used for solving differential equations, particularly PDEs [3]. Similar to the FDM, the FEM segments the solution space into sub-domains called *FEs* [3]. It then uses piecewise polynomials across these elements to approximate the solution of the given PDE [3]. However, unlike that in the FDM, the grid of elements is non-uniform and unstructured in the FEM, which can be observed in Figure 2.4. The steps involved are formalized below [115]:

1. **Discretization:** The first step in the FEM is to discretize the domain of interest into smaller, finite-sized subdomains called *elements*. These elements are typically simple shapes like triangles or quadrilaterals in 2D, or tetrahedra or hexahedra in 3D. The domain is broken down into a mesh composed of these elements.

2. **Approximation of Field Variables:** In FEM, the field variables (such as displacement, temperature, stress, etc.) are approximated within each element using piecewise interpolation functions known as *element shape functions* [3]. These shape functions describe how the variable varies within each element and are typically chosen to be continuous within each

element but discontinuous between adjacent elements. The differential equation describing the physical problem is transformed into a system of algebraic equations by integrating it over each element using the approximated field variables. This process results in a set of linear equations for each element, which are then arranged in a matrix form known as the *element coefficient matrix* or *stiffness matrix* [3].

3. **Assembling:** The next step involves assembling the equations from all the elements into a global system of equations, leading to *global coefficient matrix* [3]. This process involves accounting for the connectivity between elements and properly combining the contributions from each element into the global system. BCs are applied to the global system to incorporate constraints or known values at specific locations or on certain boundaries of the domain. These conditions are essential for obtaining a unique solution.

4. **Solving the Algebraic System:** The global system of equations, which includes both the differential equation and BCs, is solved numerically using methods like direct solvers (e.g., Gaussian elimination) or iterative solvers (e.g., conjugate gradient method). The solution provides the values of the field variable at discrete points within the domain.

The FEM is highly versatile and can handle complex geometries and material properties [3]. The accuracy of the FEM solution can be controlled by adjusting the mesh density (refining or coarsening the mesh) and using higher-order shape functions. However, it can be computationally intensive for large-scale problems and requires careful consideration of various factors, including element types, mesh quality, and solver choices.

2.4 Electromagnetic (EM) Actuators

The primary objective of this research is to evaluate the performance of both a data-driven and a hybrid approach for solving practical EM problems, as mentioned in Section 1.3. Hence, I have chosen two problems of varying complexities for this work — a coil and a C-core actuator, discussed in Sections 3.2.1 and 3.2.2, respectively.

An actuator is a component of a device that serves the general purpose of controlling mechanical movements within the device. Essentially, it converts a given form of input energy, such as electrical,

hydraulic, pneumatic, thermal, mechanical or human power, into some form of physical motion [116]. To initiate any movement, an actuator requires a feedback through a control signal from a control system. There are various kinds of actuators that produce varying motions, and use different power sources for the control signal. In fact, they are classified based on the type of the input energy they use — electrical, mechanical, hydraulic, pneumatic, etc., and the type of motion they produce — linear or rotary. Once it has power, an actuator creates specific motions depending on the purpose of the machine. For example, in the JWST, actuators are used to align the 18 hexagonal mirror segments to serve as one big primary mirror [22, 24]. Their applications can be found in various industries such as aerospace, automotive, healthcare, gaming, and many more [117–120].

An EM actuator is a device that converts electrical energy into mechanical motion and vice versa, using the interaction of magnetic fields [121, 122]. The basic principle of an EM actuator is based on Maxwell's equations, specifically Faraday's law, Ampere's law, as discussed in Sections 2.1.2 and 2.1.2, respectively, Lorentz force of EM forces, and Biot-Savart's law [121]. The EM actuator consists of two main components: a coil and an armature. The coil is a loop or a solenoid of wire that carries an electric current. The armature is a movable part that is attached to a spring or a lever. The armature can be made of iron or another ferromagnetic material, or it can be another coil or even a permanent magnet. When an electric current flows through the coil it creates a magnetic field around it. This magnetic field attracts or repels the armature, depending on its polarity and orientation. The armature moves towards or away from the coil, creating a mechanical motion. This motion can be controlled by moderating the current, voltage, frequency, or waveform of the electric signal applied to the coil.

2.4.1 EM Analysis of a C-core Actuator

An EM C-core actuator is a type of a linear actuator that uses a C-shaped stator (C-core) and a moving component (armature) [123, 124]. A 2D version of such an actuator is shown in Figure 2.5, as seen in [125]. From the figure, we can observe that the C-core is wrapped around by a copper coil with N turns, and the armature is placed close to the C-core without making any physical contact. Current, I is passed through the coil which generates a magnetic field around it. This magnetic field gets concentrated in the ferromagnetic C-core (made of iron), turning it into a magnet. This causes the C-core to attract the armature towards it, resulting in a linear motion.

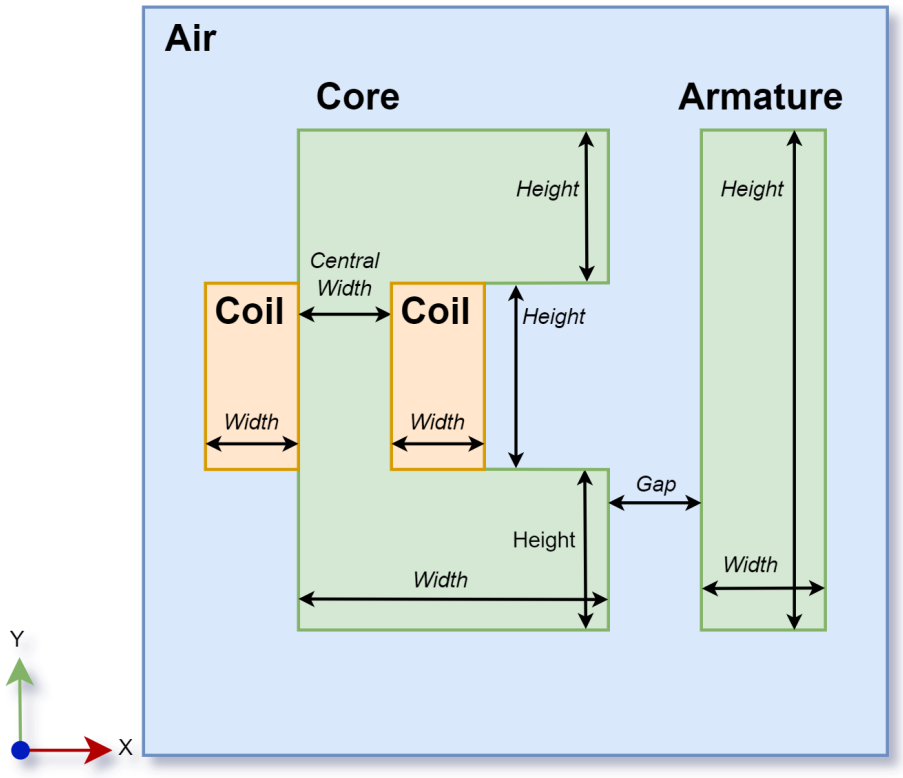

Figure 2.5: An Electromagnetic (EM) C-core actuator

For this EM problem, we can assume the magnetic materials to be linear, and flux leakage and fringing are negligible. To solve it analytically, we begin with the generalized Ampere's law, as explained in Section 2.1.2, which gives us Equation 2.23:

$$NI = H_c l_c + 2 * H_g l_g + H_a l_a, \tag{2.23}$$

where N and I are the number of wire turns and the current in the coil, respectively, H_c, H_g, and H_a are the magnetic field intensities, and l_c, l_g, and l_a are the magnetic flux path lengths in the C-core, air gap, and armature, respectively. Here, the product NI is called the magnetomotive force (MMF). Furthermore, we can find the relation between the total magnetic flux, ϕ and the MMF from Equation 2.24 [58]:

$$\phi = BS = \mu HS = \frac{\mu NIS}{l} = \frac{NI}{R}, \tag{2.24}$$

where R is the total reluctance of the magnetic circuit. Using this we can derive the equivalent

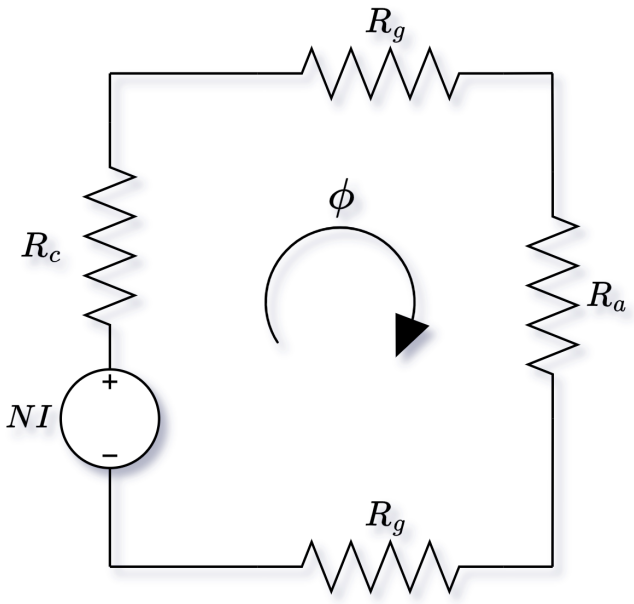

Figure 2.6: The equivalent electric circuit for the C-core problem.

electric circuit for the given magnetic circuit problem, as shown in Figure 2.6. From the circuit, we can observe that R is given by Equation 2.25:

$$R = R_c + 2 * R_g + R_a, \tag{2.25}$$

where R_c, R_g, and R_a are the reluctances of the C-core, air gap, and armature, respectively. They are given by Equations 2.26, 2.27, and 2.28, respectively:

$$R_c = \frac{l_c}{\mu_c S_c}, \tag{2.26}$$

$$R_g = \frac{l_g}{\mu_g S_g}, \tag{2.27}$$

$$R_a = \frac{l_a}{\mu_a S_a}, \tag{2.28}$$

where μ_c, μ_g, and μ_a are the magnetic permeabilities, and S_c, S_g, and S_a are the cross-sectional surface areas of the C-core, air gap, and armature, respectively. Using the values of these parameters we can compute R, and from Equation 2.24, we can find ϕ using R and NI. Finally, we can

compute the magnetic flux densities in the C-core (B_c), air gap (B_g), and armature (B_a) using Equations 2.29, 2.30, and 2.31, respectively:

$$B_c = \frac{\phi}{\mu_c S_c} \tag{2.29}$$

$$B_g = \frac{\phi}{\mu_g S_g} \tag{2.30}$$

$$B_a = \frac{\phi}{\mu_a S_a} \tag{2.31}$$

Furthermore, we can find the energy stored, W in the magnetic field using Equation 2.32 [58]:

$$W = \int_V \frac{B_g^2}{2\mu_o} dv = \frac{B_g^2}{2\mu_o} \cdot 2 S_g l_g \tag{2.32}$$

Moreover, we can compute the magnetic force, F on the armature using Equation 2.33 [58]:

$$F = \frac{dW}{dx} = \frac{\frac{B_g^2}{\mu_o} \cdot S_g dx}{dx} = \frac{B_g^2 S_g}{\mu_o}, \tag{2.33}$$

where F is the force in $Newtons$, dW is the change energy in $Joules$, and dx is the change in distance between the C-core and the armature in $meters$. This formula assumes that the actuator operates in the linear region of the magnetic circuit, where the magnetic flux density is proportional to the current. If the actuator operates in the saturation region, where the magnetic flux density reaches a maximum value, then the force will be reduced. To avoid saturation, the actuator should have a large air gap and a low current. Another factor that affects the force of a C-core actuator is the detent force, which is the residual force caused by the attraction between the permanent magnet (used as the armature) and the stator (C-core) [126]. The detent force can reduce the efficiency and accuracy of the actuator. To minimize the detent force, some design techniques can be used, such as adding auxiliary slots or arc-teeth to the stator [126].

In this work, I have tried to solve for the magnetic field distribution, B in a current carrying coil problem, discussed in Section 3.2.1, and in a C-core actuator problem, as illustrated above and further discussed in Section 3.2.2, as proof-of-concepts (POCs) for my methodologies.

2.5 Machine Learning (ML)

The word "*Artificial Intelligence (AI)*" was first coined by John McCarthy, who is considered the father of AI, in 1956 [127]. He won the Turing Award in 1971 for his contributions to the research of AI [128]. AI refers to the study of simulation of human cognition in machines. John McCarthy defined AI as follows [129]:

"*It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.*"

He further elaborated on what he meant by intelligence [129]:

"*Intelligence is the computational part of the ability to achieve goals in the world.*"

Machine Learning (ML) is a subset of AI, as shown in Figure 2.7, which refers to the science and art of developing computer algorithms that can learn from past experience, i.e., data, without explicit programming or human assistance. It was first defined by AI pioneer Arthur Samuel in 1959 as follows [130, 131]:

"*[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.*"

Decades later, in 1997, Tom M. Mitchell provided a more technical definition of ML, given as follows [130–132]:

"*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.*"

Currently, ML is at the forefront of the tech industry: powering all the high-tech products such as smartphones, laptops; auto-piloting cars; ranking web searches; suggesting recommendations for movies and music; recognizing people's faces; completing sentences; determining credit scores; predicting stock markets; and doing much more. The first worldwide mainstream ML application was the spam filter, introduced in the 1990s [131]. The spam filter learned from the labelled data (emails marked as spam by users, and other regular emails) to accurately flag any new spam email without any explicitly defined rules.

ML's burgeoning application in different domains can be attributed to its ability to formulate

the underlying relationships between large sets of data with their respective target outcomes. Computationally, ML tries to learn a generic hypothesis/function that can map some input data to its corresponding output data. This function is not well-defined, and ML learns it from the training data through calculating the errors. As a result, ML can generalize any hypothesis and make precise predictions using new unseen data. Furthermore, ML has the inherent characteristics to evolve along with new data and capture the changes in the process. This adaptability to disruptions makes it extremely versatile. Thus, ML can be applied to domains ranging from simple spam filtering to voice recognition to bio-informatics.

2.5.1 Classification of ML

ML models can be classified in more than one way. For instance, if the model is trained through human supervision or not, if it can be trained on instances on the fly or on mini-batches, if the model makes predictions based on comparison of the new data with the training ones or if it learns a predictive function, etc. However, ML models are broadly categorized, based on the method of training, in the following ways:

I **Supervised Learning** — The training set consists of input data and the corresponding labels (target outcomes). The model tries to learn a function mapping the inputs to the respective outputs.

II **Unsupervised Learning** — The training dataset is not labelled and the target outcome is unknown. The model tries to learn hidden patterns within the input data.

III **Semi-Supervised Learning** — The training set contains partially labelled data. Usually a small portion of the dataset contains a target value, and the larger portion is unlabelled. The model uses a combination of both supervised and unsupervised learning during the training process.

IV **Reinforcement Learning (RL)** — The model is called an *agent*, which tries to learn a sequence of *actions* that will earn it the highest *reward* while performing within an *environment*. The *agent* (model) takes the *observations* from the *environment* as its input and updates its *policy* of choosing the best *action* at any given point.

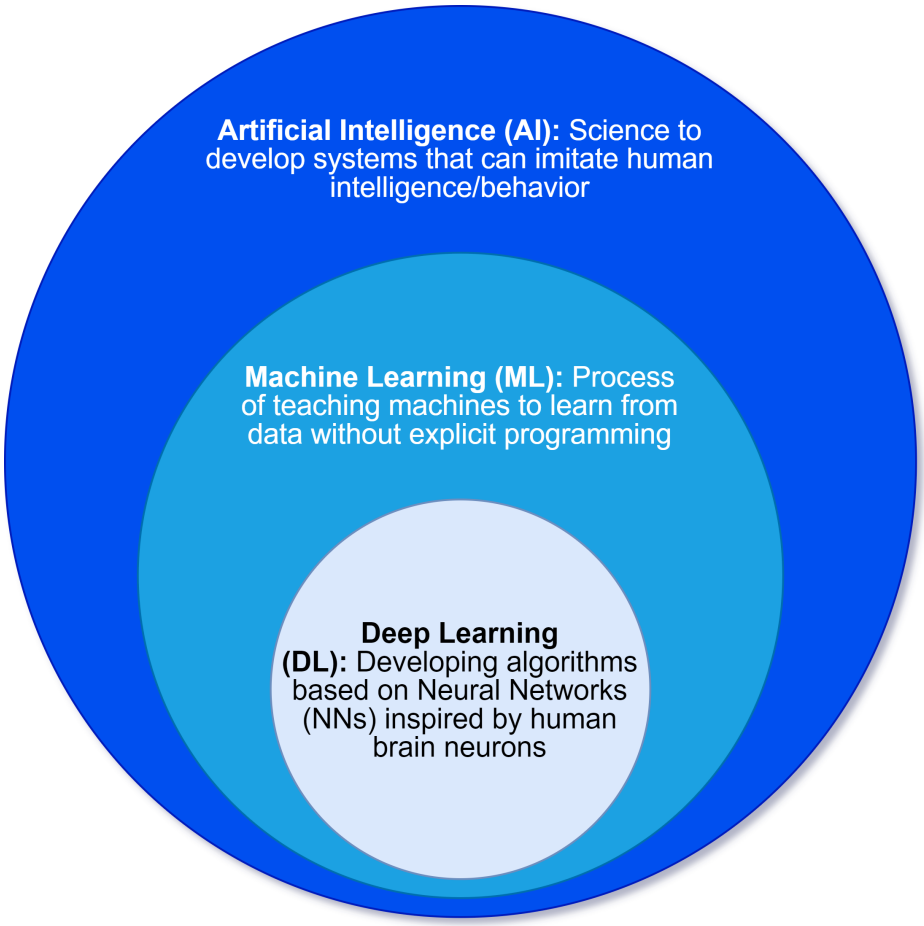

Figure 2.7: Artificial Intelligence (AI) vs Machine Learning (ML) vs Deep Learning (DL)

2.5.2 Origin of ML

Even though ML might seem to have emerged in the past decade or so, its origin dates back to as early as the 1940s. The first ever theoretical ML model conceptualized was a Neural Network (NN), discussed in Section 2.6. Published in 1943, it was a mathematical modelling of human brain cells (neurons), used to create algorithms mimicking human thought process [133]. Few years later, in 1949, Donald Hebb introduced theories of how neurons in the brain interact with one another and what excites them in his book *"The Organization of Behavior: A Neuropsychological Theory"* [134], which went on to become one of the crucial components in ML development. The very next year, in 1950, Alan Mathison Turing, one of the founding fathers of AI and the father of computer science, published the seminal paper "Turing Test", also known as "The Imitation Game", which laid out the basis to determine the intelligence of a machine [135]. He introduced the construct that if a

machine can imitate a human closely enough that it can't be differentiated one from the other, then the machine is considered intelligent. In 1952, computer scientist Arthur Samuel developed the first ever ML program [136]. The program was written for an IBM computer to play the game of checkers, and learn to improve every time it played. From this simple program of playing checkers, ML has come a long way, in the form of AlphaGo, to beating the world champion in the sophisticated game of GO in 2016 [137].

2.5.3 Deep Learning (DL)

Over the course of time, various types of ML models were introduced such as the Nearest Neighbour Algorithm, Support Vector Machine (SVM), etc., but they still lacked the true essence of human intelligence. Eventually, in order to solve more complex problems, scientists decided to keep on stacking multiple layers of Artificial Neurons (ANs), discussed in Section 2.6.1, to evolve a Neural Network (NN) into a Deep Neural Network (DNN). This led to a new subset of ML known as *Deep Learning (DL)*, as shown in Figure 2.7. DL refers to the field studying DNNs, essentially models with a consortium of multiple ("deep") computational layers. DNNs are structured in a hierarchical manner where a layer at a higher level learns from the layer below it, representing information from the dataset in multiple levels of abstraction. One of the first DNNs was the "Neocognitron", introduced by Fukushima in 1980, which could recognize patterns in images [138]. This was further extended by the A.M. Turing Award Laureate [139], Yann LeCun and his team in 1998 [140], in the form of $LeNet-5$, a pioneering 7-level Convolutional Neural Network (CNN), discussed in Section 2.7, which was used for handwriting recognition. However, with increasing number of layers in the NNs, the demand for computational resources to successfully train such models also kept rising. Due to the technological limitations during the 1990s, it was impossible to train DNNs causing a decline in research interest, and hence, turning DL into a lost art [131].

However, it wasn't until 2006, when another A.M. Turing Award Laureate [141], Geoffrey Hinton actually coined the term "Deep Learning (DL)" and branded this technique [130, 131]. In their paper [142], Geoffrey Hinton, Simon Osindero and Yee-Whye Teh were able to train a DNN capable of recognizing handwritten digits with state-of-the-art precision (>98%). Their success attracted new attention to the field and gave momentum to the study of DL. Furthermore, the technological development in the computation hardware and software architectures such as TensorFlow [143],

Theano [144] and PyTorch [145] has further facilitated the research progress. This enthusiasm led to the exploration of newer fields such as Computer Vision [146], Natural Language Processing (NLP) [147], Financial Forecasting [148], etc. The key reasons leading to the DL Tsunami can be listed as follows:

- **Availability of Sufficient Data** — real or fabricated.
- **Enhanced Hardware Capacity** — Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), Solid-State Drives (SSDs).
- **Easy-to-use Software Libraries and Framework** — Scikit-Learn [149], Pandas [150], TensorFlow [143], PyTorch [145].
- **Affordable Cloud Computing Services** — Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure, Git.
- **Advanced DL Architectures** — Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Autoencoders (AE), Generative Adversarial Network (GAN).

With such factors coming into play [151, 152], the number of units (neurons) in NNs has doubled in every 2-3 years [153].

Now, provided this surge in research towards DL, one would come up with the indispensable question of why it is so popular over the traditional ML algorithms. The answer is twofold: DL can solve complex problems much better than the latter; and it doesn't require feature engineering. Complex problems tend to have a higher number of features (input variables). Simple NNs or other ML algorithms can't always extract the relevant information from them. Whereas in DNNs, the hidden layers learn different features incrementally in a hierarchical manner. For instance, if we are dealing with a facial recognition problem, the first layer would learn to detect edges, the next would use the edges to learn different shapes like eyes, nose, etc., which could be used by the next layer to identify the face. Removal of a single layer can be detrimental to such a model's performance [154]. On the other hand, DNNs don't require explicit feature extraction by a domain expert. They solve problems in an end-to-end method, whereas traditional ML algorithms require complex problems to be broken down into simpler tasks. This comes as a mixed blessing due to the requirement of a

larger dataset. Even if the "Big Data Era" of technology makes it easier to acquire a larger dataset, the computation resource and time required for DL is still higher than other ML algorithms. But, provided the proper infrastructure, a DNN will almost always outperform its contemporaries.

2.6 Neural Networks (NNs)

Throughout human history, we have been inspired by nature for numerous inventions. As it turns out, the *Neural Network (NN)*, also known as the *Artificial Neural Network (ANN)*, is just one of such examples. In their paper, published in 1943 [133], neurophysiologist Warren McCulloch and mathematician Walter Pitts proposed the first computational NN architecture in an attempt to model the Biological Neural Network (BNN) in human brains. Their work laid the pathway for the development of many new NN/ANN architectures in the following decades.

In 1957, psychologist Frank Rosenblatt combined the mathematical model proposed by Warren McCulloch and Walter Pitts [133], and the work of Arthur Samuel on his ML model [136], and came up with the first computer trainable ANN called the *Perceptron* [155, 156]. The Perceptron was trained using an algorithm inspired by Hebb's rule [130, 131]. Donald Hebb proposed the theory that the connections between biological neurons become stronger whenever they are triggered frequently [134]. Siegrid Lowel reiterated the hypothesis with the phrase [131]:

"*Cells that fire together, wire together.*"

The Perceptron became the basis for all the modern versions of NN architectures that exist today.

