M 2024

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

SCIENTIFIC MACHINE LEARNING FOR PHENOMENA DRIVEN MODELLING

SOLID-STATE BATTERIES AS A CASE STUDY

GUSTAVO VICENTE DE CARVALHO

MASTER THESIS IN CHEMICAL ENGINEERING
PRESENTED TO THE FACULTY OF ENGINEERING
OF THE UNIVERSITY OF PORTO

Master in Chemical Engineering

SCIENTIFIC MACHINE LEARNING FOR PHENOMENA DRIVEN MODELLING

Master Thesis

of

Gustavo Vicente de Carvalho

Developed within the course of dissertation

held in

Norwegian University of Science and Technology - NTNU

Supervisor at FEUP: **Prof. Maria João Regufe**

Coordinator at NTNU: **Prof. Idelfonso Nogueira**

February 2024

Acknowledgements

I would like to express my deepest gratitude to Professor Idelfonso for allowing me to work under his guidance, suggesting my thesis topic, and enabling me to pursue my research at the esteemed NTNU institution. It was a fantastic experience, and I feel extremely lucky to have met him. I am also grateful to Maria João Regufe for being my supervisor at FEUP, I would not have been able to do my thesis if was not for her.

This journey would also not have been possible without Igor Iwakiri, who was my co-supervisor. He gave me incredible feedback, constructive criticism, and had weekly meetings with me even in very hectic times for him. Working with Igor is easily the best-case scenario that a master's student can have. His