2.6.1 Artificial Neurons (ANs)

NNs form the very core of DL. NNs are capable of learning and retaining knowledge, and using it to make predictions in the future [157]. All the DL models that have come into existence have been based on the principles of a basic NN paradigm. In order to delve deep into how an NN functions we need to examine its building block, a *neuron* (also known as an *Artificial Neuron (AN)*). An AN, a much simpler version of its biological counterpart, functions as follows:

- Takes in an input X with m features, $X = [1, X_1, X_2, ..., X_m]$
- Multiplies them with their respective weights, $W = [W_0, W_1, W_2, ..., W_m]$

- Sums the product of inputs and weights, $\sum_{j=0}^{m} X_j W_j$
- Applies a function σ to that sum and gets the output, $\hat{Y} = \sigma(\sum_{j=0}^{m} X_j W_j)$

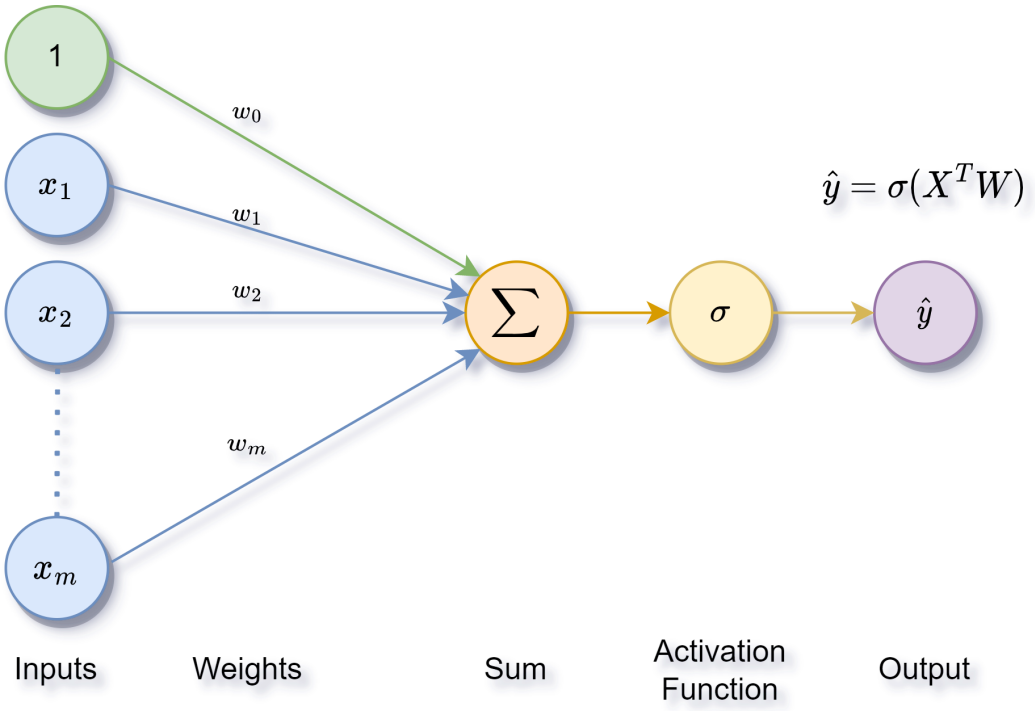

Figure 2.8: A basic Artificial Neuron (AN)

This can be better visualized in Figure 2.8. Each of the *input features*, X_i is multiplied with an associated *weight*, W_i. These weights determine the significance of the different features while generating the *output*, $\hat{Y}$. Furthermore, the input X is augmented with a constant "1", which is called a *bias*. It enhances the neuron's flexibility to learn in the solution space by allowing movements away from the origin. The function σ is called the *activation function*. In the same way that a signal has to cross a certain threshold to pass through a biological neuron, the activation function imposes a similar threshold for ANs. There are several activation functions that can be used in an NN, with the popular ones being $ReLU$, $Sigmoid$ and $TanH$ [131, 153]. Once we have the output $\hat{Y}$, we compare it to the actual value Y, which is known as the *target/label*. We compute the difference (also referred to as a *loss* or an *error*), such as the Mean Squared Error (MSE) as shown in Equation 2.34:

$$E = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.34}$$

The aim is to minimize such an error, and it's achieved by updating the weights. This process is referred to as *training* a neuron. The usual process to adjust the weights involves gradient-based methods such as *gradient descent*. The update rule for a single neuron would be as in Equation 2.35:

$$w_j^{(t)} = w_j^{(t-1)} + \eta\left(-\frac{\partial E}{\partial w_j}\right), \tag{2.35}$$

where η is the *learning rate* of the algorithm. This type of training/learning where a target/label is used to compare the output of a model is known as *supervised learning*, as mentioned in Section 2.5.1 and further discussed in Section 2.6.3.

2.6.2 Fully Connected (FC)/Dense Layer

A simple neuron, as in Figure 2.8, can be used to solve a linearly separable problem with an arbitrary accuracy. In order to tackle complex problems we need a more sophisticated model. Multiple such neurons are stacked together in a single layer, which are then repeated a few times to get what we know as an NN. Figure 2.9 shows an example of a basic NN. It has three layers — *input*, *hidden* and *output*. The input layer has no weights associated with it, it simply consists of pass-through neurons that output the input features to the next layer as it is [131]. It consists of neurons equal to the number of input features plus a bias neuron (always outputs "1") [131]. The hidden and output layers have associated weights for each neuron in the layer, and a single activation function for the entire layer. Neurons in the same layer are completely disconnected from one another. Both the hidden and output layers are "fully-connected" with their adjacent layer(s). Hence, they are called *Fully Connected (FC) layers* (also known as *dense layers*). This is the most basic type of layer in an NN.

The number of hidden layers and number of neurons in each hidden layer are both *hyperparameters*. In ML, a hyperparameter is a parameter that is used to control the learning process of a model. The number of output neurons is problem specific. This basic NN is also called a Feedforward Neural Network (FNN), since it allows signals/information to propagate through all its layers without any feedback [158]. An FNN with a single hidden layer is capable of approximating any continuous

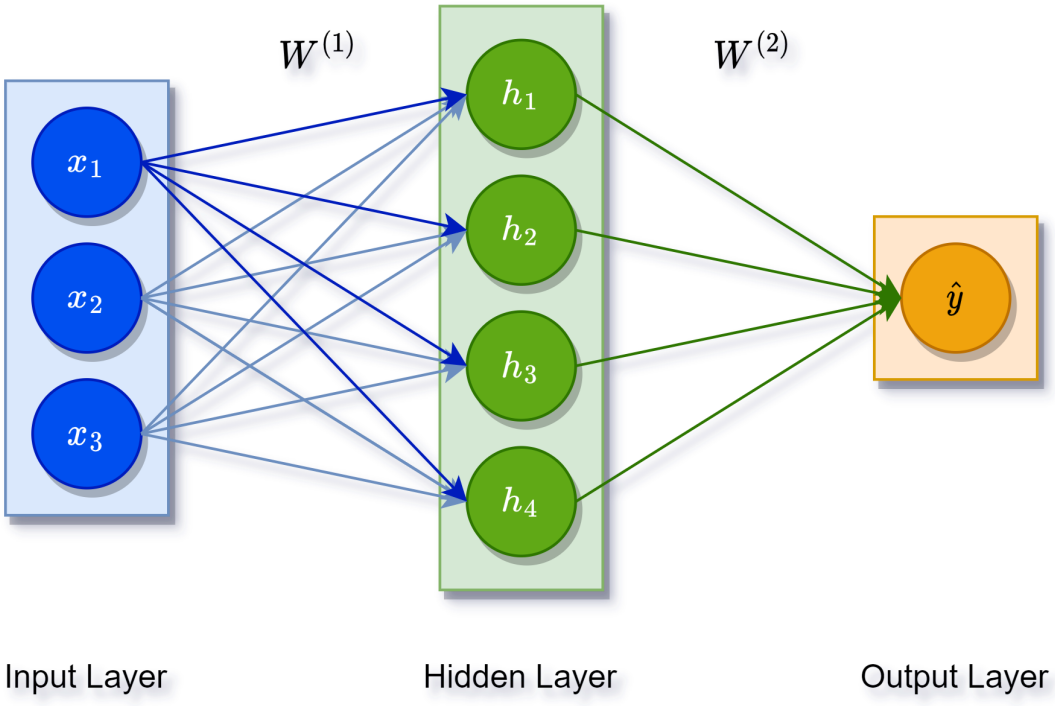

Figure 2.9: A basic Neural Network (NN) with an input and an output layer, and one hidden layer.

function with arbitrary precision [159]. It is this ability of universal function approximation that has led to the popularity of NN across a wide variety of applications.

2.6.3 Training NNs

One of the earliest and biggest hurdles of NNs was to find a suitable and stable training method. It wasn't until 1986, when David Rumelhart, Geoffrey Hinton, and Ronald Williams introduced the *backpropagation* algorithm to train NNs [160]. This was a landmark moment for DL as the process is still currently used to train all NNs. In their seminal paper, they proposed an efficient technique to compute the gradients of the error with respect to each of the weights in the entire NN in just two passes (one forward, one backward) [160]. The weights are then updated using a gradient-based method, often with "adaptive" *optimization algorithms*, such as *Adam* [161]. This whole training process is repeated until one of the following criteria is met: a certain number of runs (*epochs*) is reached; the error goes below a certain acceptable value; or when the model starts to *overfit*.

Overfitting, as the name suggests, occurs when a model has learned to predict the training data so well that it cannot generalize anymore. Normally, it arises when a complex model trains

on a small dataset, and it ends up memorizing all the data. The most common way of detecting overfitting is to use two separate sets of data — *training* and *validation* [162]. When the training error keeps on dropping while the validation error starts to rise, that's when the model starts to overfit. There are multiple ways to tackle this: increase the data size; reduce the model's complexity; or add a form of *regularization*. The last one basically penalizes the model's parameters that tend to increase drastically during the training process. A few forms of regularization techniques have been discussed later in Sections 2.6.4, 2.6.5 and 2.6.6.

As mentioned in Section 2.6.1, NNs are trained using a supervised learning technique. Such a training process involves a dataset containing some input variables (also called features or attributes) and some output variables (also called labels or targets). The NN is fed the input variables to predict the output, which is then compared with the actual target. The error calculated is used to train the NN through backpropagation, as mentioned above. Supervised learning can be classified into two categories — *regression* and *classification*. A regression algorithm performs the task of predicting a continuous value from the real number domain ($\mathbb{R}$) as an output based on the input data. It tries to learn a function that can simply map the input features to the target values in the dataset. On the other hand, a classification algorithm tries to map the input samples to some predefined categories of targets known as labels or classes.

2.6.4 Dropout

One of the most popular forms of regularization in DL is *dropout*. It was first introduced in 2012 by Geoffrey Hinton [163] and further elaborated in 2014 by Nitish Srivastava et al. [164]. It can enhance the performance of even the state-of-the-art DL architectures by 1-2% [131]. A dropout layer consists of a parameter p, which is called the *dropout rate*. During each training step, each of the neurons in a given layer has the probability, p of being "dropped out". This means that the neuron will be completely ignored or turned off during that particular training step. Once the training is completed, the neurons are not dropped anymore. Essentially, the dropout layer creates a collection of 2^N different NNs, where N is the number of neurons available for dropout. As a result, the chance of a certain combination of neurons being trained twice is almost impossible. The final trained network will represent an averaging ensemble of all the different NNs that have been trained with extensive parameter (weight) sharing.

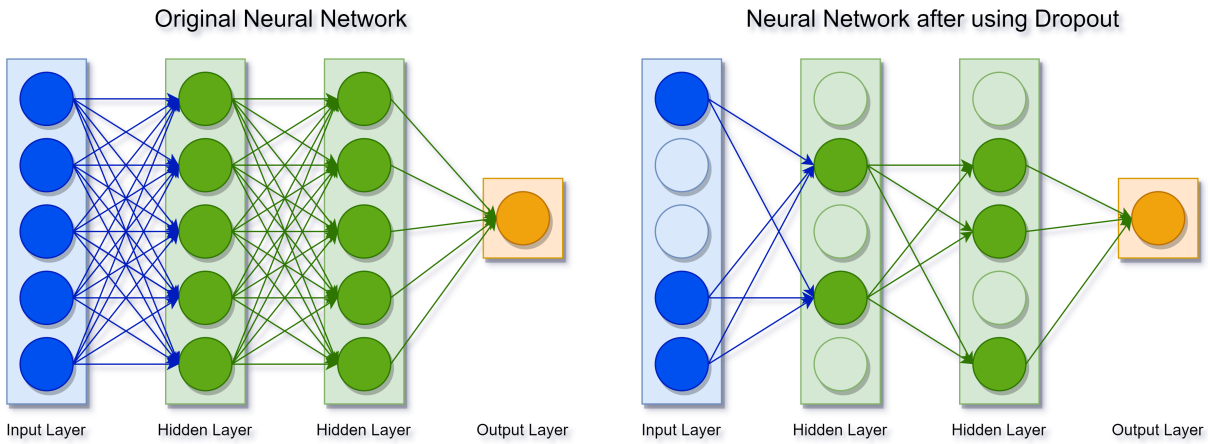

Figure 2.10: A Neural Network (NN) before and after dropout is applied.

2.6.5 Batch Normalization (BN)

Another famous regularization technique is the *Batch Normalization (BN)*. Normalization and standardization are some of the established and popular preprocessing techniques that are commonly used in both Statistical ML, and DL. This idea of scaling the inputs was further extended by Sergey Ioffe and Christian Szegedy in their paper in 2015 in the form of BN [165]. The BN operation basically applies the standardization technique to the inputs of any given layer in an NN. Sergey Ioffe and Christian Szegedy demonstrated that BN not only drastically reduces the training time for all the DNNs tested, but also improves their performance significantly [165]. Furthermore, it solves the problems of vanishing gradients and overfitting during the training process.

BN simply zero-centers and normalizes each input, and then scales and shifts it before passing onto the next layer. The standardization is done using the mean and standard deviation, calculated from the current mini-batch the model is being trained on, as shown in Figure 2.11. Whereas, the scaling and shifting is done using two learnable parameters, respectively. Essentially, the operation allows the model to find the optimal distribution for each of the inputs for the layer. The entire process can be summarized with the following steps:

I Calculate the mean, μ_B for the i^{th} input x_i in the mini-batch B, given by Equation 2.36:

$$\mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} x_i, \tag{2.36}$$

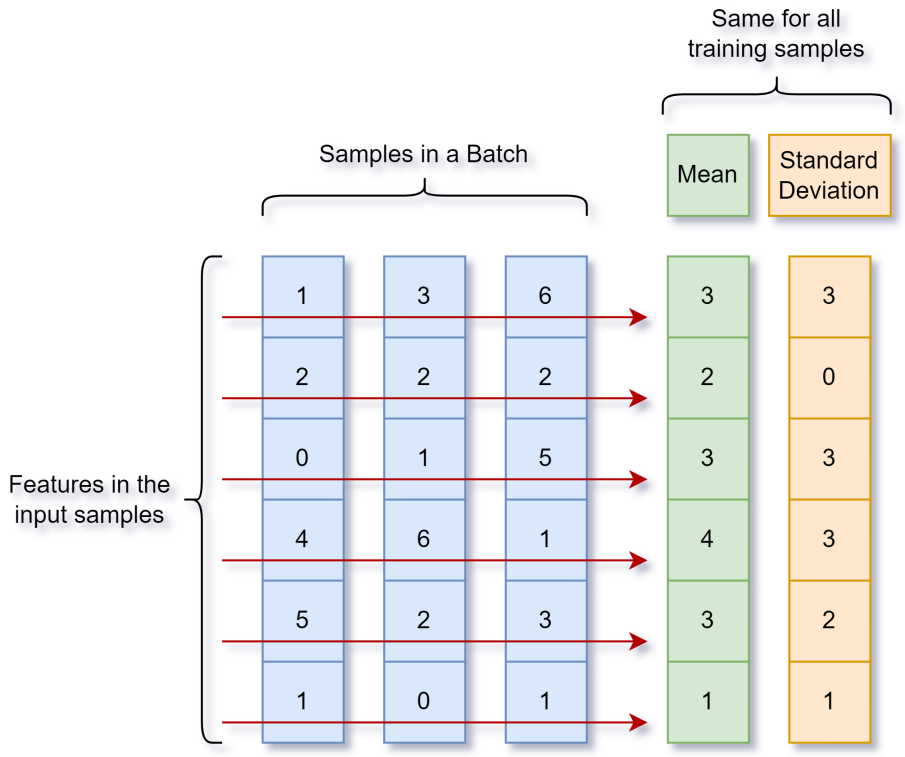

Figure 2.11: Computing the mean and the standard deviation for Batch Normalization (BN).

where m_B is the number of samples in the mini-batch B.

II Calculate the standard deviation, σ_B for the i^{th} input x_i in the mini-batch B, given by Equation 2.37:

$$\sigma_B{}^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (x_i - \mu_B)^2 \tag{2.37}$$

III Compute the zero-centered and normalized value, $\hat{x}_i$, given by Equation 2.38:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B{}^2 + \epsilon}}, \tag{2.38}$$

where ϵ ($\sim 10^{-5}$) is called a smoothing term, which avoids division by zero.

IV Compute the scaled and shifted output (final) value, z_i, given by Equation 2.39:

$$z_i = \gamma_i \cdot \hat{x_i} + \beta_i, \tag{2.39}$$

where z_i is the output of the BN operation, γ_i is the learnable scaling parameter and β_i is the learnable offset parameter.

During the training process, while the BN process computes the mean and standard deviation for the mini-batches it also keeps track of the mean, μ and the standard deviation, σ for the entire training set using an exponential moving average. These are then used after the training is complete to make predictions, for instance, while using the test set.

2.6.6 Layer Normalization (LN)

Even though BN is a well-performing regularization technique, it doesn't work well on sequential data. In their paper, in 2015, César Laurent et al. showed that BN hardly improved the performance of Recurrent Neural Networks (RNNs), a type of NN which can train on sequential data, discussed in Section 3.3 [166]. In order to overcome this issue, Jimmy Lei Ba et al. introduced a new form of regularization technique called the *Layer Normalization (LN)* in their paper in 2016 [167]. LN and BN function similarly, except that LN performs the normalization across all the features of every sample instead of all the samples in a mini-batch, as shown in Figure 2.12. The operation steps are almost the same as those in BN, given as follows:

I Calculate the mean, μ_L for the i^{th} feature x_i of the input sample, given by Equation 2.40:

$$\mu_L = \frac{1}{m_F} \sum_{i=1}^{m_F} x_i, \tag{2.40}$$

where m_F is the number of features of the input.

II Calculate the standard deviation, σ_L for the i^{th} feature x_i of the input sample, given by Equation 2.41:

$${\sigma_L}^2 = \frac{1}{m_F} \sum_{i=1}^{m_F} (x_i - \mu_L)^2 \tag{2.41}$$

III Compute the zero-centered and normalized value, $\hat{x_i}$, given by Equation 2.42:

$$\hat{x_i} = \frac{x_i - \mu_L}{\sqrt{{\sigma_L}^2 + \epsilon}}, \tag{2.42}$$

where ϵ ($\sim 10^{-5}$) is called a smoothing term, which avoids division by zero.

IV Compute the scaled and shifted output (final) value, z_i, given by Equation 2.43:

$$z_i = \gamma_i \cdot \hat{x_i} + \beta_i, \tag{2.43}$$

where z_i is the output of the LN operation, γ_i is the learnable scaling parameter and β_i is the learnable offset parameter.

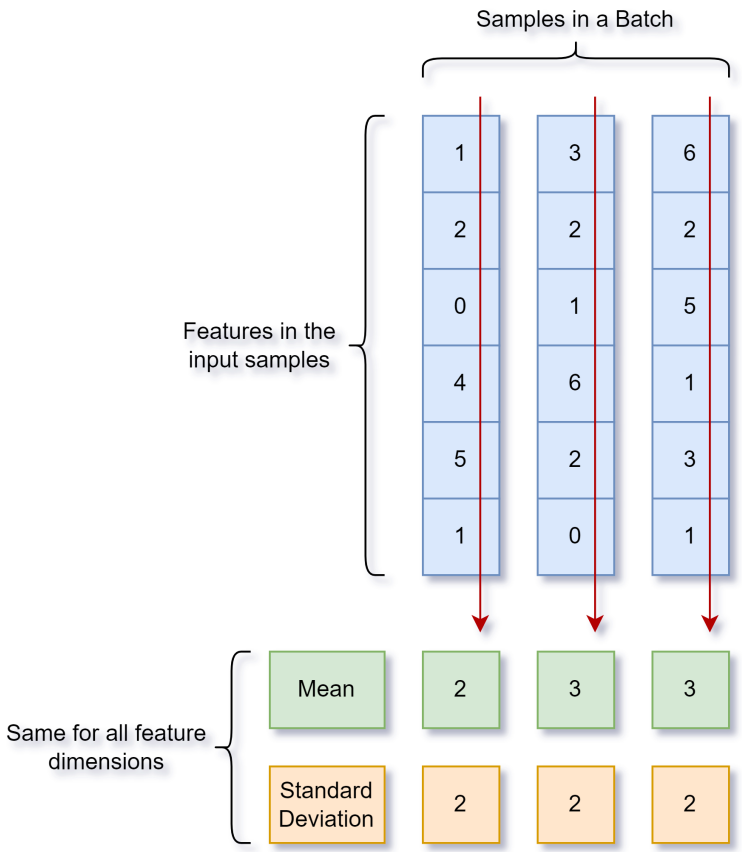

Figure 2.12: Computing the mean and the standard deviation for Layer Normalization (LN).

Unlike BN, LN can compute the statistical values independently for each input, i.e., it doesn't have to rely on the mini-batches. This further indicates that the computation for the training and testing phase is the same. Hence, it doesn't require to keep track of the mean and standard deviation of the entire training set using an exponential moving average like BN.

2.7 Convolutional Neural Networks (CNNs)

Nobel Prize laureates David H. Hubel and Torsten Wiesel unravelled some of the sophisticated mechanisms in which the visual system of vertebrates processes information [168]. In their papers in 1958 [169] and 1959 [170], as they experimented on cats, and later, in 1968, on monkeys [171], they gained some key insights of how the brain's visual cortex is structured. Specifically, they observed that the neurons in the visual cortex responded to only a limited region of the entire visual field, leading to the idea that they have a small local receptive field. Furthermore, some neurons got stimulated by only a certain fixed pattern such as a horizontal line or a vertical line. Others with relatively larger receptive fields reacted to even more complex patterns. These led to the conceptualization of a hierarchical structure of the neurons, where some high-level neurons take the output from the ones in their respective neighbouring lower-level, which can process any form of complicated pattern in the visual field. Inspired by these findings, Kunihiko Fukushima came up with a novel architecture of NN in 1980 called the "Neocognitron", which could recognize patterns regardless of their positions in the visual field [138]. The Neocognitron was the first iteration of what is known today as the *Convolutional Neural Network (CNN)*. Efforts on CNNs continued over the next decades, and they bore fruit in the form of the famous $LeNet-5$, a pioneering 7-level CNN introduced by Yann LeCun et al. in 1998 [140]. It was widely used for handwritten digit recognition on checks by the banks.

CNNs consist of multiple layers of neurons just like the regular NNs, discussed in Section 2.6. However, the key factor that differentiates the former from the latter is the type of layers used to build the network. A basic NN is simply made up of FC layers, as mentioned in Section 2.6.2, whereas a CNN requires some additional types along with the FC layers — convolutional layers and pooling layers, which are discussed in Sections 2.7.1 and 2.7.2, respectively.

A CNN functions fundamentally in the same way as a simple NN, discussed in Section 2.6.1. The data is fed into the network through the input layer (which has no weights). As the data passes through each hidden layer it gets multiplied by the respective weights, summed up, and a non-linear function applied to it. However, a CNN expects the input data to be arranged in a certain shape, either a 2D or 3D grid format. It was essentially based on the digital image format, which was the primary target data type for CNNs. Nevertheless, modern CNNs are capable of processing not just

images, but any data type that can be structured accordingly.

2.7.1 Convolutional Layer

The core of a CNN is the *convolutional layer*. It performs the primary feature extraction for the CNN. Unlike a dense layer, discussed in Section 2.6.2, a convolutional layer doesn't have all its neurons connected to all the neurons in the previous layer. Inspired by the structure of the brain's visual cortex, a convolutional layer consists of small local receptive fields known as *kernels/filters*, as shown in Figure 2.13. These kernels are essentially 2D matrices with an associated weight for each position in the matrices. Each kernel is slid across the 2D input data, and the dot product of its weights and the local region of the input data is computed, as shown in Figure 2.13. This produces a 2D output known as the *activation/feature map*. Therefore, the output of a convolutional layer is a stack of these feature maps, whose depth depends upon the number of kernels in the layer. The height and the width of the feature maps are usually smaller than those of the input data, but it could be maintained the same by using *zero padding*, as shown in Figure 2.13. The size of the kernels, the stride at which they are slid, and padding are all hyperparameters.

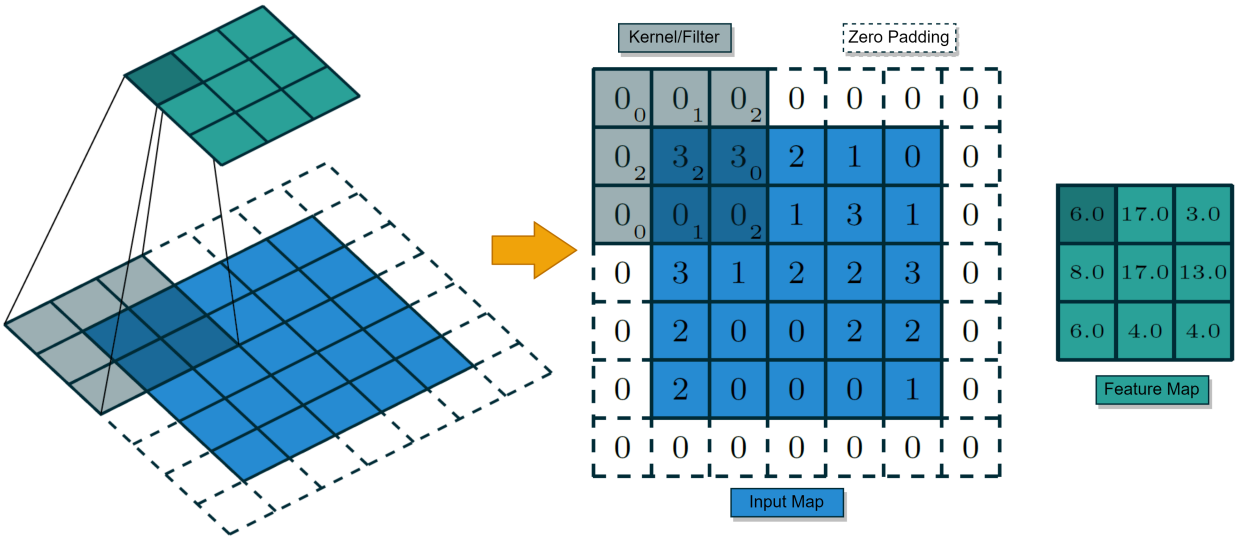

Figure 2.13: Feature map computation in a convolutional layer using a 3×3 kernel on a 5×5 input map using zero padding [4].

During the training process, the weights of the kernels are updated through backpropagation. As a consequence, each kernel learns to detect a specific pattern in the input data. Each feature

map generated shares the same weights of the respective kernel. This sharing puts a constraint on the number of parameters in the layer, making the training process of the CNN computationally faster and cheaper [153]. Normally multiple convolutional layers are stacked up one after the other, just like the neurons in the visual cortex. This hierarchical structure allows the first layer to extract low-level features, then aggregates them for the next layer to extract higher-level features, and so on.

2.7.2 Pooling Layer

The *pooling layer* essentially acts as a regularizer. The goal is to reduce the number of parameters to cut down the required computation time and resources, and prevent the CNN from overfitting. To achieve that pooling layers are placed in-between two convolutional layers in order to down-sample the spatial dimension of the feature maps. It consists of a single receptive field like the ones in a convolutional layer, discussed in Section 2.7.1. However, there are no weights associated with it. The filter simply applies an aggregation function, such as max or mean, over the local input patch. The filter size, stride, and padding can be tuned just like in the convolutional layer. Figure 2.14 shows an example of a max-pooling layer with a 3×3 filter, a stride of 1 and no padding.

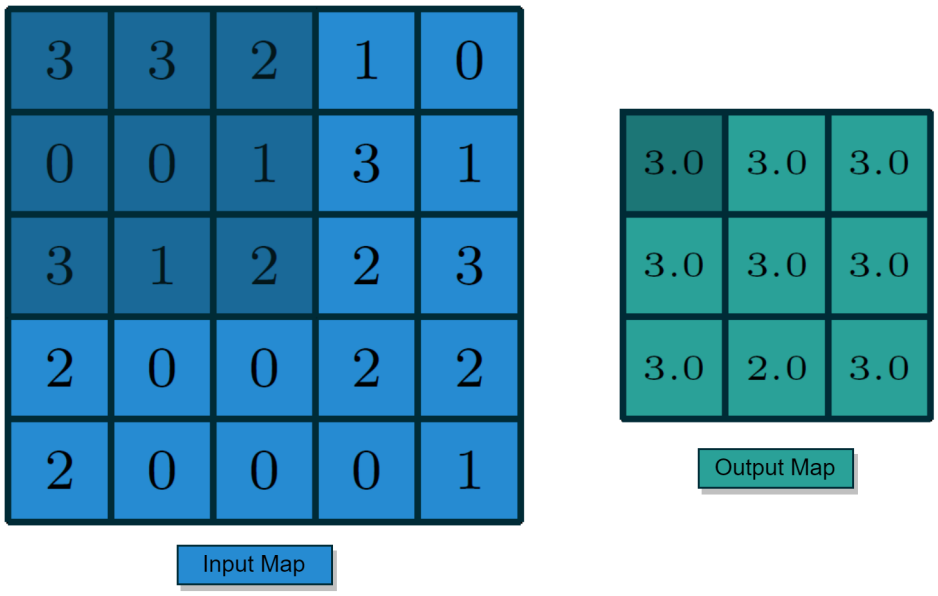

Figure 2.14: Max-pooling computation using a 3×3 kernel on a 5×5 input map [4].

2.7.3 Literature Survey

One of the first popular CNN models was the $LeNet-5$, introduced in 1998, that was able to recognize handwritten digits [140]. It had 3 convolutional layers alternating with 2 average-pooling layers, and 2 FC layers at the end. The average-pooling layers had a trainable weight and bias for each feature map from the previous layer. The $LeNet-5$ was able to show that a CNN, that assembles simple features into more complex ones over multiple layers, can be used for image classification. Further developments in CNN models were motivated by the ImageNet classification challenge, officially known as ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Inaugurated in 2010, ILSVRC currently has 1,281,167 training images, 50,000 validation images and 100,000 test images of about 1000 object classes for image classification and localization [172]. In 2012, AlexNet was the first CNN to win the challenge [154]. It had a top-5 error rate of 16.4%, whereas the second best had around 26%. It was the first CNN with adjacent convolutional layers without any pooling layer in-between. While being similar to the $LeNet-5$, AlexNet was bigger and deeper. Furthermore, it used data augmentation and Local Response Normalization (LRN) to enhance its performance [154]. Its success marked the start of the "DL era" of ML. In 2014, GoogLeNet won the competition with a top-5 error rate of 6.7% [173]. Even though it had 22 layers, the total number of parameters were less than that in the AlexNet. It was possible because of the Inception modules that utilizes the parameters much more efficiently [173]. However, the very next year, in 2015, the champion model ResNet brought down the error rate to only 3.6% [174]. It was developed with an astounding 152 layers. Training such a deep NN was possible due to the "skip connections", which simply connects the input of one layer to the output of another located a bit higher up the stack [174].

Even though CNNs were primarily developed for image processing, as mentioned in Section 2.7, they have been applied in a wide variety of research fields, one of which is CEM. In 2017, Wei Tang et al. [72] demonstrated that a CNN could be used to solve Poisson's equation. They predicted the electric potential distribution, in both 2D and 3D models, using the electrical permittivity distribution and source information organized in 2D and 3D grids for respective models. The CNN had 7 convolutional layers with ReLU activation function without any pooling layers. Another use case of CNNs was to predict the motor torque based on rotor images and act as a surrogate for an

FEM model [73]. Similarly, an architecture based on a CNN and an FNN was used to estimate flux linkage in motors [175]. In their paper in 2020, Ruohan Gong and Zuqi Tang used a CNN, named $U-Net$, for predicting magnetic and temperature fields for different topologies [74]. Furthermore, a CNN with an Encoder-Decoder architecture was used to predict magnetic field distribution for low-frequency EM problems with different geometric complexity, material, and current intensity [5].

2.8 Role of Machine Learning (ML) in Electromagnetic (EM) Simulations

Traditional first-principle methods, such as FEM, FDM, etc., are quite demanding in terms of time and resources, as discussed in Section 1.2.1. For instance, some researchers concluded the infeasibility of FEM for real-time simulation due to its increasing complexity and computation cost with increasing model size and time step [176, 177]. This can be attributed to the computationally intensive mesh generation and storage, which become even more expensive for 3D problems, and the large stiffness matrix required to solve for a large mesh, as discussed in Section 2.3.5. Others have proposed surrogate models to accelerate the design and optimization process of certain specific devices [178–181]. However, such models are only applicable for a limited number of design problems [179]. Thus, there remains this void of a general purpose surrogate model that is computationally inexpensive and adequately accurate, allowing engineers to generate quick initial results for preliminary design selection [179, 182]. Furthermore, optimizing the design of EM machines requires simulating the problem multiple times while changing only a small number of parameters. Using conventional numerical methods, such as FEM and/or FDM, requires the same amount of time for each computation regardless of the magnitude of the change. This inevitably restricts the exploration/search space for optimal design parameters due to high analysis costs [183, 184].

In order to facilitate the design exploration and optimization process various data-driven methods, discussed in Section 1.2.2, have been introduced in the recent past. This approach is implemented using DL, discussed in Section 2.5.3, to develop models that can act as general purpose surrogates for the first-principle methods. DL models can utilize the similarity of different designs and speed up the calculations, hence making them an attractive method. Previously, DL models such as a CNN, using dense regression [72–74], have been used for EM field prediction and showed

favorable results. Such networks based on a CNN expect the input data as a stacked 2D input with the geometry along with excitation and material properties, represented as a structured pixelated grid of data points. Such an approach can only extract patterns limited by the pixel density of the input data, which is predefined and remains fixed as part of the CNN architecture. Furthermore, the pixel resolution of the input is fixed throughout the input geometry; thus, it is difficult to handle complex geometries. This approach can also be wasteful in terms of resource allocation as very granular pixels can be placed in regions of the input with little or no significance.

The proposed RNN model, discussed later in Section 3.3.6, in this thesis provides an improved solution in terms of input representation by being able to process a non-uniform mesh, allowing it to handle designs of variable mesh granularity locally, without increasing the overall size of the input geometric representation. Another disadvantage of the CNN-based approach is the inability to handle input of varying dimensionality without re-tuning. With my approach, a trained model can handle both coarse and fine meshes without any architectural modification or retraining. This is possible due to the inherent property of *parameter sharing* of RNNs, as discussed later in Section 3.3.

At the same time, the RNN-based neural architecture uses a lower memory footprint than the CNN-based designs [72–74] at the time of model training and storage of the trained network. It was also observed that the proposed approach was computationally faster than the ones introduced in [72–74]. Such an approach can act as a surrogate for the solution obtained using the FEM. Although an RNN-based approach is novel in the domain of EM simulations, similar architectures have been used for efficiency map prediction [175], and a parameter-shared NN was used for classification and segmentation of point cloud based non-uniform geometric shapes [185].

The other contribution of this thesis is the application of a hybrid learning method, discussed in Section 1.2.3. This approach involves using the above proposed RNN-based architecture and combining it with a physics-based learning methodology, resulting in a Physics-Informed Recurrent Neural Network (PI-RNN) model, discussed later in Section 3.4.5, for EM analysis. PI-RNNs utilize prior knowledge about the physical laws in the form of PDEs and BCs, and embed them into a loss function, which is used to train a surrogate in the form of an NN. This allows them to circumnavigate the need for expensive meshes. Such NNs follow symmetries and conservation principles extracted from the physical laws observed in the training data. Several approaches have been proposed: a variational form, Galerkin-type projections, or the use of the PDE in a strong form [186].

3

A Deep Learning Approach for Computational Electromagnetics

3.1 Mesh Representation as a Sequence

Traditional CEM methods, such as the FDM and the FEM, solve any given EM problem by first discretizing the domain of interest into small structures, as discussed in Sections 2.3.1 and 2.3.5. This generates a mesh composed of discrete points (also called *nodes*) throughout the entire domain which are then solved iteratively. Now, for the DL approach proposed in this thesis, I have also chosen to utilize the discretization process to solve the given EM problems.

The input representing an EM problem consists of a mesh-based grid with material behavior and excitation properties. The unstructured mesh is then data engineered into a sequence of mesh points. An example is provided in Figure 3.1, where the mesh of a simple coil problem with two

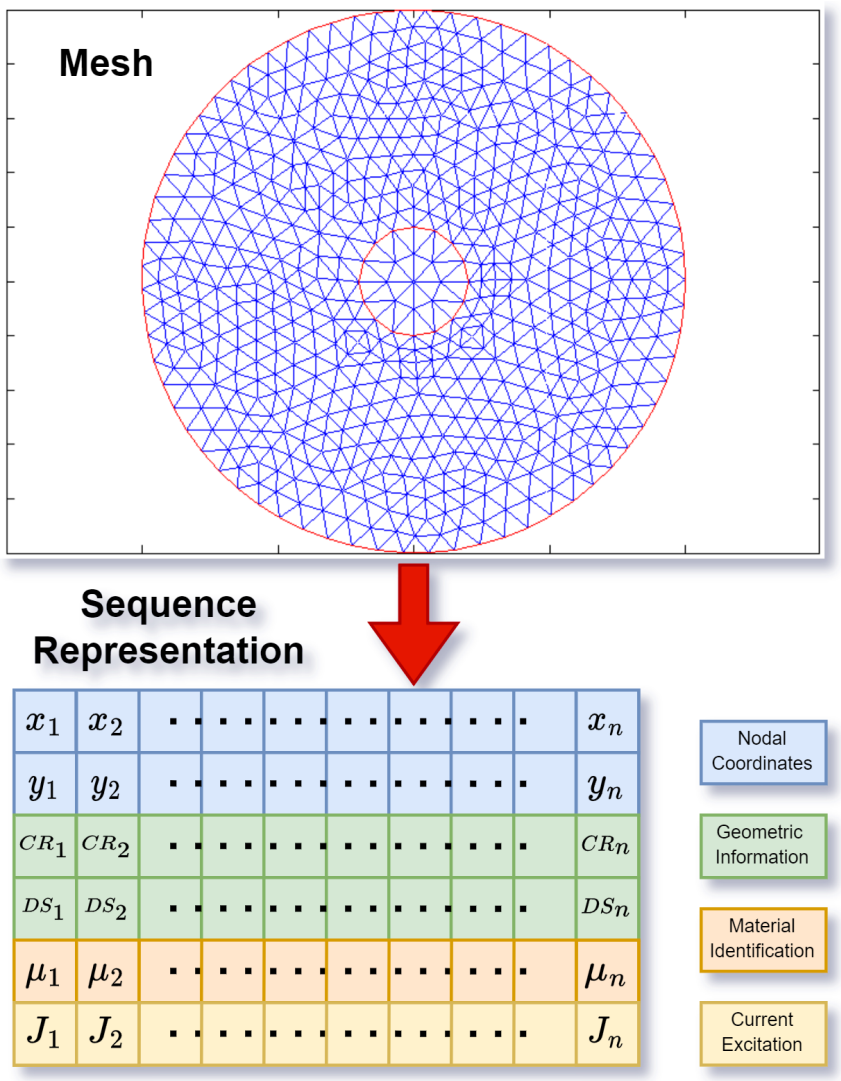

Figure 3.1: Computer-Aided Design (CAD) geometric mesh represented as a sequence.

different materials, coil and domain, is translated into a sequential format. Each point, (x, y) on the mesh is associated with a value of the properties — permeability (μ) and current density (J). Those properties are retained with the respective nodal positions while creating the sequential dataset. Furthermore, other geometric properties, such as the coil radius (CR) and the domain size (DS), are also incorporated in the sequence. For points on the interface two copies are stored — one for each domain meeting at the interface. Such a representation is then fed into the DL architecture, as described in Section 3.3.6. Once the data is input into the model, the training procedure can be conducted either by pure supervised learning, pure physics-based learning, or hybrid learning, as discussed in Sections 3.3.6 and 3.4.5, respectively.

3.2 Data Acquisition

In this work, all the DL architectures, discussed later in this chapter, are trained using some form of data-driven supervised learning method. The deeper the model, the higher the number of free parameters in the model. This imposes the requirement of a sufficiently large input dataset to train all these model parameters, otherwise, it would lead to model overfitting, as discussed in Section 2.6.3. This input dataset has a certain format. Each instance of data points in a dataset is called a *sample* or an *example*. As mentioned in Section 2.6.3, supervised learning involves two types of data — input and output. The input (also called features or attributes), as the name suggests, is what is fed into the DL models that predict a resultant output. This predicted output is then compared with the actual output (also called targets) from the dataset.

The adage "Garbage In, Garbage Out (GIGO)" couldn't be more appropriate when training ML models. Especially, for models that learn using a supervised learning method. A model can only be as good as the data it trains on. Even more so for DL models, since they also perform feature extraction during the training process. Hence, both the quantity and quality of the dataset matter for DL architectures. In terms of the quantity, the rule of thumb is — the bigger the better. As for the quality, it requires some domain expertise to examine. Ideally, it can be measured with the following aspects [187]:

- **Reliability:** How reliable is the source of data? How well does it represent the real-world problem?
- **Feature Representation:** Is the data presented with useful features?
- **Minimize Skew:** Is the data biased? If so, by how much?

To address these quantity and quality issues, the simulated data collected for this thesis are generated through industry standard Computer-Aided Design (CAD) simulations — Simcenter's MAGNET [69]. The CAD geometry represents the spatial coordinates of the problems while incorporating the material information and current excitations with sufficient resolution.

DL models are designed based on the tasks at hand along with the size of the datasets. Simple problems are better handled by simple models than by more complicated ones with a higher number of trainable parameters. As a result, to thoroughly test my hypothesis of being able to predict a

magnetic field with DL, I have chosen problems with varying complexities of dimension, material properties, and current densities. The first problem is a current carrying coil inside a specified domain, and the other one is a C-core actuator consisting of a coil wrapped around the ferromagnetic core and an armature. For the supervised learning method, the dataset needs to be formatted with some input features and output targets. The sequence representation of the mesh-based grid is used as the input with the magnetic field value being the target. For physics-based learning, only the coordinates, (x, y) are used as inputs. For each problem, around 3000 geometries were simulated, of which 90% were used for training and 10% for validation.

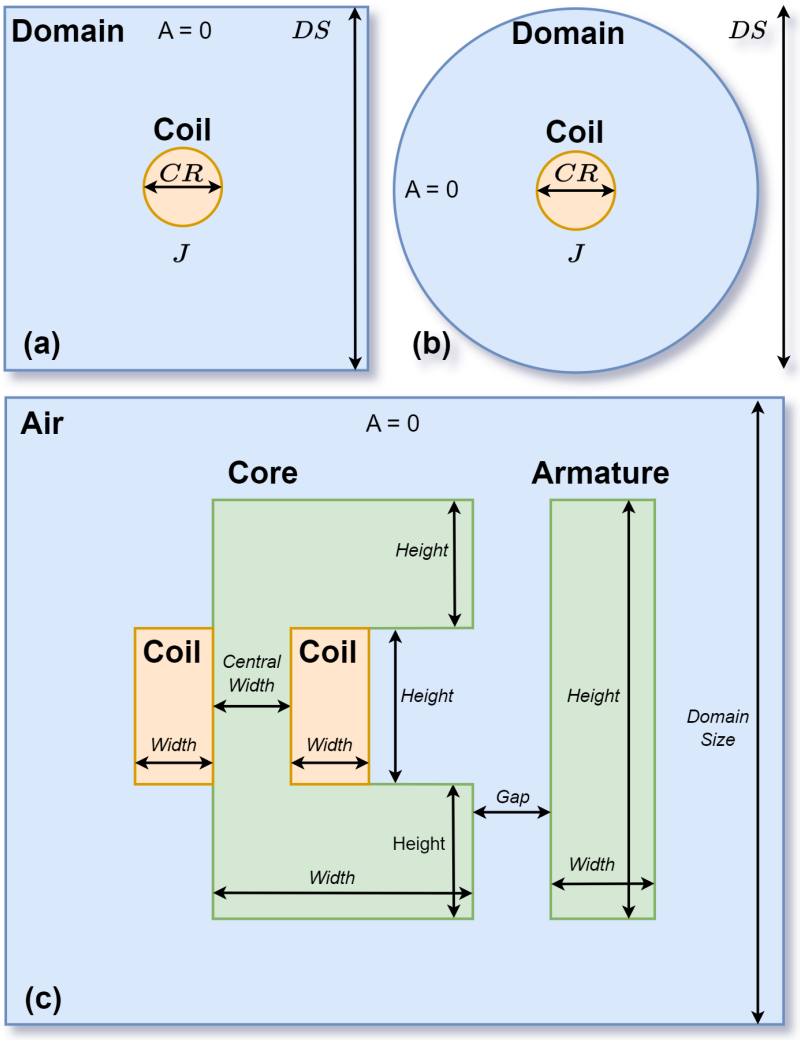

Figure 3.2: (a) Coil in a box problem, (b) coil in a cylinder problem, and (c) C-core problem.

3.2.1 Coil Problem

The first problem is a two-domain setup. A coil carrying Direct Current (DC) is placed inside either a box or a cylinder, as shown in Figures 3.2(a) and 3.2(b) respectively. The coil is made up of copper, whereas the domain around it is varied between a linear and a non-linear magnetic material. The coil radius (CR) is varied from 0.5 to 3.5 mm, the domain size (DS) is varied from 12 to 24 mm, and the current (J) is varied from 0.5 to 5.5 A. The data for this problem is collected from the Finite Element Analysis (FEA) software package — MAGNET [69]. The different material properties used in the setup were collected from the software library [69].

3.2.2 C-core Problem

The second problem is much more complex and challenging, as it involves three different domains and multiple interfaces. Here, we have an EM C-core actuator, as discussed in details in Section 2.4.1. The C-core is made of iron and wrapped around with a copper coil carrying a current, as shown in Figure 3.2(c). In the 2D image format, the coil is represented by a "go" and a "return" conductor. The current in the coil is varied from 450 to 600 At. On the right of the core is the armature, which is placed close to the C-core while avoiding any form of contact. Both the C-core and the armature are placed inside a box filled with air. The whole setup is shown in Figure 3.2(c), and the dimension ranges are provided in Table 3.1. The data for this problem was collected from three different sources — the MATLAB PDE solver [188], MAGNET [69], and the Finite Element Method Magnetics (FEMM) solver [189].

Parameter	**Range (mm)**
Armature width	$[4.5, 7.5]$
Armature height	$[27, 33]$
Coil width	$[1.5, 4.5]$
Coil height	$[15, 21]$
C-core central width	$[4.5, 7.5]$
C-core upper/lower width	$[15, 21]$
C-core upper/lower height	$[4.5, 7.5]$
Air gap	$[0.5, 1.5]$
Domain size	50

Table 3.1: C-core dimension configuration

3.3 Recurrent Neural Networks (RNNs)

Along with the basic NN, Warren McCulloch and Walter Pitts also discussed another form of NN, referred to as *Nets with Circles*, in their 1943 paper [133]. They were simply conceptualizing what is currently known as the *Recurrent Neural Network (RNN)*. In fact, the term "Recurrent Network" was formally mentioned years later by Minsky and Papert, in 1969, in their book *"Perceptrons: An introduction to computational geometry"* [190]. However, it was not until 1986, when Rumelhart, Hinton and Williams showed that an RNN could be successfully trained [160].

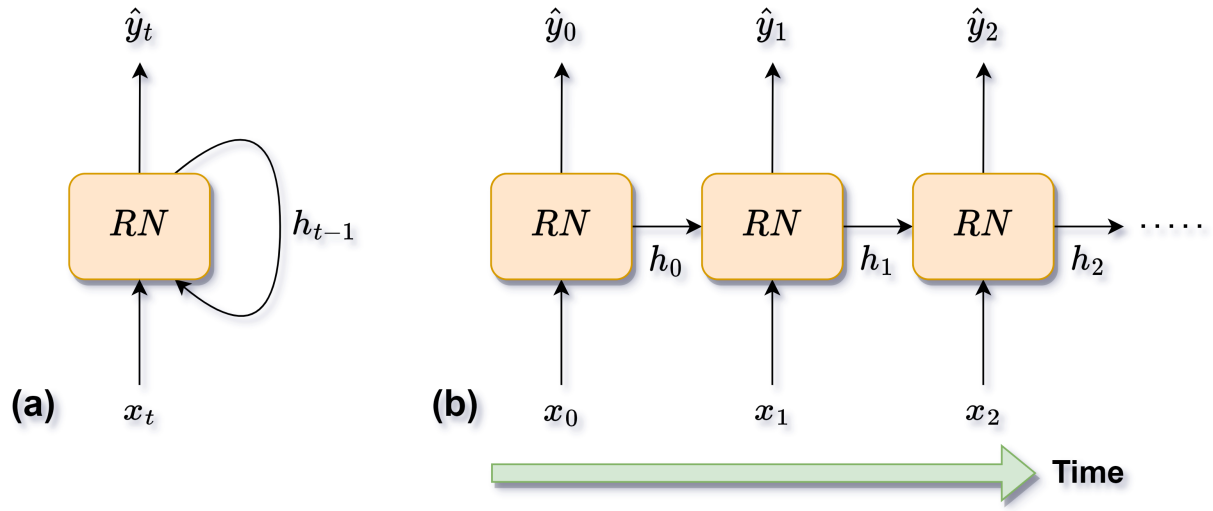

Figure 3.3: (a) A Recurrent Neural Network (RNN) with only one Recurrent Neuron (RN), and (b) the RNN unrolled through time.

An RNN, just like the regular NN, has weights, biases and an activation function for each of its layers. The main difference in an RNN is that it also consists of feedback connections along with the rest of the forward connections. This can be visualized in Figure 3.3(a), which shows the simplest possible RNN consisting of just one neuron (also known as a Recurrent Neuron (RN)). A loop in the RN can be observed, which passes the output back to the neuron as an input. This loop is executed with a step delay. For instance, the RN, in Figure 3.3(a), takes in an input, x_t at step t of the sequence, as well as the output from the previous step, h_{t-1}. As a result, the output at each sequence step, y_t depends not only on the input, x_t but also on the inputs from previous steps in the sequence. This leads to the creation of some form of memory in the RNs, and hence, they are often referred to as *memory cells*. The state of such a cell at step t is denoted as h_t, which is a

function of the input, x_t and its state at the previous sequence step, h_{t-1}, as shown in Equation 3.1:

$$h_t = f(x_t, h_{t-1}) \tag{3.1}$$

For simple memory cells, the *hidden state*, h_t is the same as the output, y_t, but it gets more complicated with other complex cells that are discussed later. Furthermore, the RN cell uses the same trainable weight, w to compute the hidden state, h_t and/or the output, y_t for each of the steps in the input sequence, which is known as *parameter sharing*. Due to this feature, RNNs can take inputs of varying sequence length, without increasing the number of trainable parameters, which makes them very efficient, and hence, ideal for processing sequential data.

RNNs are often represented against the time axis to display the flow of sequential information better, as shown in Figure 3.3(b). This is known as *unrolling the RNN through time*. Unrolling the RNN is also needed for training purposes. Regular backpropagation is applied to the unrolled RNN, and is known as Backpropagation Through Time (BPTT). The input sequence of an RN doesn't have to be literally time-dependent, i.e., it could be a sequence of any type of data where t represents the position of the data in the sequence.

3.3.1 Long Short-Term Memory (LSTM)

In theory, regular RNNs are capable of retaining memory from all the past inputs, but, unfortunately, in practice that doesn't seem to be the case. Furthermore, trying to train regular RNNs on long sequences of data leads to either exploding or vanishing gradient problems. This occurs due to the multiplicative nature of the weights in the RNN memory cells. In order to solve these problems, Hochreiter and Schmidhuber came up with a new architecture for the memory cells known as *Long Short-Term Memory (LSTM)* units [191]. LSTMs are explicitly designed to solve the long-term learning problem.

RNN layers form a chain of memory cells when unrolled, as mentioned in Section 3.3 and shown in Figure 3.3(b). Similarly, LSTMs form a repeated pattern except that the memory cell structure is different, as shown in Figure 3.4. Here the *cell state*, C_t is different than the hidden state, h_t. The cell state, C_t retains the long-term state of the cell for future use, whereas the hidden state, h_t is used to generate the output for the given time step. The information flowing through an LSTM cell

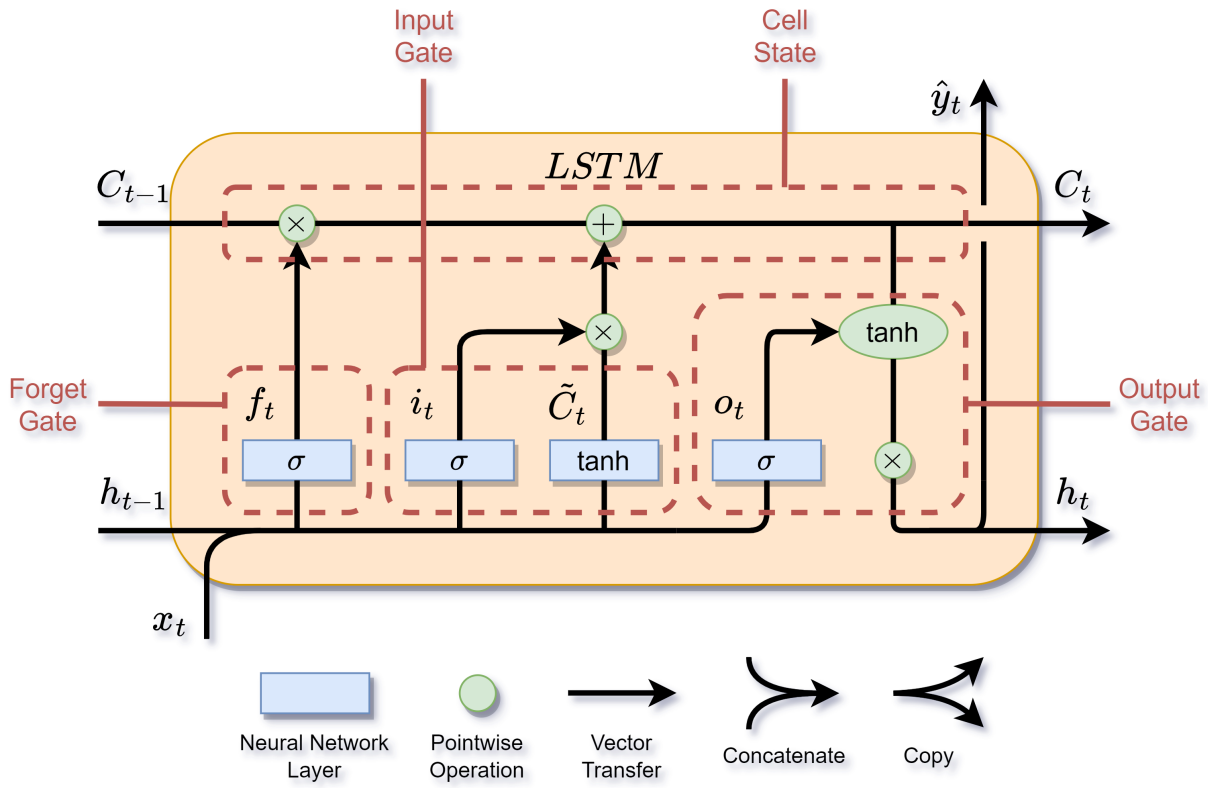

Figure 3.4: A Long Short-Term Memory (LSTM) cell

is moderated through a series of gates — *Forget*, *Input* and *Output*. A gate is simply a combination of a sigmoid layer (σ) and element-wise multiplication ($\times$) of vectors. The sigmoid function scales the information from 0 to 1 which helps to control the flow. The step-by-step update process of an LSTM cell is described as follows:

I Forget Gate: Controls the amount of information kept from the previous cell state, C_{t-1}, given by Equation 3.2:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.2}$$

II Input Gate: Controls the amount of information transferred to the current cell state, C_t, from the current input, x_t and previous hidden state, h_{t-1}, given by Equations 3.3 and 3.4:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.3}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.4}$$

III Cell State Update: Updates the current cell state, C_t as an additive linear combination of previous cell state, C_{t-1} and processed input, given by Equation 3.5:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.5}$$

IV Output Gate: Controls the amount of information needed, in terms of the hidden state, h_t, to make the current prediction, given by Equations 3.6 and 3.7:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.6}$$

$$h_t = o_t * \tanh(C_t) \tag{3.7}$$

3.3.2 Gated Recurrent Unit (GRU)

A *Gated Recurrent Unit (GRU)* cell, as shown in Figure 3.5, is a newer variant of an RN introduced by Kyunghyun Cho, et al. [192]. It is a simpler model than the LSTM unit, discussed in Section 3.3.1. Unlike LSTM, a GRU cell does not have a cell state anymore. It uses the hidden state, h_t to retain long-term memory and to make current predictions. Furthermore, it has only two gates — *Reset* and *Update*. The Update gate has the combined functionality of the Forget and Input gates of an LSTM, given by Equation 3.8. It controls the amount of information to retain from the previous hidden state, h_{t-1} and how much to add from the current input, x_t. It concatenates both h_{t-1} and x_t, and then passes them through an NN layer and applies an activation function, σ. Similarly, the Reset gate controls how much past information to forget as given by Equation 3.9. The results from both the gates are then combined using Equations 3.10 and 3.11 to generate the current output, $\hat{y}_t$ and the current hidden state, h_t. All RN cells share the same trainable weights ($W_g,\ W_r,\ W_z$), which is also known as *parameter sharing*, as discussed in Section 3.3. With parameter sharing, it can be inferred that a similar problem is solved at each node, with only the input

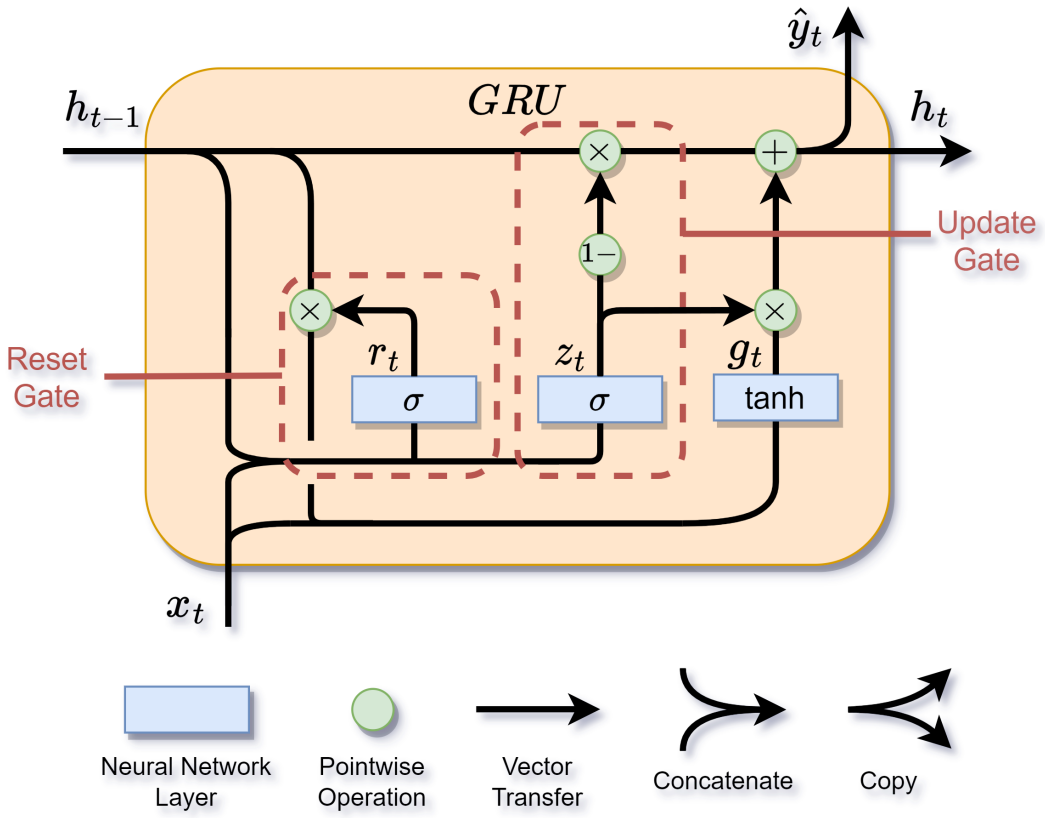

Figure 3.5: A Gated Recurrent Unit (GRU) cell

nodal information and the memory varying.

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \tag{3.8}$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \tag{3.9}$$

$$g_t = \tanh(W_g \cdot [r_t * h_{t-1}, x_t] + b_g) \tag{3.10}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * g_t \tag{3.11}$$

3.3.3 Bidirectional Recurrent Layer

A regular RNN layer simply produces an output based on the current and past inputs. It is useful for making future predictions such as weather or stock forecasts. But in many cases, such as in Natural Language Processing (NLP), it is also important to look ahead and see the future inputs in the sequence to generate the current output. Hence, the introduction of a *bidirectional recurrent layer*, as shown in Figure 3.6. In this layer, two copies of the same recurrent layer are used, where one copy takes the input from one direction and the second copy from the opposite direction (in terms of the order of the sequence). Their outputs at each time step are then combined, usually through concatenation. A bidirectional recurrent layer could be constructed using any type of RNN cell such as the basic RN, LSTM, GRU, etc.

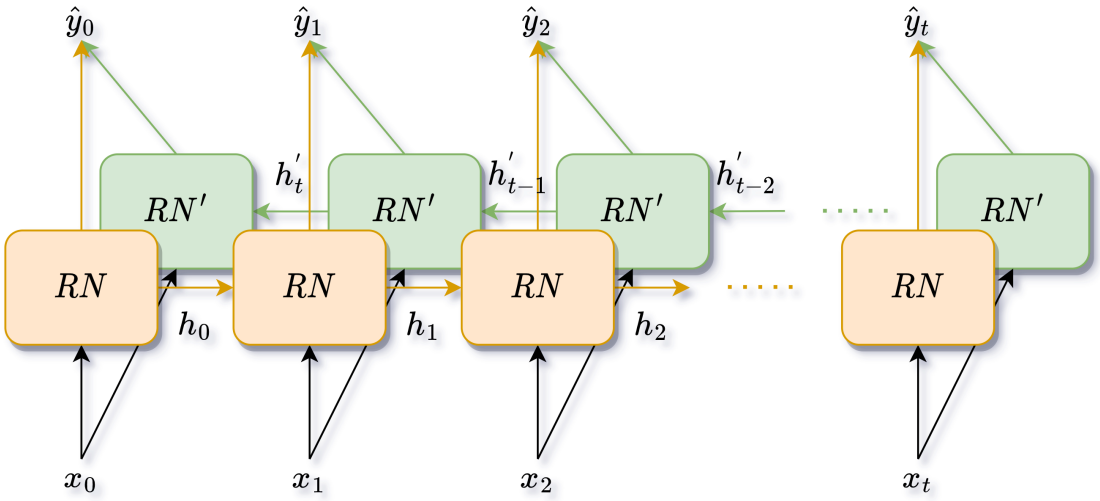

Figure 3.6: A bidirectional recurrent layer

3.3.4 RNN Architectures

One of the other benefits of an RNN over a regular NN, aside from memory retention, is that it can handle inputs of variable sizes. It can achieve that by simply unrolling itself over the required number of time steps or length of the input sequence. This flexibility facilitates the construction of different network architectures, as discussed below:

I **One-to-One:** This is almost like a basic AN, where the RN takes in one input (vector) and predicts one output (vector) just for one time step, as shown in Figure 3.7(I). Ex — traditional

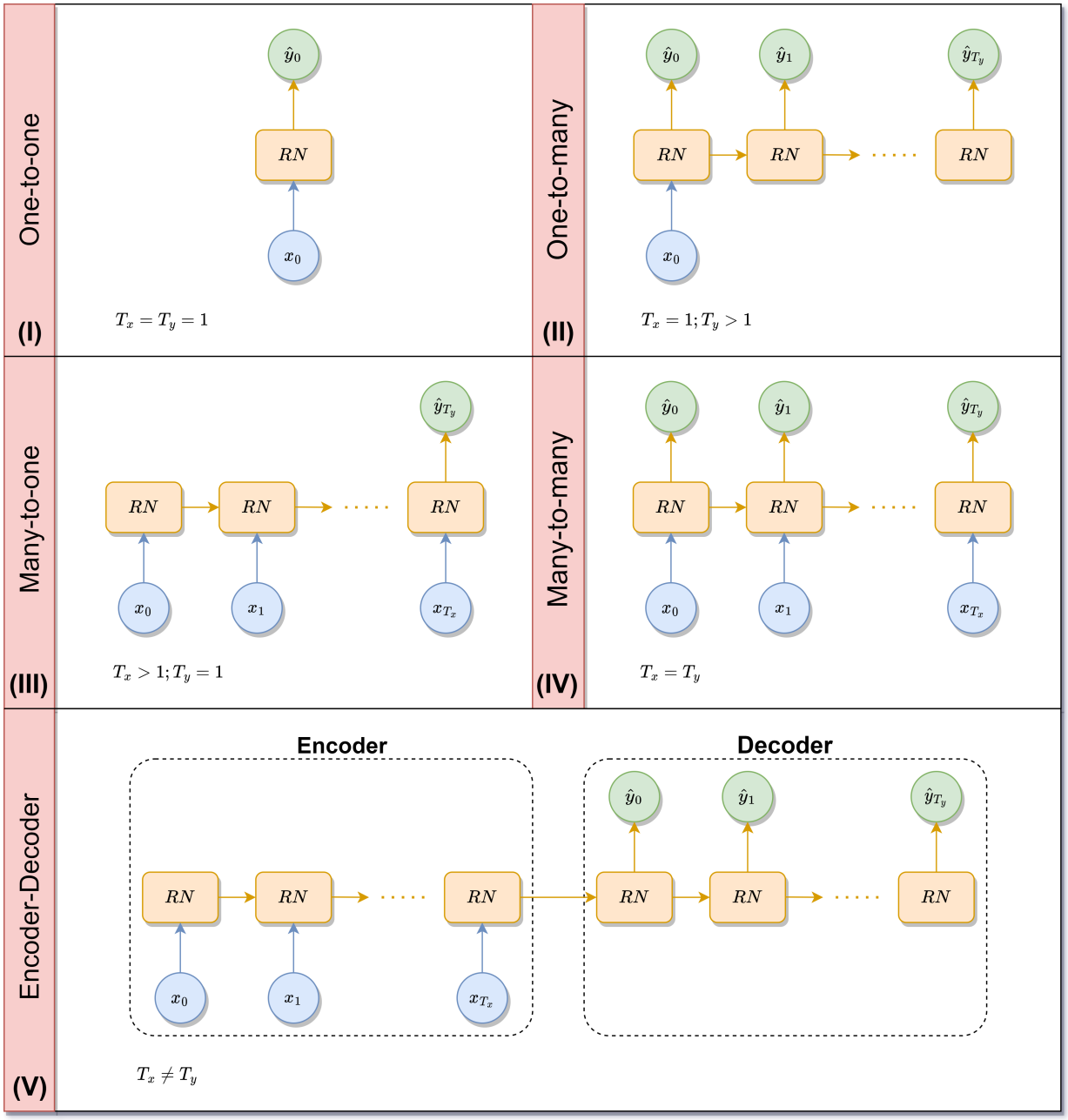

Figure 3.7: RNN architectures

RNN.

II One-to-Many: Here one input (vector) is used to generate a sequence of outputs by feeding the output from the previous time step as the current input, as shown in Figure 3.7(II). Ex — music/text generation.

III Many-to-One: Here a sequence of inputs is used to generate a sequence of outputs, but only the output in the final time step is considered, as shown in Figure 3.7(III). Ex — text classification.

IV Many-to-Many: This is similar to the many-to-one model, except that the entire output sequence is taken into account, as shown in Figure 3.7(IV). Ex — forecasting time series, such as stock, weather, etc.

V Encoder-Decoder: This could be considered a variation of the many-to-many model, but this architecture is a generic one and does not only pertain to RNNs. For the RNN version, the *encoder* and the *decoder* are represented by a many-to-one and a one-to-many model, respectively, as shown in Figure 3.7(V). The fundamental concept remains the same, where the encoder tries to map the useful features of the input sequence into a vector representation, called the *context*, which is then used by the decoder to generate a sequence of output. Ex — machine translation.

3.3.5 Literature Survey

RNN based DL models have been used for EM analysis in the past, such as predicting efficiency maps for motors using current excitation, torque values, and base speeds [175]. The paper uses an Encoder-Decoder architecture with an RNN, which was first implemented by Kyunghyun Cho et al. in 2014 [192]. In the standard architecture the context vector depends on the final output of the encoder, but sometimes different parts from the encoder are useful in generating the output in the decoder. To resolve this issue, a technique known as an *attention mechanism* was first introduced by Dzmitry Bahdanau et al. [193]. It computes a weight vector at each time-step of the decoder that controls the level of influence that each encoder hidden state has on the context. These weights, also referred to as *attention weights*, are often computed using a softmax and a scoring function

based on the encoder's current hidden states and the decoder's previous hidden states. This simply allows the neighboring values in the input sequence to contribute to the prediction of a given point. Other implementations of the attention mechanism have been shown by Luong et al. [194] and by Ashish Vaswaniet et al. [195].

Due to their acyclic nature, RNNs are designed to handle data formatted in sequences. The distinct attribute of RN cells to preserve memory allows them to link the geometry's spatial information, which is inputted as a sequence. By representing a 2D/3D geometry as a 1D sequence, the spatial information is lost. However, the acyclic nature of RNNs and the inherent memory linking every element of the sequence can compensate for this loss of representation. In the case of large problems with high numbers of nodes, bidirectional recurrent layers, discussed in Section 3.3.3, and an attention mechanism [193] can be used. However, for problems discussed in this work, a simple RNN architecture was sufficient.

3.3.6 RNN Model Architecture

In this work, the first proposed architecture is an entirely data-driven RNN model. The RNN model, shown in Figure 3.8, uses an Encoder-Decoder architecture [192], as discussed in Section 3.3.4. The encoder encapsulates the information from the input sequence into its final output as a vector representation, referred to as the *context vector*. This vector is then used as the input to the decoder, which then makes the corresponding sequential predictions.

The encoder is represented by two GRU layers, discussed in Section 3.3.2, which take in variable-length sequences of mesh nodes (x, y) (based on the number of nodes in the mesh design), along with geometric and material information, and current values. Once the input data has been *encoded*, it is passed to the decoder. The context vector contains the encoded information extracted from the input that would be relevant for the decoding process. The encoder performs a form of feature extraction as mentioned in Section 3.2.

The decoder is made up of a dense layer (also called an FC layer), as mentioned in Section 2.6.2, wrapped around by a *Time Distributed* layer, as shown in Figure 3.8. A Time Distributed layer simply applies the core layer (in this case, a dense layer) to each of the steps of the incoming sequence of data [131]. In other words, the same dense layer (parameter sharing) is applied to produce one output per input node of the incoming sequence vector from the encoder, resulting in a sequence of

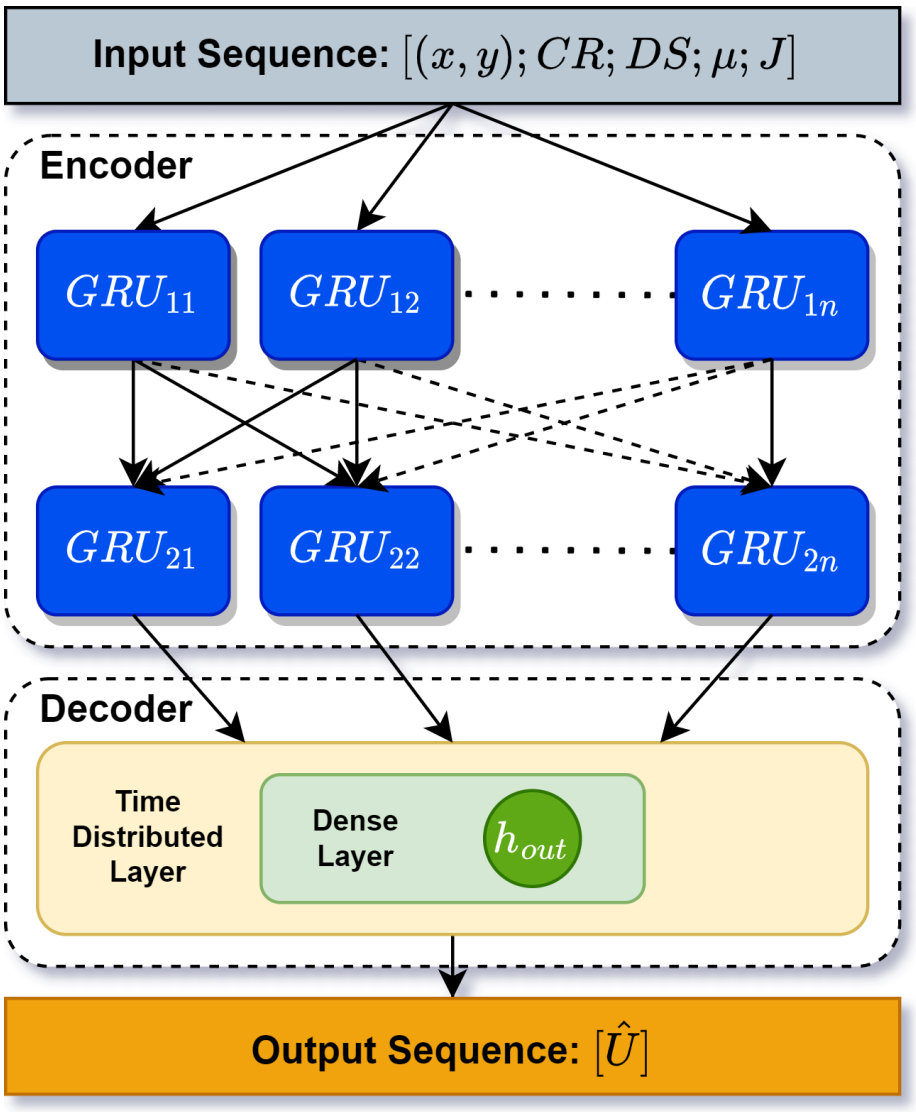

Figure 3.8: The RNN architecture used for the coil problem.

outputs. This sequence represents the magnetic flux density, B values as the predicted output. The model's flexibility of being able to handle inputs of varying length (based on the number of nodes) is due to RNN's *parameter sharing* feature, discussed in Sections 3.3 and 3.3.2. Consequently, RNNs can be *unrolled* to any length of sequence without increasing the number of model parameters, since the RN cells share the same trainable weight. Likewise, the Time Distributed layer uses the same dense layer to predict each of the values in the output sequence. The model was trained by minimizing the MSE between the predicted magnetic field values and the ground truth from the FEA solver.

To enhance the performance of the model and to avoid the exploding gradient problem [196, 197], a dropout rate of 0.2 (20%) was used for each of the GRU layers [163, 164]. As discussed in Section 2.6.4, it acts like a regularization agent by randomly switching off certain neurons during the training process. Furthermore, LN, discussed in Section 2.6.6, was added after each GRU layer to improve the gradient stability during the training process [167]. It learns a scale and an offset parameter for each input and normalizes the features. The training is done using the Adam optimization algorithm [161].

3.4 Physics-Informed Neural Networks (PINNs)

All the DL models that have been discussed so far are based on a supervised learning technique, discussed in Section 2.6.3. As a result, they are inherently dependent on the dataset that is used in the training process. In today's world, with the emergence of internet and the proliferation of online users, we have been able to collect and store data on a more massive scale than ever before. The variety of datasets includes images, text, voices recordings, financial transactions, etc. These data have been fueling the development of such DL models. However, unfortunately, it's not always possible to acquire large datasets for all problems. For instance, in the EM systems being considered in this thesis, the datasets are just not big enough and are expensive to generate. With the lack of datasets of sufficient size, even the state-of-the-art DL models cannot perform up to par. However, these problems conform to the laws of EMs, i.e., Maxwell's equations. Such laws have been studied, postulated and well-defined throughout human history. Thus, this sparsity of data has led to the development of a new kind of model that incorporates these physical laws, known as the *Physics-Informed Neural Network (PINN)* [198].

3.4.1 Partial Differential Equations (PDEs)

To constrain PINNs to the physical laws, we need to define them in mathematical terms that can be solved analytically or numerically to begin with. When investigating any natural phenomenon we try to parameterize the causal effect of something, i.e., what happens to Y when X changes. This has led to the use of ODEs and PDEs in the formulation of various physical laws governing electromagnetism, electrostatics, thermodynamics, diffusion, heat, quantum physics, etc. Since most

of the mathematical models developed through the course of time are multivariate objects instead of univariate ones, the focus is placed on PDEs here. Although the application of ODEs and PDEs in PINNs are interchangeable. A PDE can be defined using a generic form, as shown in Equation 3.12:

$$F(x_1, x_2, ..., x_n, y, D^{k_1}, D^{k_2}, ..., D^{k_m}) = 0, \tag{3.12}$$

where

$$y = y(x_1, x_2, ..., x_n), \tag{3.13}$$

and

$$D^k = \frac{\partial^k y}{\partial x_1} ... \frac{\partial^k y}{\partial x_n} \tag{3.14}$$

Here, F is the given function, y in the unknown/latent function (given in Equation 3.13), x_n is the n^{th} independent variable, and D^k is the $k^{\text{th}}-$order partial derivative with respect to all the variables (given in Equation 3.14). According to the French mathematician Jacques Hadamard, mathematical models of any physical phenomenon should be *well-posed*. This means that the PDE should have these following properties:

- **Existence:** For every (valid) input variable, at least one solution exists.
- **Uniqueness:** For every (valid) input variable, the solution is unique.
- **Stability:** The solution depends continuously on input parameters and ICs, i.e., small changes in the input and ICs lead to small changes in the solution.

In practice, when defining a well-posed PDE, we need to consider both the ICs and the BCs for the problem. There are primarily three possible options for BCs: *Dirichlet*, *Neumann* and *Robin*.

In this work, I have focused on electromagnetism, more specifically how the magnetic field varies in low-frequency EM systems, as mentioned in Sections 2.1.3, 2.2 and 2.3. For that I need to use Ampere's law of electromagnetism, discussed in Section 2.1. The final form of the law, that I intend to use for my application, leads to Poisson's equation for magnetic field, as given in Equation 2.19. For my problem, I use the Dirichlet BC for the PDEs, which is discussed in Section 3.4.5. Furthermore, the PDEs in question are also well-posed.

3.4.2 Designing PINNs

Now that we have a well-defined mathematical model of the physical law for the problem, how do we modify the architectures of the conventional DL models to take that into account? As it turns out, all we need to do is replace the loss function, as described in Section 2.6.1, with the PDE, BC and IC of the problem, and try to minimize them. This idea is predicated on the Universal Approximation Theorem [159] and the advancements in Automatic Differentiation (AD) [199]. As mentioned in Section 2.6, NNs can approximate any function, y with arbitrary precision. Furthermore, due to AD, we have access to the derivatives of the function, y with respect to all its inputs, x and t. We can now re-arrange the underlying PDE as follows:

$$f := y_t + \mathcal{N}[y] + c = 0, \quad x \in \Omega,\ t \in [0, T], \tag{3.15}$$

where $y(x, t)$ is the latent function that we are trying to approximate using an NN, $\mathcal{N}[\cdot]$ is a nonlinear differential operator and c is a constant. The function $f(x, t)$, that we have defined in Equation 3.15, represents the *Physics-Informed Neural Network (PINN)* [198].

The procedures to implement a PINN, which is shown in Figure 3.9, to solve any given problem are summarized below:

I Define a DL model, such as an NN, CNN, RNN, etc. which will form the basis for the PINN.

II Setup separate training sets for the PDE, IC and BC.

III Define the loss functions in terms of the PDE, IC and BC (usually by summing the weighted MSE losses of those equations).

IV Train the DL model through backpropagation and minimize the loss functions.

The DL model, NN(θ) is used to approximate the latent function, $y(x, t)$, and the PINN function, $f(x, t)$ is also defined by this NN(θ) with the aid of AD. AD enables us to compute the derivative (of any order) of the output of NN(θ) with respect to any of the input variables by applying the chain rule for differentiation [199]. Hence, due to the parameter sharing between the functions $y(x, t)$ and $f(x, t)$, training NN(θ) implicitly implies training the PINN.

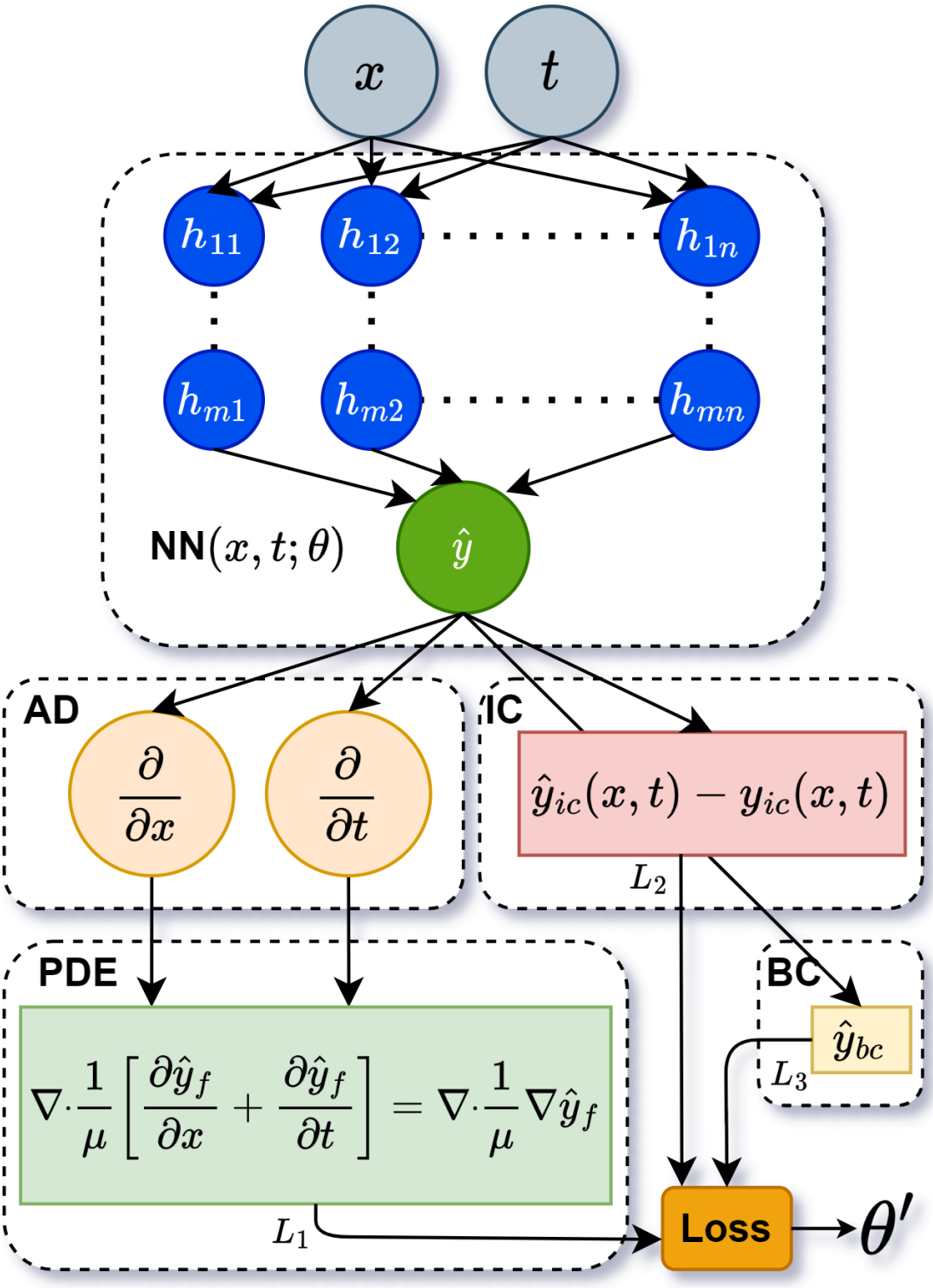

Figure 3.9: A Physics-Informed Neural Network (PINN) architecture with a basic NN as the base, and loss function computed as a linear combination of the MSEs of the PDE, IC, and BC.

3.4.3 Training PINNs

PINNs are trained through backpropagation, just like any other NN. The key difference being the loss function, which now incorporates the physical constraints of the given problem. Usually, the MSE is used for convenience when specifying the loss function [198, 200]. The loss function for a PINN, given by Equation 3.16, is defined as a linear combination of the MSEs of the PDE, IC, and BC of the problem, given by Equations 3.17, 3.18, and 3.19, respectively:

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(\theta) + w_{ic} \mathcal{L}_{ic}(\theta) + w_{bc} \mathcal{L}_{bc}(\theta), \tag{3.16}$$

where

$$\mathcal{L}_f(\theta) = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(x_f^i, t_f^i)|^2, \tag{3.17}$$

$$\mathcal{L}_{ic}(\theta) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |\hat{y}(x_{ic}^i, t_{ic}^i) - y_{ic}^i|^2, \tag{3.18}$$

$$\mathcal{L}_{bc}(\theta) = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} |\hat{y}(x_{bc}^i, t_{bc}^i) - y_{bc}^i|^2, \tag{3.19}$$

and w_f, w_{ic}, and w_{bc} are the respective weights. The training set for PINNs consists of the collocation, initial, and boundary data points denoted as $\{x_f^i, t_f^i\}_{i=1}^{N_f}$, $\{x_{ic}^i, t_{ic}^i, y_{ic}^i\}_{i=1}^{N_{ic}}$, and $\{x_{bc}^i, t_{bc}^i, y_{bc}^i\}_{i=1}^{N_{bc}}$, respectively. Here, the IC and BC MSEs are minimized using the ground truth (target) values, y^i, i.e., supervised learning. Whereas, for the PDE the equation itself is minimized using only the collocation points, i.e., unsupervised learning. This imposes a structured information on the learning process of the model, which enables it to generalize well with only a small sub-set of the training data [198]. The minimization is done through an optimization algorithm such as Adam [161] or Limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) [201]. As discussed in Section 2.6.3, to prevent a model from overfitting we need to use some form of regularization. Here, $w_f \mathcal{L}_f(\theta)$ performs such a role for PINNs, by constraining the model's parameters θ to only those outputs satisfying the underlying PDE [198, 200].

Theoretically, the convergence of PINNs cannot be guaranteed. However, in their paper, Raissi, Perdikaris, and Karniadakis have shown that if the PDE is well-posed then this method of training "*is capable of achieving good prediction accuracy given a sufficiently expressive neural network archi-*

tecture and a sufficient number of collocation points N_f" [198]. Furthermore, it has been suggested that proper hyperparameter tuning of the boundary weight, w_{bc} would enhance both the speed and performance of the model [202]. In addition, the authors of [203] have recommended to use a large positive value for w_{bc} for better prediction results.

3.4.4 Literature Survey

The concept of a PINN was first formally introduced in 2017 as a new category of data-driven DL models [198]. Although it is trained using a hybrid method, a PINN can be viewed as an unsupervised learning approach when it is trained solely using the underlying differential equations (PDEs/ODEs) and BCs [198]. In fact, this approach of solving PDEs by an NN was first attempted in 1994 by M. W. M. G. Dissanayake and N. Phan-Thien [204], and later by I. E. Lagaris, A. Likas, and D. I. Fotiadis on irregular boundary problems [205]. Due to its ability to exploit the physical laws and self-regularize itself, PINN has become increasing popular and is at the forefront of DL research. Moseley et al. [206] suggested using separate NNs, one for each subdomain in a multi-domain analysis. In his paper [186], A. Khan has shown promising results in the application of PINNs for EM analysis. His experimental results show the performance of PINNs on multi-domain problems with varying complexities. Recently, another application of PINNs for solving electrostatic Laplace's equation was also demonstrated in [207].

Furthermore, researchers have taken the liberty to experiment with different variations of PINNs to suit their problems. One such is a hp-Variational Physics-Informed Neural Network (hp-VPINN) [208], and another being a Conservative Physics-Informed Neural Network (CPINN) [53]. The latter was proposed by Jagtap et al [53] on discrete domains. In this framework, the complete solution is recreated by patching together all of the solutions in each sub-domain using the appropriate interface conditions. The domain segmentation makes it possible to parallelize the training process, which is crucial to achieve computing efficiency. Another variation involves using Hypernetworks with PINNs to solve inverse EM problems [209].

Activation functions play an important role in the training process of DL models, as discussed in Section 2.6.1. The most common ones used are ReLU, Sigmoid and Tanh [131, 153]. Researchers have come up with a dynamic activation function with a trainable parameter known as the *Swish* activation function. It is defined using Equation 3.20:

$$Swish(x) = x \cdot Sigmoid(\beta x), \tag{3.20}$$

where β is the trainable parameter. Swish has been claimed to perform the best among the ones mentioned above [210].

Similarly, the architecture of the DL models forming the bases for PINNs is crucial to their performance. Smaller DNNs may be unable to effectively approximate unknown functions, whereas over-large DNNs may be difficult to train, particularly with small datasets. From the many experimental applications of PINNs to real-world phenomena, it can be observed that the underlying NNs can become very deep with multiple layers stacking up. This may lead to higher training costs and poor computational efficiency.

3.4.5 PI-RNN Model Architecture

In the second model proposed in this work, a hybrid learning algorithm is used. The PINN, discussed in Section 3.4, incorporates a supervised learning model for the boundary and the interfaces, as well as a PDE solver for the domains, i.e., unsupervised learning. The supervised component is used to accelerate the training process as demonstrated by Khan [186]. In this thesis, RNNs, discussed in Section 3.3, are used as the bases for the PINN, as shown in Figure 3.10. Hence, I refer to my model as *Physics-Informed Recurrent Neural Network (PI-RNN)*. Each RNN is used to solve for a particular material type. For instance, one RNN is used to approximate for the coil made of copper, another RNN for the C-core and armature, which are both made of iron (ferromagnetic material), and another RNN for air. To maintain continuity at the interfaces, solutions from the RNNs for the domains on either side of the interface are averaged and used as a loss component to be minimized, as shown in Equation 3.27. For the two problems being considered in this work, the underlying well-posed PDEs, as discussed in Section 2.1.3, are given by Equations 3.21 and 3.22:

$$f := -\nabla^2 \frac{1}{\mu_r \mu_0} A = J, \quad (x, y) \in \Omega_{coil} \tag{3.21}$$

$$f := -\nabla^2 \frac{1}{\mu_r \mu_0} A = 0, \quad (x, y) \in \Omega_{domain}, \; and \; \in \Omega_{core} \tag{3.22}$$

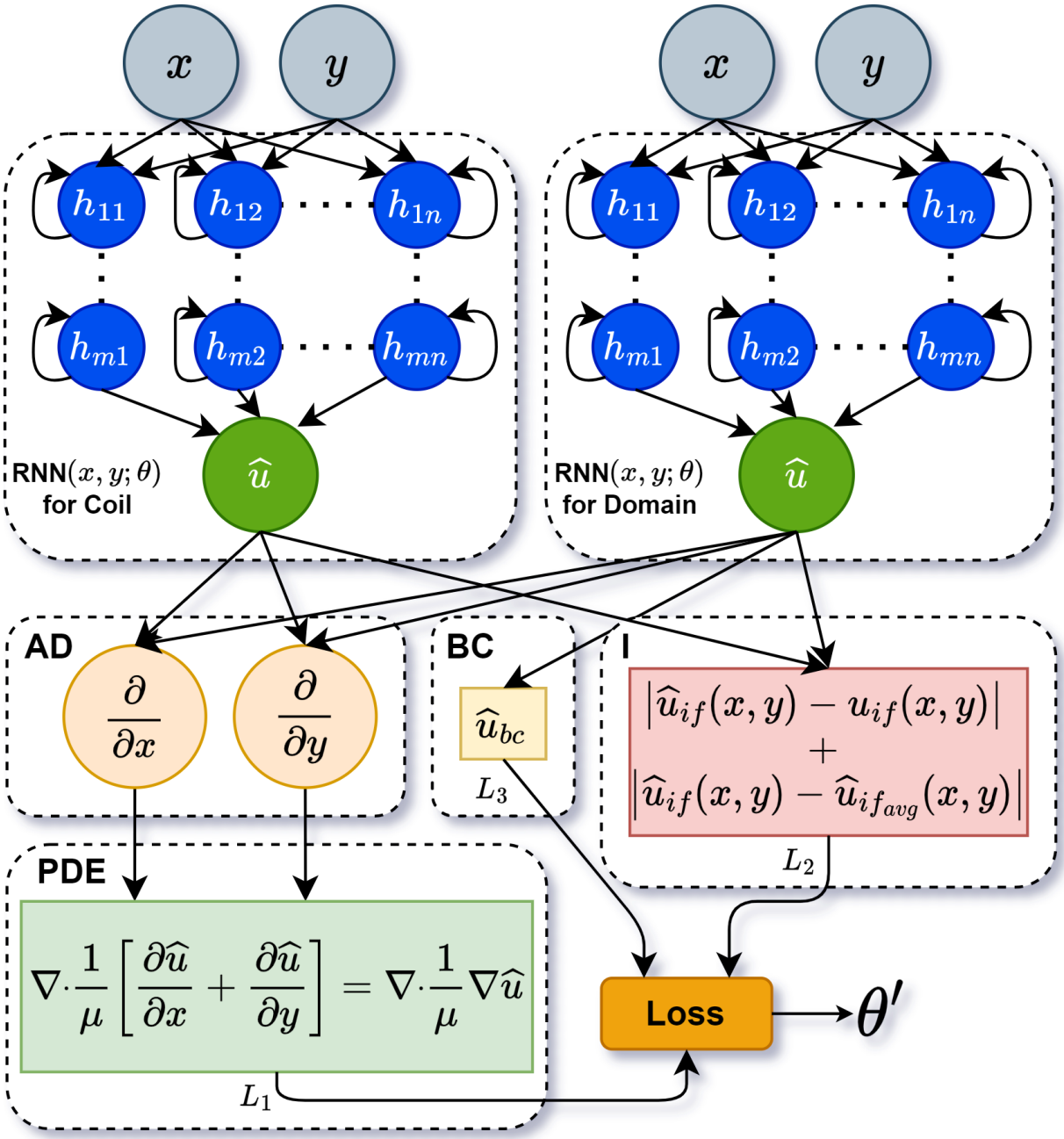

Figure 3.10: The Physics-Informed Recurrent Neural Network (PI-RNN) architecture used for the coil problem.

where μ_r is 1 for linear paramagnetic materials like air and copper, and 5000 for linear high-permeability magnetic material, such as iron, operating below saturation. For materials with non-linear $B-H$ properties, such as transformer steel, μ_r is defined using Equation 3.23:

$$\mu_r = (\ \frac{\mu_{max}}{1 + c||\nabla A||^2} + \mu_{min})\ , \tag{3.23}$$

where $\mu_{max} = 5000$, $\mu_{min} = 200$, and $c = 0.05T^{-2}$ [211]. The value for A ranges from 0 to 2.5 Tm, which is well into the non-linear portion of the material's $B-H$ curve. The Dirichlet BC used for the problem is given by Equation 3.24:

$$A = 0, \quad (x, y) \in \Omega_{boundary} \tag{3.24}$$

As discussed in Section 3.4.3, PI-RNNs are trained through backpropagation, just like any other NN. The loss function for the PI-RNN is defined as a linear combination of the MSEs of the PDE (L_1), the continuity at the interfaces, IF (L_2), and the BC (L_3), as shown in Equations 3.25, 3.26, 3.27 and 3.28, respectively:

$$MSE = w_{PDE}MSE_{PDE} + w_{IF}MSE_{IF} + w_{BC}MSE_{BC}, \tag{3.25}$$

where

$$MSE_{PDE} = \mathcal{L}_1(\theta) = \frac{1}{N_f}\sum_{i=1}^{N_f}|f(x_f^i, y_f^i)|^2, \tag{3.26}$$

$$MSE_{IF} = \mathcal{L}_2(\theta) = \frac{1}{N_{if}}\left[\sum_{i=1}^{N_{if}}|\hat{u}(x_{if}^i, y_{if}^i) - u_{if}^i|^2 + \sum_{i=1}^{N_{if}}|\hat{u}(x_{if}^i, y_{if}^i) - \hat{u}_{if_{avg}}^i|^2\right], \tag{3.27}$$

$$MSE_{BC} = \mathcal{L}_3(\theta) = \frac{1}{N_{bc}}\sum_{i=1}^{N_{bc}}|\hat{u}(x_{bc}^i, y_{bc}^i) - u_{bc}^i|^2, \tag{3.28}$$

and w_{PDE}, w_{IF}, and w_{BC} are the respective multipliers used as regularization parameters to prevent the model from overfitting, discussed in Section 2.6.3 [198, 200]. The training set for PI-RNN consists of the collocation, interface, and boundary data points denoted as $\{x_f^i, y_f^i\}_{i=1}^{N_f}$, $\{x_{if}^i, y_{if}^i, u_{if}^i\}_{i=1}^{N_{if}}$, and $\{x_{bc}^i, y_{bc}^i, u_{bc}^i\}_{i=1}^{N_{bc}}$, respectively, where x and y are the coordinates, and u and $\hat{u}$ are the corresponding actual and predicted magnetic field values, respectively. The minimization is done through the Adam

optimization algorithm [161].

3.5 Methodology Overview

In this thesis, I have proposed a DL approach for CEM. For demonstration of the POCs I have chosen to solve for the magnetic field distribution for two different problems — coil and C-core, discussed in Sections 3.2.1 and 3.2.2, respectively. I have used specialized types of DNNs — RNNs and PI-RNNs, which have been discussed in Sections 3.3 and 3.4, respectively.

The initial step was to collect and prepare the data for my DNN models, discussed in Section 3.2. The next step was designing the model architectures for RNN and PI-RNN, followed by hyperparameter tuning, discussed in Sections 3.3.6, 3.4.5 and 4.2, respectively. Finally, I evaluated my models' predictions by comparing the experimental results with the ground truth, collected from FEM models, and compared the performances with that of other DL models used as benchmarks, discussed in Section 4.2.

4

Experiments

4.1 Benchmark Performance

A previous published work from the lab is used as a benchmark to validate the proposed model architectures [5, 212]. The benchmark is also applied on similar EM problems used in this thesis, as mentioned in Section 3.2.

A. Khan has used the data-driven approach using a CNN-based model, discussed in Section 2.7, to predict the magnetic field distribution in different EM problems. The DL model uses an Encoder-Decoder architecture, discussed in Section 3.3.4. It is comprised of 32 layers — 16 convolutional layers, 8 pooling/up-sampling layers, and 8 dropout layers [5]. The layers are grouped together in blocks, each of which consists of 2 sets of 3×3 convolutional layers, a batch-normalization layer, and a 2×2 max-pooling/up-sampling layer.

The CNN model is trained on an NVIDIA 1080 Ti GPU. For a batch size of 16, the model took

around 10 *minutes* per epoch to train. With $60 - 100$ epochs to achieve the convergence, the net training time taken was around $12 - 15$ *hours* [5]. The prediction time for the whole geometry was about $20 - 30$ ms, and the error rate for prediction was around 1.5%. All the results are tabulated in Table 4.1.

Problem	**Coil with Linear Material**		
Models	*Error*	*Training Time*	*Prediction Time (whole geometry)*
CNN Model	$\sim 1.5\%$	$12 - 15$ *hours*	$20 - 30$ ms

Table 4.1: CNN performance benchmark [5]

4.2 Results and Discussion

All the models experimented with in this thesis were run on an Intel(R) Xeon(R) Silver 4210 CPU @2.20GHz with 10 cores, an NVIDIA Quadro P5000 GPU with 16GB memory and 64GB of RAM. A 64-bit Windows 10 operating system was used.

4.2.1 RNN Model Performance

In this work, the RNN model, discussed in Section 3.3.6, was used to solve the coil problem only, mentioned in Section 3.2.1. It has been trained in two different ways: using a single model for both the coil and the outer domain, and using separate models for each material. For the former method, the input was a sequence of nodal coordinates, (x, y) of the underlying mesh; geometric information such as the coil radius (CR) and the domain size (DS); material identification as a one-hot encoded vector (μ); and current excitation (J). On the other hand, when training using separate models, the material identification (μ) was skipped, and the input sequences were fed to each RNN model for the respective material. For both the methods, the models were used to predict a sequence of the corresponding magnetic field values $[\hat{U}(x, y, CR, DS, \mu, J)]$ for the given input sequence. The inputs were normalized, and hence, a Sigmoid activation function was used in the output layers. The models were tuned for optimal hyperparameters using a grid search, with the final configurations shown in Table 4.2.

The models were built in the Python 3.8.13 programming language [213] using TensorFlow 2.10.0 framework [143], CUDA Toolkit 11.2.2, and cuDNN SDK 8.1.0. The performances of the

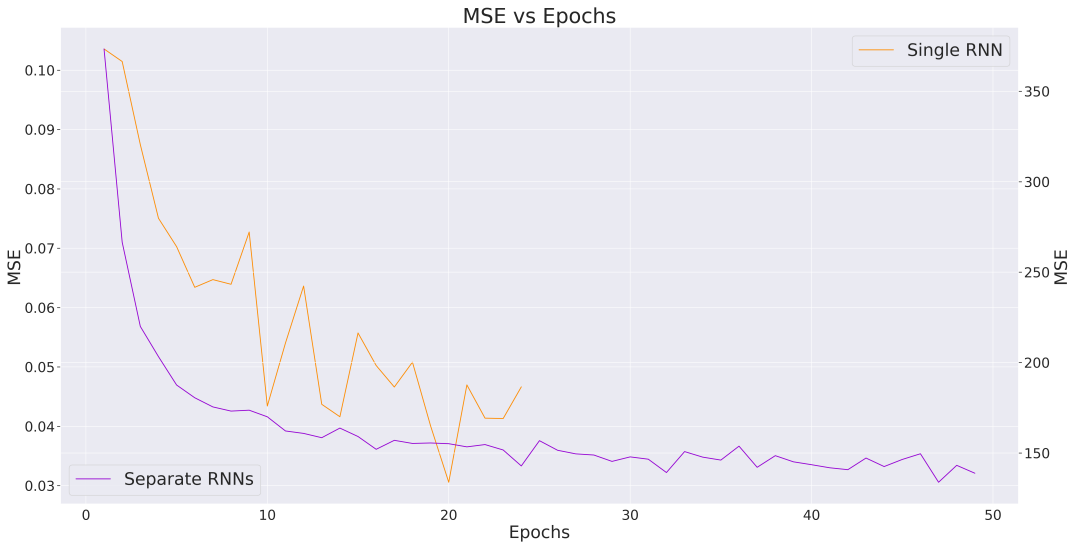

Figure 4.1: MSE plots for the single-model (orange) and the separate-model (purple) RNNs for the coil problem, with the corresponding MSE scales on the left and the right Y-axes, respectively.

Configuration	**No. of Layers**	**No. of Neurons in each Layer**	**Activation Function**
Single Model for all materials	2	128, 64	Elu, Elu
Separate Models for each material	2	64, 32	Tanh, Tanh

Table 4.2: RNN model hyperparameter configurations

best architectures, based on the lowest MSE, are shown in Figure 4.1. The predictions for the single-model and separate-model RNN architectures along with the solutions (ground truth) for the magnetic field, B values from the FEM solver are shown in Figures 4.2 and 4.3, respectively. The color scale shows the normalized intensity of the magnetic field, B $[0, 1]$ across the core and the domain for the coil problem. Although the unit for the magnetic field, B is T $(tesla)$, the output values of the models are unitless since they are all normalized. From the figures we can observe a close match between the models' predictions and the FEM solutions. The field predictions for the cores are almost always accurate, and for the domains are quite comparable to the ground truth values. However, the nodes near the interfaces are always the trickiest since they have to deal with the field traversing from one material to another, and hence, that's where the models' predictions

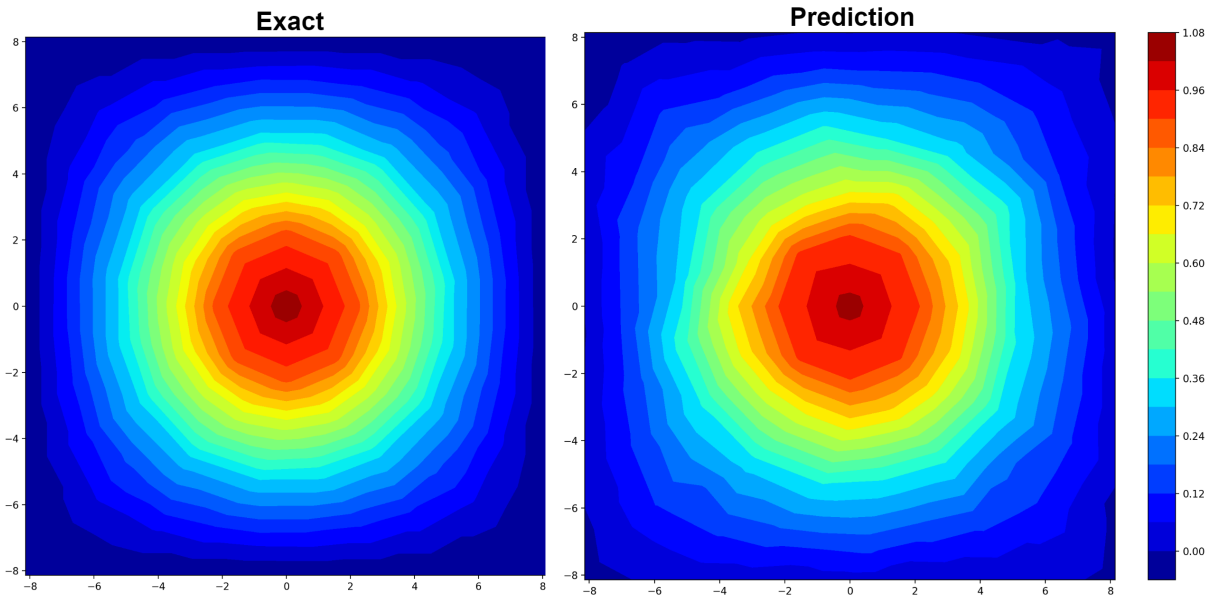

Figure 4.2: FEM solution (ground truth) of the magnetic field distribution for the coil problem with linear material, alongside the predicted magnetic field distribution from the single-model RNN architecture. The color scale represents the normalized magnetic field, B values.

suffer the most.

Nevertheless, the average MSE rate of the RNN models was found to be around $3-6\%$ of the ground truth. Both the models were trained using batch sizes of 16 and 32, and learning rates of 0.01 and 0.001. Each epoch took around $3-6$ *minutes* to complete depending upon the combination of hyperparameters used. Even though the models were trained for 50 epochs, convergence was achieved much earlier. Hence, the total training time for a model was about $150-300$ *minutes*. The prediction time was clocked at around 1.335 *ms* and 1.031 *ms* for the whole geometry for the single RNN and the separate RNN architectures, respectively. The performances are tabulated in Table 4.3.

Problem	**Coil with Linear Material**		
Models	*Error*	*Training Time*	*Prediction Time (whole geometry)*
RNN Model for all domains	$\sim 3\%$	$150-300$ *mins*	1.335 *ms*
RNN Models for each domain	$\sim 4\%$	$150-300$ *mins*	1.031 *ms*

Table 4.3: RNN performance measure

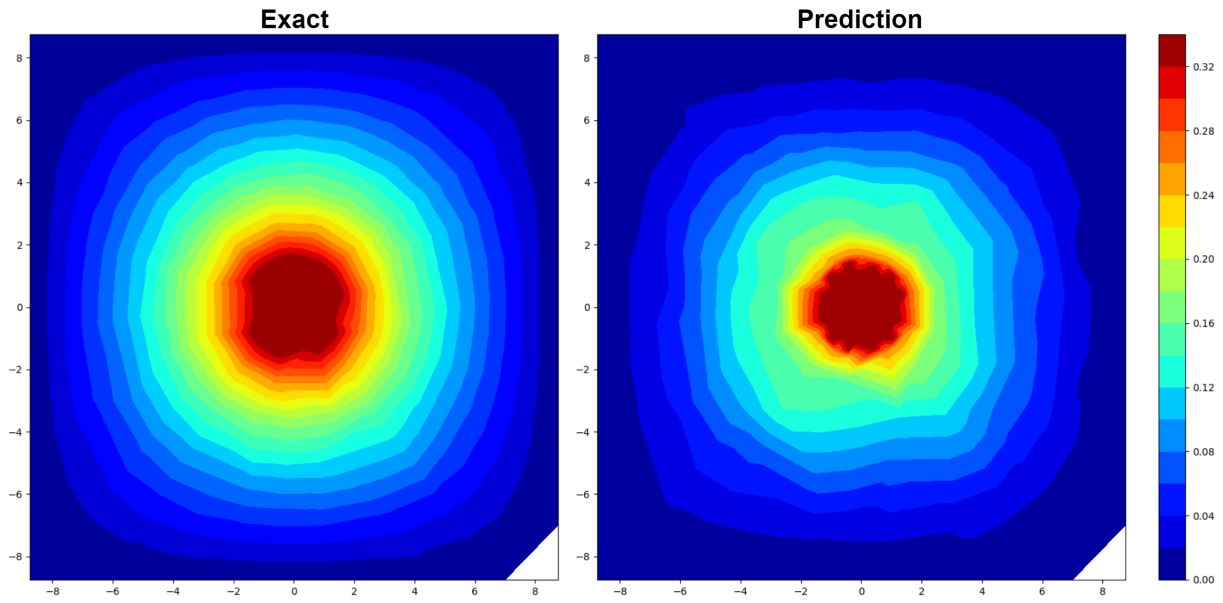

Figure 4.3: FEM solution (ground truth) of the magnetic field distribution for the coil problem with linear material, alongside the predicted magnetic field distribution from the separate-model RNN architecture. The color scale represents the normalized magnetic field, B values.

When compared to that of the CNN model [5, 74], as discussed in Section 4.1, the performances of the RNN models show some significant improvement. With similar level of accuracy of about $2.5 - 3\%$, the RNNs train much faster than the CNN [175], by cutting down the average training time by more than 72%, as provided in Table 4.1. Furthermore, the RNNs can generate predictions around 18 times faster than the CNN. The DL models are trained in batches, and the CNN model could only be trained with a smaller number of samples per batch (i.e., small batch size), as compared to an RNN-based model. The RNN model also provides the flexibility of being able to process non-uniform meshes of variable sizes without changing the geometric representation of the different problems due to its parameter sharing feature, discussed in Section 3.3.6. This also enables the RNN architecture to limit the number of trainable weights, and hence, use less computation memory while training and storing the model [175].

4.2.2 PI-RNN Model Performance

The PI-RNN model, discussed in Section 3.4.5, was used for the coil problem with both linear and non-linear magnetic materials, and also for the C-core problem with linear magnetic materials,

Problem	**Coil with Linear Material**		**Coil with Non-Linear Material**	
Base Models for PI-RNN	*RNN for Coil*	*RNN for Domain*	*RNN for Coil*	*RNN for Domain*
No. of Layers	3	4	3	4
No. of Neurons in each Layer	300	400	300	400
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid
Problem	**C-core with Linear Material**			
Base Models for PI-RNN	*RNN for Left Coil*	*RNN for Right Coil*	*RNN for Domain*	*RNN for Core/ Armature*
No. of Layers	6	6	6	6
No. of Neurons in each Layer	150	150	300	450
Activation Function	Sigmoid	Sigmoid	Sigmoid	Sigmoid

Table 4.4: PI-RNN model hyperparameter configurations

discussed in Sections 3.2.1 and 3.2.2, respectively. The network configuration for each problem is tabulated in Table 4.4, although the input and output layers were of fixed sizes for all the models — 2 neurons and 1 neuron, respectively. However, unlike the RNN model, only separate models for each different material were used for PI-RNNs. The reason behind that is because each material has a different physical property, which has to be modelled accordingly. The input data consisted of only the coordinates, (x, y) of the geometry for the respective materials, and the output was the magnetic field value, $\hat{u}(x, y)$ at the corresponding input point. The magnetic properties of the materials, μ_r, and the current excitation, J were all incorporated in the PDEs and BC used in the loss function. The data points were normalized, and hence, a Sigmoid activation function was used in the output layers. Furthermore, adaptive activation functions, which consist of a trainable parameter that speeds up the model's convergence [214], were used in all other layers. The training epochs were limited to $20,000$ to put a constraint on the resources being used. Hyperparameter optimization for PI-RNNs was performed using the Ray framework [215]. The Asynchronous Successive Halving Algorithm (ASHA) was chosen for the tuning purpose, which combines the random search with principled early stopping in an asynchronous way [216].

The code was entirely developed in the Python 3.8.13 programming language [213] using PyTorch 1.12.1 framework [145] and CUDA Toolkit 11.3.1. The training graph, as shown in Figure 4.4,

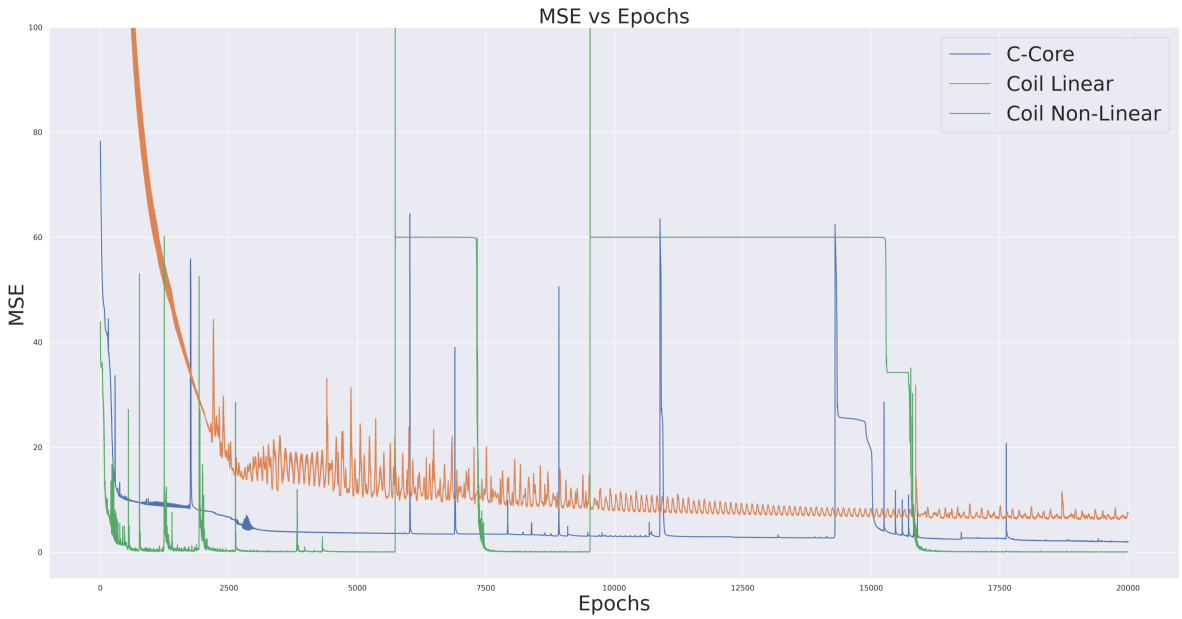

Figure 4.4: MSE plots for the PI-RNNs for all three problems — the linear coil, the non-linear coil, and the C-core.

provides evidence of convergence. To better visualize the performance of the models, contour plots of the predicted magnetic field, B values were compared to those using the actual values from the commercial FEM solver. The contour plots for the linear coil, non-linear coil, and the C-core problems are shown in Figures 4.6, 4.7, and 4.8, respectively. The color scale shows the normalized intensity of the magnetic field, B $[0, 1]$ across all the different domains for both the coil and C-core problems. Although the unit for the magnetic field, B is T $(tesla)$, the output values of the models are unitless since they are all normalized. It can be seen in the figures, that the predicted plots are very similar to the actual plots, for all the problems, which proves that the PI-RNNs were able to closely match the ground truth. Furthermore, contour plots showing the error distribution at the nodal coordinates of the FEM mesh were also generated, as given in Figures 4.6(c), 4.7(c), and 4.8(c). The color scale represents the absolute error values of the predicted field relative to the ground truth from the FEM solver.

For both the coil problems, the field predictions are almost perfect for the core and the domain near the boundary. As discussed in Section 4.2.1, the nodes near the interfaces suffer the most inaccuracies during the model prediction. However, for the non-linear problem the error near the interface is lower than that for the linear one. The average error rate for the linear problem was

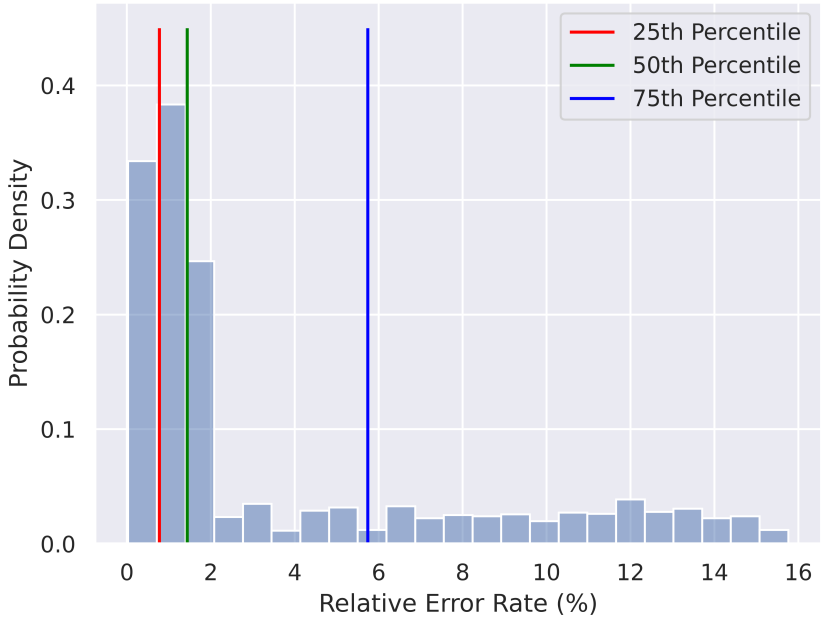

Figure 4.5: Histogram of the relative error (%), along with the error percentiles for PI-RNN for the non-linear coil problem.

around 6%, whereas for the non-linear problem it was around 3.5%, as shown in Table 4.5. This can be attributed to the well-defined modeling of the non-linear property of the material in the loss function of the PI-RNN architecture, as discussed in Section 3.4.5.

The C-core problem is a much more challenging one, incorporating three different domains and six interfaces. Nevertheless, the PI-RNN model was able to predict the field values with an average error rate of about 3.6%, as shown in Table 4.5, which is almost comparable to that of the non-linear coil problem. However, the domain to suffer the most inaccuracies during the prediction was the C-core, which is made of iron. The most probable reason is because of its three different interfaces with the "go" conductor, the "return" conductor, and the domain, as described in Section 3.2.2. Furthermore, the model only uses unsupervised learning for this part of the domain, which makes it even more difficult.

The average error for all the mesh points was found to be around 3.5% for both the coil and the C-core problems. The histogram of the relative error (%) along with the percentiles for the non-linear coil problem is shown in Figure 4.5. From the plot, the 25^{th}, the 50^{th}, and the 75^{th} percentiles can be observed to be around 0.75%, 1.35%, and 5.70%, respectively. For both the coil

problems, each training process took around $60 - 90$ *minutes* to complete, as shown in Table 4.5, while using learning rates ranging from 0.0008 to 0.01. However, the training time for the C-core problem was about $90 - 120$ *minutes*, as shown in Table 4.5, while using learning rates ranging from 0.0001 to 0.01. This is due to the bigger model architecture used for the problem, as given in Table 4.4, in order to capture its higher complexities. The prediction time was clocked at 1.527 ms, 1.519 ms, and 2.975 ms for the whole geometry for the linear coil, non-linear coil, and linear C-core problems, respectively. The performances are tabulated in Table 4.5.

Problem	**Error**	**Training Time**	**Prediction Time (whole geometry)**
Coil with Linear Material	$\sim 6\%$	$60 - 90$ *mins*	1.527 ms
Coil with Non-Linear Material	$\sim 3.5\%$	$60 - 90$ *mins*	1.519 ms
C-core with Linear Material	$\sim 3.6\%$	$90 - 120$ *mins*	2.975 ms

Table 4.5: PI-RNN performance measure

Nevertheless, for the coil problems, the PI-RNN model was able to cut down the average training time of the RNN model by a further 66.7%, as provided in Table 4.3, and that of the benchmark CNN model [175] by over 90%, as provided in Table 4.1. This shows that hybrid learning is much faster than a supervised one. This can be attributed to the much smaller amount of data required to train the PI-RNN, when compared to the RNN model. Specifically, only the boundary and interface data points were needed for the hybrid method, since points in the domain could be learnt just from the PDEs. Furthermore, the predictions generated by the PI-RNN model was about 16 times faster than the CNN.

For the C-core problem, two variants of the PI-RNN were tested — one with separate RNNs for the C-core and the armature, and the other with a single RNN for both. Now, the C-core and the armature are made of the same material, iron, which means that the underlying PDEs are the same. Theoretically, a single RNN should be able to approximate the field values for a specific material regardless of its spatial position. The experimental results support the hypothesis, i.e., the design with a single RNN for both the C-core and the armature produced similar output to the one with separate RNNs. This shows the potential for generalizability of PI-RNNs with the assistance of transfer learning.

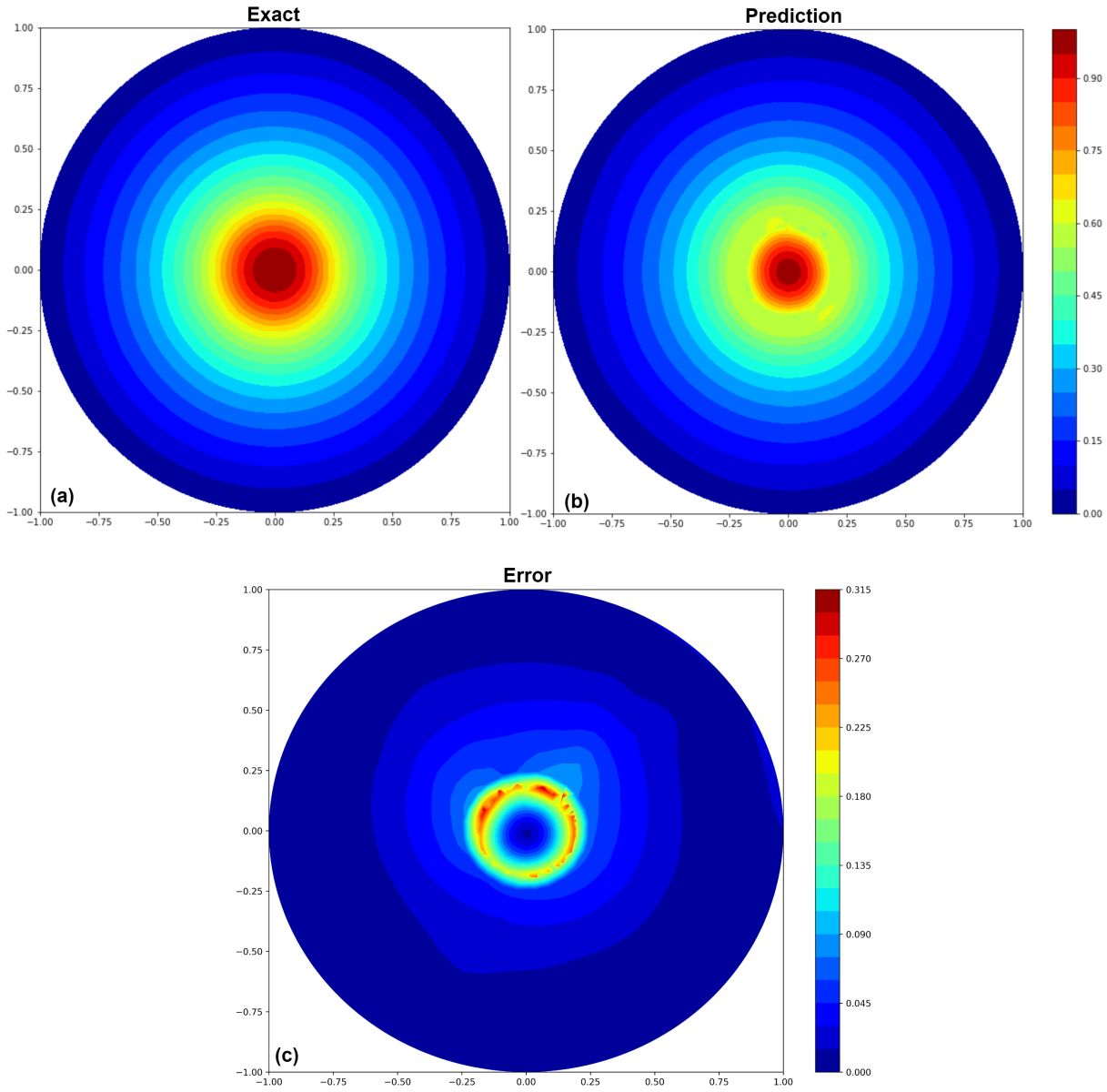

Figure 4.6: (a) FEM solution (ground truth) of the magnetic field distribution for the coil problem with linear material, and (b) the predicted magnetic field distribution from the PI-RNN architecture. The color scale represents the normalized magnetic field, B values. (c) The absolute error distribution of the predicted values relative to the ground truth. The color scale represents the absolute error values.

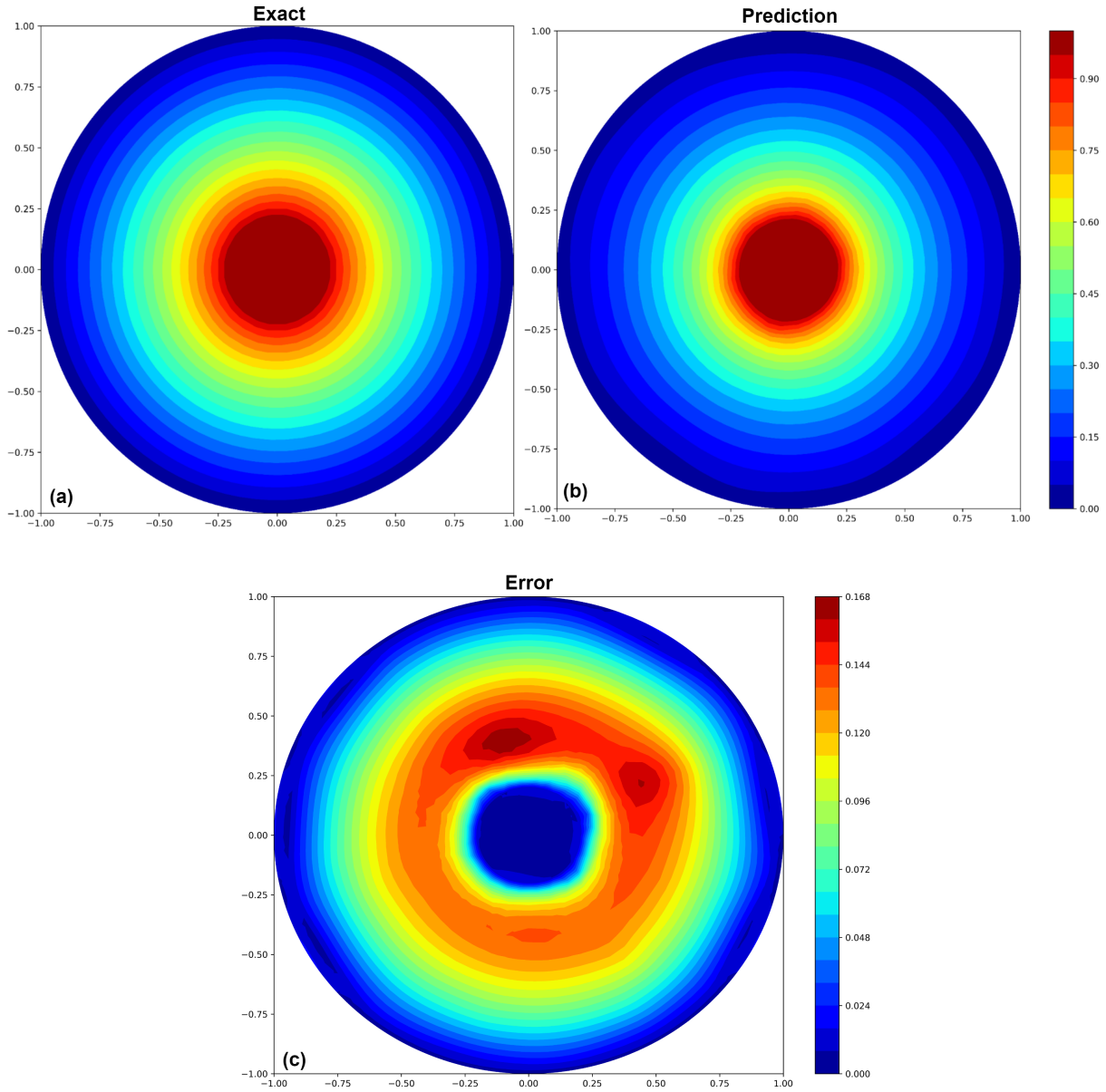

Figure 4.7: (a) FEM solution (ground truth) of the magnetic field distribution for the coil problem with non-linear material, and (b) the predicted magnetic field distribution from the PI-RNN architecture. The color scale represents the normalized magnetic field, B values. (c) The absolute error distribution of the predicted values relative to the ground truth. The color scale represents the absolute error values.

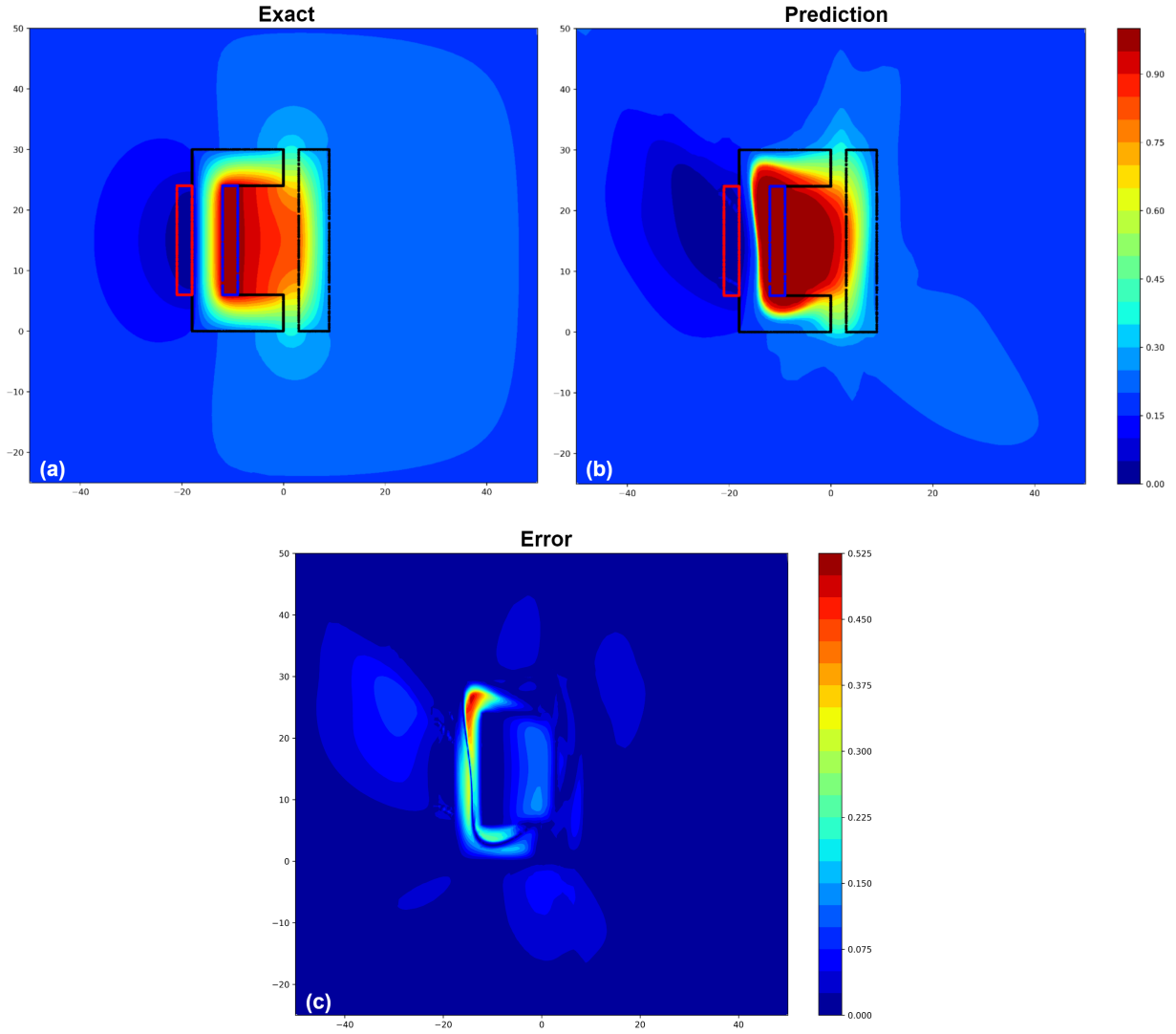

Figure 4.8: (a) FEM solution (ground truth) of the magnetic field distribution for the C-core problem, and (b) the predicted magnetic field distribution from the PI-RNN architecture. The color scale represents the normalized magnetic field, B values. (c) The absolute error distribution of the predicted values relative to the ground truth. The color scale represents the absolute error values.

5

Conclusion and Future Work

5.1 Conclusion

The goal of this research was to explore the two different fields of study — Computational Electromagnetics (CEM) and Machine Learning (ML), and combine them in a harmonious way so as to benefit both the fields. In this thesis, a thorough review of the current literature in both the fields have been discussed, and how one field can have a significant impact on the other. Conventional Finite Element Analysis (FEA) software have high computation requirements which limits the design and analysis of EM systems. The intent was to examine both a data-driven and a hybrid approach to simulate and solve EM problems, and evaluate their efficiencies so that they could be used as preliminary estimators before resorting to the FEA solvers. Both the approaches were accomplished using novel Deep Learning (DL) models, and were tested using different problems of varying complexities.

In this work, I have shown two new DL architectures — RNN and PI-RNN, for magnetic field evaluation. The RNN models were trained using a purely supervised (data-driven) learning technique, while a hybrid approach, involving both supervised and unsupervised learning methods, was used for the PI-RNN architectures. The magnetic field distribution was predicted using the models for three different scenarios: coil in a linear magnetic material, coil in a non-linear magnetic material, and C-core with a linear magnetic material. Trained on the actual data from the FEA software, both the RNN and PI-RNN models were able to efficiently estimate the field values. This will allow the DL models to make predictions on unseen new problems with varying geometries while incurring lower computational costs. Furthermore, the models can be run using GPUs, allowing the process to be parallelized for enhanced efficiency. Thus, the DL models can be used as surrogates for the solutions obtained using the conventional FEA solvers.

The performance of the RNN model was compared to that of another supervised learning model (CNN), which was used as a benchmark, and the RNN showed significant improvement in terms of training time with similar accuracy, while using less computational resources. Furthermore, its prediction time was much lower than that of the benchmark CNN model.

Moreover, the PI-RNN's hybrid learning method has been shown to improve the training period even further, while maintaining the accuracy level. PI-RNN was also able to evaluate the magnetic field distribution for a much more complex problem (C-core) with similar training time and accuracy, while using only the boundary and interface data points for supervised learning. This approach significantly cuts down the data dependency as compared to the data-driven model, making it more robust to changes in the geometries of any EM problem.

Even though the experimental results have been promising, I still cannot conclude if a PI-RNN can be used as a replacement for the traditional FEM solvers. The primary issue being the interface points. The boundary point values can be computed for any problem with a well-posed PDE with some form of BC (such as a Dirichlet BC), but the interface labels used for the supervised learning part are much more challenging to acquire. In this work, those were collected from a traditional FEM solver to begin with.

Nonetheless, the work done in this thesis can definitely serve as a guide to designing and developing DL architectures for future work in the field of CEM. Furthermore, it serves as a concrete example of showcasing the versatility of ML, especially DL, through its application in this field.

The architectures explored in this thesis can be applied to other fields as well. Finally, it can be used as a new benchmark for any future work that incorporates any data-driven or hybrid approach for CEM.

5.2 Future Work

For future work, I would like to test the potential for generalizability of PI-RNNs that we had a glimpse of in this work. The procedure would involve using a model, trained for a specific material on a particular problem, for predicting the solution for the same material but on a different unseen problem using transfer learning. If the process becomes successful, it will further reduce the computation time and resources by cutting down the training process.

Bibliography

[1] M. M. Rahman, A. Khan, D. Lowther, and D. Giannacopoulos, "Evaluating magnetic fields using deep learning," *COMPEL - The international journal for computation and mathematics in electrical and electronic engineering*, 8 2023.

[2] G. Pelosi, A. Savini, and S. Selleri, "A Brief History of Computational Electromagnetics," in *2021 7th IEEE History of Electrotechnology Conference (HISTELCON)*, 2021, pp. 81–84.

[3] M. N. Sadiku, *Numerical Techniques in Electromagnetics with MATLAB®*. CRC Press, 9 2015.

[4] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *ArXiv e-prints*, 3 2016.

[5] A. Khan, V. Ghorbanian, and D. Lowther, "Deep Learning for Magnetic Field Estimation," *IEEE Transactions on Magnetics*, vol. 55, no. 6, pp. 1–4, 2019.

[6] "WWDC23 - Apple Developer." [Online]. Available: https://developer.apple.com/wwdc23/

[7] "Google I/O 2023." [Online]. Available: https://io.google/2023/

[8] "Tesla Announces Date for 2023 Investor Day | Tesla Investor Relations." [Online]. Available: https://ir.tesla.com/press-release/tesla-announces-date-2023-investor-day

[9] "2023 Investor Day - YouTube." [Online]. Available: https://www.youtube.com/watch?v=Hl1zEzVUV7w

[10] "Avatar: The Way of Water (2022) - IMDb." [Online]. Available: https://www.imdb.com/title/tt1630029/

[11] "God of War Ragnarök." [Online]. Available: https://www.playstation.com/en-ca/games/god-of-war-ragnarok/

[12] "James Webb Space Telescope Science | NASA." [Online]. Available: https://www.nasa.gov/mission_pages/webb/science/index.html

[13] J. Banks, J. S. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation*, 5th ed. Prentice Hall, 2010.

[14] P. Bratley, B. L. Fox, and L. E. Schrage, *A Guide to Simulation*, 2nd ed. Springer-Verlag, 1983.

[15] D. Goldsman, R. Nance, and J. Wilson, "A brief history of simulation," in *Proceedings - Winter Simulation Conference*, 8 2010, pp. 310 – 313.

[16] E. Winsberg, "Computer Simulations in Science," *The Stanford Encyclopedia of Philosophy*, 2022.

[17] "Introduction to Simulation and Modeling: Historical Perspective." [Online]. Available: https://uh.edu/~lcr3600/simulation/historical.html

[18] "Oppenheimer (2023) - IMDb." [Online]. Available: https://www.imdb.com/title/tt15398776/

[19] "Manhattan Project | Definition, Scientists, Timeline, Locations, Facts, & Significance | Britannica." [Online]. Available: https://www.britannica.com/event/Manhattan-Project

[20] "Need To Know Webb/NASA." [Online]. Available: https://webb.nasa.gov/content/webbLaunch/needToKnow.html#webbImages

[21] "Key Facts - Webb/NASA." [Online]. Available: https://webb.nasa.gov/content/about/faqs/facts.html

[22] "Designing the James Webb Space Telescope with Simulation." [Online]. Available: https://www.ansys.com/blog/designing-the-james-webb-space-telescope-with-simulation

[23] "The Sunshield Webb/NASA." [Online]. Available: https://webb.nasa.gov/content/observatory/sunshield.html

[24] "How Ansys Helped the James Webb Space Telescope Be Destined for Success | Engineering.com." [Online]. Available: https://www.engineering.com/story/how-ansys-helped-the-james-webb-space-telescope-be-destined-for-success

[25] S. Yoon, J. Rosales, and K. Richon, "James Webb Space Telescope Orbit Determination Analysis," in *24th International Symposium on Space Flight Dynamics*, 8 2014.

[26] D. Dichmann, C. Alberding, and W. Yu, "STATIONKEEPING MONTE CARLO SIMULATION FOR THE JAMES WEBB SPACE TELESCOPE," in *International Symposium on Space Flight Dynamics (ISSFD)*, 8 2014.

[27] J. Gersh-Range, W. Arnold, M. Peck, and H. Stahl, "A Parametric Finite-Element Model for Evaluating Segmented Mirrors with Discrete, Edgewise Connectivity," *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 8125, 8 2011.

[28] "After two decades, the Webb telescope is finished and on the way to its launch site – Spaceflight Now." [Online]. Available: https://spaceflightnow.com/2021/09/30/webb-on-the-way-to-french-guiana/

[29] "NASA's James Webb Space Telescope arrives in French Guiana ahead of Dec. 18 launch | Space." [Online]. Available: https://www.space.com/nasa-james-webb-space-telescope-arrives-french-guiana

[30] "Simulation Made the James Webb Space Telescope Possible | Engineering.com." [Online]. Available: https://www.engineering.com/story/simulation-made-the-james-webb-space-telescope-possible

[31] "Using Femap helps NASA develop next-generation space telescope." [Online]. Available: https://resources.sw.siemens.com/en-US/case-study-nasa-goddard-space-flight-center

[32] "What is Coefficient of Thermal Expansion (CTE)? How Do I Measure It?" [Online]. Available: https://ctherm.com/resources/newsroom/blog/coefficient-of-thermal-expansion/

[33] "Where Is Webb? NASA/Webb." [Online]. Available: https://webb.nasa.gov/content/webbLaunch/whereIsWebb.html

[34] "10 Facts About the Airbus A380 - Nauman Saleem." [Online]. Available: https://naumansaleem.com/10-facts-about-the-airbus-a380/

[35] "Airbus Develops Fuel Management System for the A380 Using Model-Based Design - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/company/user_stories/airbus-develops-fuel-management-system-for-the-a380-using-model-based-design.html

[36] "Competition between Airbus and Boeing - Wikipedia." [Online]. Available: https://en.wikipedia.org/wiki/Competition_between_Airbus_and_Boeing

[37] "Airbus A380: Most Up-to-Date Encyclopedia, News & Reviews." [Online]. Available: https://academic-accelerator.com/encyclopedia/airbus-a380

[38] S. Grihon, "A380 Weight Savings using Numerical Structural Optimisation," in *AAAF Materials for Aerospace Applications 20th AAAF Colloquium*, Paris, 9 2003.

[39] R. J. Madachy and D. X. Houston, *What Every Engineer Should Know About Modeling and Simulation*. CRC Press, 9 2017.

[40] J. Banks, Ed., *Handbook of Simulation*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 8 1998.

[41] "The significance of simulations | Science News Learning." [Online]. Available: https://www.sciencenews.org/learning/guide/component/the-significance-of-simulations

[42] A. Khan, C. Midha, and D. A. Lowther, "Sequence-Based Environment for Topology Optimization," *IEEE Transactions on Magnetics*, vol. 56, no. 3, pp. 1–4, 3 2020.

[43] A. Khan, C. Midha, and D. Lowther, "Reinforcement Learning for Topology Optimization of a Synchronous Reluctance Motor," *IEEE Transactions on Magnetics*, vol. 58, no. 9, pp. 1–4, 9 2022.

[44] "How 'Avatar: The Way of Water' Solved the Problem of CGI Water - The New York Times." [Online]. Available: https://www.nytimes.com/2022/12/16/movies/avatar-2-fx-cgi.html

[45] "What Are Simulation Games | Simulation Video Games | Music Gateway." [Online]. Available: https://www.musicgateway.com/blog/gaming-industry/games-business/what-are-simulation-games

[46] "Simulation in Cinema." [Online]. Available: https://openjournals.uwaterloo.ca/index.php/kinema/article/download/1072/1234?inline=1

[47] "Compiled models vs. first principles models for fault detection and diagnosis." [Online]. Available: https://gregstanleyandassociates.com/whitepapers/FaultDiagnosis/Compiled-Models/compiled-models.htm

[48] "Chemical process models – from first principle to hybrid models." [Online]. Available: https://blog.navigance.com/chemical-process-models

[49] "First-Principle Versus Data-Driven Models | Control Global." [Online]. Available: https://www.controlglobal.com/manage/asset-management/article/11382297/first-principle-versus-data-driven-models

[50] M. K. Habib, S. A. Ayankoso, and F. Nagata, "Data-Driven Modeling: Concept, Techniques, Challenges and a Case Study," in *2021 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, 8 2021, pp. 1000–1007.

[51] S. Kurz, H. De Gersem, A. Galetzka, A. Klaedtke, M. Liebsch, D. Loukrezis, S. Russenschuck, and M. Schmidt, "Hybrid modeling: towards the next level of scientific computing in engineering," *Journal of Mathematics in Industry*, vol. 12, no. 1, p. 8, 12 2022.

[52] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2 2019.

[53] 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," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113028, 6 2020.

[54] "Most Popular Electronics Worldwide [October 2022 Update]." [Online]. Available: https://www.oberlo.ca/statistics/most-popular-electronics

[55] "BBC News - Meet Marty Cooper - the inventor of the mobile phone." [Online]. Available: http://news.bbc.co.uk/1/hi/programmes/click_online/8639590.stm

[56] "Guglielmo Marconi – Facts - NobelPrize.org." [Online]. Available: https://www.nobelprize.org/prizes/physics/1909/marconi/facts/

[57] P. K. Bondyopadhyay, "Guglielmo Marconi - The father of long distance radio communication - An engineer's tribute," in *1995 25th European Microwave Conference*, vol. 2, 1995, pp. 879–885.

[58] B. M. Notaros, *Electromagnetics*. Prentice Hall, 2011. [Online]. Available: https://books.google.ca/books?id=alBXAAAACAAJ

[59] P. G. Huray, *Maxwell's equations*. Hoboken, N.J. :: Wiley :, 2010.

[60] B. J. Hunt, "Oliver Heaviside: A first-rate oddity," *Physics Today*, vol. 65, no. 11, p. 48, 10 2012. [Online]. Available: https://physicstoday.scitation.org/doi/abs/10.1063/PT.3.1788

[61] L. D. Kovach, *Boundary-value problems*. Addison-Wesley Pub. Co., 1984. [Online]. Available: https://cir.nii.ac.jp/crid/1130282270231046912

[62] T. G. Campbell, R. A. Nicolaides, and M. D. Salas, Eds., *Computational Electromagnetics and Its Applications*. Dordrecht: Springer Netherlands, 1997, vol. 5.

[63] A. Taflove, S. C. Hagness, and M. Piket-May, "Computational electromagnetics: the finite-difference time-domain method," *The Electrical Engineering Handbook*, vol. 3, no. 629-670, p. 15, 2005.

[64] R. Mittra, *Computational electromagnetics*. Springer, 2016.

[65] P. P. Silvester, "Finite elements solution of homogeneous waveguide problems," *Alta Frequenza*, vol. 38, pp. 313–317, 1969.

[66] R. Ferrari, "The finite-element method, Part 2: P. P. Silvester, an innovator in electromagnetic numerical modeling," *IEEE Antennas and Propagation Magazine*, vol. 49, no. 3, pp. 216–234, 2007.

[67] D. R. Wilton, E. Arvas, C. M. Butler, and J. R. Mautz, "Roger F. Harrington, 1989 IEEE AP-S Distinguished Achievement awardee," in *2017 IEEE International Symposium on Antennas and Propagation & USNC/URSI National Radio Science Meeting*. IEEE, 7 2017, pp. 657–658.

[68] A. Taflove, "Application of the Finite-Difference Time-Domain Method to Sinusoidal Steady-State Electromagnetic-Penetration Problems," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-22, no. 3, pp. 191–202, 8 1980.

[69] "MagNet: 2D/3D Electromagnetic Field Simulation Software." [Online]. Available: http://www.infolytica.com/

[70] "Ansys Maxwell | Electromechanical Device Analysis Software." [Online]. Available: https://www.ansys.com/products/electronics/ansys-maxwell

[71] "Release History." [Online]. Available: https://www.comsol.com/release-history

[72] W. Tang, T. Shan, X. Dang, M. Li, F. Yang, S. Xu, and J. Wu, "Study on a Poisson's equation solver based on deep learning technique," in *2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, 2017, pp. 1–3.

[73] S. Barmada, N. Fontana, L. Sani, D. Thomopulos, and M. Tucci, "Deep Learning and Reduced Models for Fast Optimization in Electromagnetics," *IEEE Transactions on Magnetics*, vol. 56, no. 3, pp. 1–4, 3 2020.

[74] R. Gong and Z. Tang, "Investigation of convolutional neural network U-net under small datasets in transformer magneto-thermal coupled analysis," *COMPEL - The international journal for computation and mathematics in electrical and electronic engineering*, vol. 39, no. 4, pp. 959–970, 8 2020.

[75] "Ansys HFSS | 3D High Frequency Simulation Software." [Online]. Available: https://www.ansys.com/products/electronics/ansys-hfss

[76] "COMSOL - Software for Multiphysics Simulation." [Online]. Available: https://www.comsol.com/

[77] "CST Studio Suite 3D EM simulation and analysis software." [Online]. Available: https://www.3ds.com/products-services/simulia/products/cst-studio-suite/

[78] "History | CompEM Lab." [Online]. Available: https://www.compem.ece.mcgill.ca/history.html

[79] P. P. Silvester and R. L. Ferrari, *Finite Elements for Electrical Engineers*. Cambridge University Press, 9 1996.

[80] S. Selleri, "A Brief History of Finite Element Method and Its Applications to Computational Electromagnetics," *The Applied Computational Electromagnetics Society Journal (ACES)*, 11 2022.

[81] D. A. Lowther and P. P. Silvester, *Computer-Aided Design in Magnetics*. Berlin: New York : Springer-Verlag, 1986.

[82] "History of Ansoft Corporation – FundingUniverse." [Online]. Available: http://www.fundinguniverse.com/company-histories/ansoft-corporation-history/

[83] H. W. Ott, *Electromagnetic compatibility engineering*. John Wiley & Sons, 2009.

[84] "Computational electromagnetics | Electrical and Computer Engineering - McGill University." [Online]. Available: https://www.mcgill.ca/ece/research/cadlab

[85] "Magnetic Resonance Imaging (MRI) | Johns Hopkins Medicine." [Online]. Available: https://www.hopkinsmedicine.org/health/treatment-tests-and-therapies/magnetic-resonance-imaging-mri

[86] R. Kouyoumjian and P. Pathak, "A uniform geometrical theory of diffraction for an edge in a perfectly conducting surface," *Proceedings of the IEEE*, vol. 62, no. 11, pp. 1448–1461, 1974.

[87] H. M. Macdonald, "The Effect Produced by an Obstacle on a Train of Electric Waves," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 212, pp. 299–337, 1913. [Online]. Available: http://www.jstor.org/stable/91056

[88] P. Y. Ufimtsev, "The 50-Year Anniversary Of the PTD: Comments on the PTD's Origin and Development," *IEEE Antennas and Propagation Magazine*, vol. 55, no. 3, pp. 18–28, 6 2013.

[89] .-. Thom, A. (Alexander) and C. J. C. J. Apelt, *Field Computations in Engineering and Physics*. London: Van Nostrand, 1961.

[90] F. C. Trutt, "Analysis of homopolar inductor alternators," Ph.D. dissertation, University of Delaware, Newark, 1964.

[91] Kane Yee, "Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media," *IEEE Transactions on Antennas and Propagation*, vol. 14, no. 3, pp. 302–307, 5 1966.

[92] A. Taflove and M. Brodwin, "Numerical Solution of Steady-State Electromagnetic Scattering Problems Using the Time-Dependent Maxwell's Equations," *IEEE Transactions on Microwave Theory and Techniques*, vol. 23, no. 8, pp. 623–630, 8 1975.

[93] A. Taflove and K. R. Umashankar, "Solution of Complex Electromagnetic Penetration and Scattering Problems in Unbounded Regions," in *Computational Methods for Infinite Domain Media-Structure Interactions*. American Society of Mechanical Engineers, 1981, vol. 46, pp. 83–113.

[94] A. Taflove and K. Umashankar, "A hybrid moment method/finite-difference time-domain approach to electromagnetic coupling and aperture penetration into complex geometries," *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 4, pp. 617–627, 7 1982.

[95] K. Umashankar and A. Taflove, "A Novel Method to Analyze Electromagnetic Scattering of Complex Objects," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-24, no. 4, pp. 397–405, 11 1982.

[96] A. Taflove and K. R. Umashankar, "The Finite-Difference Time-Domain Method for Numerical Modeling of Electromagnetic Wave Interactions," *Electromagnetics*, vol. 10, no. 1-2, pp. 105–126, 1 1990.

[97] M. Okoniewski, "Vector wave equation 2-D-FDTD method for guided wave problems," *IEEE Microwave and Guided Wave Letters*, vol. 3, no. 9, pp. 307–309, 9 1993.

[98] J. P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," *Journal of Computational Physics*, vol. 114, no. 2, pp. 185–200, 10 1994.

[99] D. Katz, E. Thiele, and A. Taflove, "Validation and extension to three dimensions of the Berenger PML absorbing boundary condition for FD-TD meshes," *IEEE Microwave and Guided Wave Letters*, vol. 4, no. 8, pp. 268–270, 8 1994.

[100] A. Taflove, "A Perspective on the 40-Year History of FDTD Computational Electrodynamics," *The Applied Computational Electromagnetics Society Journal (ACES)*, vol. 22, no. 1, pp. 1–21, 6 2022. [Online]. Available: https://journals.riverpublishers.com/index.php/ACES/article/view/16191

[101] R. J. Leveque, "High resolution finite volume methods on arbitrary grids via wave propagation," *Journal of Computational Physics*, vol. 78, no. 1, pp. 36–63, 9 1988.

[102] N. K. Madsen and R. W. Ziolkowski, "A Three-Dimensional Modified Finite Volume Technique for Maxwell's Equations," *Electromagnetics*, vol. 10, no. 1-2, pp. 147–161, 1 1990.

[103] "Clemson Vehicular Electronics Laboratory: The Finite Volume Time Domain Method." [Online]. Available: https://cecas.clemson.edu/cvel/modeling/tutorials/techniques/fvtd/fvtd.html

[104] R. F. Harrington, *Field Computation by Moment Methods*. IEEE, 1993.

[105] E. M. Wright, "Approximate Methods of Higher Analysis. By L. V. Kantorovich and V. I. Krylov. Translated from the 3rd Russian Edition by C. D. Benster. Pp. 681. 1958. (Groningen, Noordhoff)," *The Mathematical Gazette*, vol. 44, no. 348, pp. 145–145, 5 1960.

[106] H. F. . Davis, "Method of Moments in Applied Mathematics, by Yu. V. Vorobyev. Translated from the Russian by Bernard Seckler. Gordon and Breach, New York, 1965. 168 pages." *Canadian Mathematical Bulletin*, vol. 10, no. 4, pp. 615–616, 10 1967.

[107] R. Harrington, "Origin and development of the method of moments for field computation," *IEEE Antennas and Propagation Magazine*, vol. 32, no. 3, pp. 31–35, 6 1990.

[108] J. Richmond, "Digital computer solutions of the rigorous equations for scattering problems," *Proceedings of the IEEE*, vol. 53, no. 8, pp. 796–804, 1965.

[109] R. Harrington, "Matrix methods for field problems," *Proceedings of the IEEE*, vol. 55, no. 2, pp. 136–149, 1967.

[110] C. Delgado, E. Garcia, J. Moreno, I. Gonzalez-Diego, and M. F. Catedra, "AN OVERVIEW OF THE EVOLUTION OF METHOD OF MOMENTS TECHNIQUES IN MODERN EM SIMULATORS (Invited Paper)," *Progress In Electromagnetics Research*, vol. 150, pp. 109–121, 2015.

[111] R. Courant, "Variational Methods for the Solution of Problems of Equilibrium and Vibration," *Bullettin of the American Mathematical Society*, vol. 49, pp. 1–23, 1943.

[112] G. Pelosi, "The finite-element method, Part I: R. L. Courant [Historical Corner]," *IEEE Antennas and Propagation Magazine*, vol. 49, no. 2, pp. 180–182, 2007.

[113] A. Cangellaris, Chung-Chi Lin, and Kenneth Mei, "Point-matched time domain finite element methods for electromagnetic radiation and scattering," *IEEE Transactions on Antennas and Propagation*, vol. 35, no. 10, pp. 1160–1173, 10 1987.

[114] R. Coccioli, T. Itoh, G. Pelosi, and P. Silvester, "Finite-element methods in microwaves: a selected bibliography," *IEEE Antennas and Propagation Magazine*, vol. 38, no. 6, pp. 34–48, 1996.

[115] M. N. O. Sadiku, "A simple introduction to finite element analysis of electromagnetic problems," *IEEE Transactions on Education*, vol. 32, no. 2, pp. 85–93, 1989.

[116] Inamuddin, R. Boddula, and A. M. Asiri, Eds., *Actuators and Their Applications: Fundamentals, Principles, Materials, and Emerging Technologies*. Wiley, 4 2020. [Online]. Available: https://books.google.ca/books?hl=en&lr=&id=6SnfDwAAQBAJ&oi=fnd&pg=PP2&dq=Actuators+Fundamentals,+Principles,+Materials,+and+Applications.&ots=nUkYM5MJwH&sig=ZfbffbTsPFb2iDOiieVZdi1l000#v=onepage&q=Actuators%20Fundamentals%2C%20Principles%2C%20Materials%2C%20and%20Applications.&f=false

[117] D. Howe, "Magnetic actuators," *Sensors and Actuators A: Physical*, vol. 81, no. 1-3, pp. 268–274, 4 2000.

[118] A. M. Pawlak, *Sensors and Actuators in Mechatronics*. CRC Press, 12 2017.

[119] D. Popovich, *Mechatronics in Engineering Design and Product Development*. CRC Press, 9 1998.

[120] A. Garcia, J. Cusidó, J. A. Rosero, J. A. Ortega, and L. Romeral, "Reliable electro-mechanical actuators in aircraft," *IEEE Aerospace and Electronic Systems Magazine*, vol. 23, no. 8, pp. 19–25, 8 2008.

[121] "Electromagnetic Actuators in Mechatronics | System Analysis Blog | Cadence." [Online]. Available: https://resources.system-analysis.cadence.com/blog/msa2021-electromagnetic-actuators-in-mechatronics

[122] "Electromagnetic Actuators - ElectronicsHub." [Online]. Available: https://www.electronicshub.org/electromagnetic-actuators/

[123] S. Khalid, F. Khan, Z. Ahmad, and B. Ullah, "Design and Analysis of Modular C-core Moving Magnet Linear Oscillating Actuator for Miniature Compressor Application," in *2022 Joint MMM-Intermag Conference (INTERMAG)*. IEEE, 1 2022, pp. 1–5.

[124] ——, "Design and finite element analysis of modular C-Core stator tubular linear oscillating actuator for miniature compressor," *World Journal of Engineering*, vol. 20, no. 2, pp. 266–272, 1 2023. [Online]. Available: https://doi.org/10.1108/WJE-03-2021-0142

[125] S. I. Park and S. Min, "Magnetic actuator design for maximizing force using level set based topology optimization," *IEEE Transactions on Magnetics*, vol. 45, no. 5, pp. 2336–2339, 2009.

[126] H. Zhang, L. Jin, H. Yu, Z. Xu, X. Wei, J. Leng, and S. Fang, "Detent Force Reduction Design for the C-Core Single-Phase Permanent Magnet Linear Oscillation Actuator," *IEEE Transactions on Industry Applications*, vol. 59, no. 2, pp. 1577–1587, 3 2023.

[127] N. J. Nilsson and N. J. Nilsson, *Artificial intelligence: a new synthesis.* Morgan Kaufmann, 1998.

[128] "John McCarthy - A.M. Turing Award Laureate." [Online]. Available: https://amturing.acm.org/award_winners/mccarthy_1118322.cfm

[129] J. Mccarthy, "WHAT IS ARTIFICIAL INTELLIGENCE?" 11 2007. [Online]. Available: http://www-formal.stanford.edu/jmc/whatisai/whatisai.html

[130] M. Awad and R. Khanna, *Efficient learning machines: Theories, concepts, and applications for engineers and system designers.* Apress Media LLC, 1 2015.

[131] A. Géron, *Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. O'Reilly Media, 2017. [Online]. Available: https://books.google.ca/books?id=I6qkDAEACAAJ

[132] T. M. Mitchell, *Machine Learning.* McGraw-Hill Science/Engineering/Math, 3 1997.

[133] W. S. Mcculloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," *BULLETIN OF MATHEMATICAL BIOPHYSICS*, vol. 5, pp. 115–133, 1943.

[134] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory.* Oxford, England: Wiley, 1949.

[135] A. M. Turing, "Computing Machinery and Intelligence," *Mind, New Series*, vol. 59, no. 236, pp. 433–460, 10 1950. [Online]. Available: https://www.jstor.org/stable/2251299

[136] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 7 1959. [Online]. Available: http://ieeexplore.ieee.org/document/5392560/

[137] "Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol - BBC News." [Online]. Available: https://www.bbc.com/news/technology-35785875

[138] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 4 1980.

[139] "Yann LeCun - A.M. Turing Award Laureate." [Online]. Available: https://amturing.acm.org/award_winners/lecun_6017366.cfm

[140] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 11 1998.

[141] "Geoffrey E Hinton - A.M. Turing Award Laureate." [Online]. Available: https://amturing.acm.org/award_winners/hinton_4791679.cfm

[142] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A Fast Learning Algorithm for Deep Belief Nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 7 2006.

[143] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed
}Systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

[144] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermüller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, Y. Bengio, A. Bergeron, J. Bergstra, V. Bisson, J. B. Snyder, N. Bouchard, N. Boulanger-Lewandowski, X. Bouthillier, A. de Brébisson, O. Breuleux, P. L. Carrier, K. Cho, J. Chorowski, P. F. Christiano, T. Cooijmans, M.-A. Côté, M. Côté, A. C. Courville, Y. N. Dauphin, O. Delalleau, J. Demouth, G. Desjardins, S. Dieleman, L. Dinh, M. Ducoffe, V. Dumoulin, S. E. Kahou, D. Erhan, Z. Fan, O. Firat, M. Germain, X. Glorot, I. J. Goodfellow, M. Graham, C. Gulcehre, P. Hamel, I. Harlouchet, J.-P. Heng, B. Hidasi, S. Honari, A. Jain, S. Jean, K. Jia, M. Korobov, V. Kulkarni, A. Lamb, P. Lamblin, E. Larsen, C. Laurent, S. Lee, S. Lefrançois, S. Lemieux, N. Léonard, Z. Lin, J. A. Livezey, C. Lorenz, J. Lowin, Q. Ma, P.-A. Manzagol, O. Mastropietro, R. McGibbon, R. Memisevic, B. van Merriënboer, V. Michalski, M. Mirza, A. Orlandi, C. J. Pal, R. Pascanu, M. Pezeshki, C. Raffel, D. Renshaw, M. Rocklin, A. Romero, M. Roth, P. Sadowski, J. Salvatier, F. Savard, J. Schlüter, J. Schulman, G. Schwartz, I. V. Serban, D. Serdyuk, S. Shabanian, E. Simon, S. Spieckermann, S. R. Subramanyam, J. Sygnowski, J. Tanguay, G. van Tulder, J. P. Turian, S. Urban, P. Vincent, F. Visin, H. de Vries, D. Warde-Farley, D. J. Webb, M. Willson, K. Xu, L. Xue, L. Yao, S. Zhang, and Y. Zhang, "Theano: A Python framework for fast computation of mathematical
}expressions," *CoRR*, vol. abs/1605.02688, 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[145] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," *NIPS 2017 Workshop on Autodiff*, 2017.

[146] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 12 2015, pp. 1026–1034.

[147] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language
}Understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[148] E. Jurczenko, *Machine Learning for Asset Management New Developments and Financial Applications*. London :: Wiley-ISTE„ 2020.

[149] F. Pedregosa FABIANPEDREGOSA, V. Michel, O. Grisel OLIVIERGRISEL, M. Blondel, P. Prettenhofer, R. Weiss, J. Vanderplas, D. Cournapeau, F. Pedregosa, G. Varoquaux, A. Gramfort, B. Thirion, O. Grisel, V. Dubourg, A. Passos, M. Brucher, M. Perrot andÉ-douardand, a. Duchesnay, and F. Duchesnay EDOUARDDUCHESNAY, "Scikit-learn:

Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011. [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html

[150] W. McKinney and others, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, 2010, pp. 51–56.

[151] Y. Bengio and Y. LeCun, "Scaling Learning Algorithms towards AI," *Large-Scale Kernel Machines*, 2007.

[152] O. Delalleau and Y. Bengio, "Shallow vs. Deep Sum-Product Networks," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., vol. 24. Curran Associates, Inc., 2011. [Online]. Available: https://proceedings.neurips.cc/paper/2011/file/8e6b42f1644ecb1327dc03ab345e618b-Paper.pdf

[153] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[154] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, pp. 1097–1105.

[155] F. Rosenblatt, "The Perceptron, a Perceiving and Recognizing Automaton (Project Para)," Cornell Aeronautical Laboratory, Buffalo, Tech. Rep., 1957. [Online]. Available: https://books.google.ca/books?id=P_XGPgAACAAJ

[156] ——, *Principles of neurodynamics; perceptrons and the theory of brain mechanisms.* Washington: Spartan Books, 1962. [Online]. Available: //catalog.hathitrust.org/Record/000203591http://hdl.handle.net/2027/mdp.39015039846566

[157] Y. Lecun and Y. Bengio, "Pattern recognition and neural networks," *The Handbook of Brain Theory and Neural Networks. MIT Press, Cambridge, MA*, 1995.

[158] B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge University Press, 1 1996. [Online]. Available: https://www.cambridge.org/core/product/identifier/9780511812651/type/book

[159] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1 1989.

[160] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 10 1986.

[161] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *International Conference for Learning Representations*, 12 2014.

[162] R. Hecht-Nielsen, *Neurocomputing*. USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[163] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 7 2012.

[164] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[165] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, pp. 448–456, 2 2015. [Online]. Available: https://arxiv.org/abs/1502.03167v3

[166] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch Normalized Recurrent Neural Networks," *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 2016-May, pp. 2657–2661, 10 2015. [Online]. Available: https://arxiv.org/abs/1510.01378v1

[167] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," 7 2016. [Online]. Available: https://arxiv.org/abs/1607.06450v1

[168] "The Nobel Prize in Physiology or Medicine 1981 - Press release." [Online]. Available: https://www.nobelprize.org/prizes/medicine/1981/press-release/

[169] D. H. Hubel, "Single unit activity in striate cortex of unrestrained cats," *The Journal of Physiology*, vol. 147, no. 2, pp. 226–238, 9 1959.

[170] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of Physiology*, vol. 148, no. 3, pp. 574–591, 10 1959.

[171] ——, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology*, vol. 195, no. 1, pp. 215–243, 1968. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1968.sp008455

[172] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[173] 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 (CVPR)*, 6 2015.

[174] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 6 2016, pp. 770–778.

[175] A. Khan, M. H. Mohammadi, V. Ghorbanian, and D. Lowther, "Efficiency Map Prediction of Motor Drives Using Deep Learning," *IEEE Transactions on Magnetics*, vol. 56, no. 3, pp. 1–4, 2020.

[176] Z. Liu, O. Mohammed, and Shuo Liu, "Equivalent Hardware Representation of PM Synchronous Motors From the Physics-Based Phase Variable Model Obtained Through FE Computation," *IEEE Transactions on Magnetics*, vol. 45, no. 3, pp. 1450–1453, 3 2009.

[177] N. R. Tavana and V. Dinavahi, "Real-Time Nonlinear Magnetic Equivalent Circuit Model of Induction Machine on FPGA for Hardware-in-the-Loop Simulation," *IEEE Transactions on Energy Conversion*, vol. 31, no. 2, pp. 520–530, 6 2016.

[178] B. Wang, T. Rahman, K. Chang, M. H. Mohammadi, and D. A. Lowther, "A neural network based surrogate model for predicting noise in synchronous reluctance motors," in *2016 IEEE Conference on Electromagnetic Field Computation (CEFC)*. IEEE, 11 2016, pp. 1–1.

[179] R. C. P. Silva, M. Li, T. Rahman, and D. A. Lowther, "Surrogate-based MOEA/D for electric motor design with scarce function evaluations," in *2016 IEEE Conference on Electromagnetic Field Computation (CEFC)*. IEEE, 11 2016, pp. 1–1.

[180] V. Ghorbanian and D. A. Lowther, "A Statistical Solution to Efficiently Optimize the Design of an Inverter-Fed Permanent-Magnet Motor," *IEEE Transactions on Industry Applications*, vol. 53, no. 6, pp. 5315–5326, 11 2017.

[181] V. Ghorbanian, A. Salimi, and D. A. Lowther, "A Computer-Aided Design Process for Optimizing the Size of Inverter-Fed Permanent Magnet Motors," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 2, pp. 1819–1827, 2 2018.

[182] M. Yilmaz and P. T. Krein, "Capabilities of finite element analysis and magnetic equivalent circuits for electrical machine analysis and design," in *2008 IEEE Power Electronics Specialists Conference*. IEEE, 6 2008, pp. 4027–4033.

[183] S. J. Salon, *Finite Element Analysis of Electrical Machines*. New York, NY: Springer US, 1995.

[184] N. Bianchi and S. Bolognani, "Design optimisation of electric motors by genetic algorithms," *IEE Proceedings - Electric Power Applications*, vol. 145, no. 5, p. 475, 1998.

[185] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," *CoRR*, vol. abs/1612.00593, 2016. [Online]. Available: http://arxiv.org/abs/1612.00593

[186] A. Khan and D. A. Lowther, "Physics Informed Neural Networks for Electromagnetic Analysis," *IEEE Transactions on Magnetics*, vol. 58, no. 9, 9 2022.

[187] "The Size and Quality of a Data Set | Machine Learning | Google Developers." [Online]. Available: https://developers.google.com/machine-learning/data-prep/construct/collect/data-size-quality

[188] "Solving Partial Differential Equations - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/help/matlab/math/partial-differential-equations.html

[189] "HomePage:Finite Element Method Magnetics." [Online]. Available: https://www.femm.info/wiki/HomePage

[190] M. Minsky and S. Papert, *Perceptrons: An introduction to computational geometry*. Cambridge, MA: MIT press, 1969, vol. 479.

[191] S. Hochreiter and J. Schmidhuber, "Long Short-term Memory," *Neural computation*, vol. 9, pp. 1735–1780, 10 1997.

[192] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *Conf. on Empirical Methods in NLP*, pp. 1724–1734, 6 2014.

[193] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," in *3rd International Conference on Learning Representations*. San Diego: ICLR, 5 2015.

[194] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," 8 2015.

[195] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *Advances in neural information processing systems*, pp. 5998–6008, 6 2017.

[196] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 6 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[197] G. Philipp, D. Song, and J. G. Carbonell, "The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions," 12 2017. [Online]. Available: https://arxiv.org/abs/1712.05577v4

[198] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations," 11 2017.

[199] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: a survey," *The Journal of Machine Learning Research*, pp. 1–43, 2 2015.

[200] V. Schäfer, D. M. Bracke, P. D. R. Pinnau, and D. r. n. M. Siggel, "Generalization of PINNs for Various Boundary and Initial Conditions," Ph.D. dissertation, University of Kaiserslautern, 1 2022.

[201] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A Limited Memory Algorithm for Bound Constrained Optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995. [Online]. Available: https://doi.org/10.1137/0916069

[202] 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," *Communications in Computational Physics*, vol. 28, no. 5, pp. 2042–2074, 4 2020.

[203] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000.

[204] M. W. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *Communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, 3 1994. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/cnm.1640100303https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303https://onlinelibrary.wiley.com/doi/10.1002/cnm.1640100303

[205] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998.

[206] B. Moseley, A. Markham, and T. Nissen-Meyer, "Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations," 7 2021. [Online]. Available: https://arxiv.org/abs/2107.07871v1

[207] E. Museljic, A. Reinbacher-Kostinger, and M. Kaltenbacher, "Solving the electrostatic Laplace's equation with a parameterizable physics informed neural network," in *2022 IEEE 20th Biennial Conference on Electromagnetic Field Computation (CEFC)*. IEEE, 10 2022, pp. 1–2.

[208] E. Kharazmi, Z. Zhang, and G. E. Karniadakis, "hp-VPINNs: Variational physics-informed neural networks with domain decomposition," *Computer Methods in Applied Mechanics and Engineering*, vol. 374, p. 113547, 2 2021.

[209] M. Baldan, P. Di Barba, and D. A. Lowther, "Physics-Informed Neural Networks for Inverse Electromagnetic Problems," in *2022 IEEE 20th Biennial Conference on Electromagnetic Field Computation (CEFC)*. IEEE, 10 2022, pp. 1–2.

[210] L. Sun, H. Gao, S. Pan, and J. X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112732, 4 2020.

[211] "Magnetic Field in Two-Pole Electric Motor: PDE Modeler App - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/help/pde/ug/magnetostatics.html

[212] A. Khan, *Statistical methods for design and analysis of electrical machines*. McGill University (Canada), 2021.

[213] "Welcome to Python.org." [Online]. Available: https://www.python.org/

[214] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis, "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks," *Journal of Computational Physics*, vol. 404, p. 109136, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999119308411

[215] "Welcome to the Ray documentation — Ray 2.0.0." [Online]. Available: https://docs.ray.io/en/latest/index.html#

[216] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Hardt, B. Recht, and A. Talwalkar, "A System for Massively Parallel Hyperparameter Tuning," *MLSys Conference*, 10 2020.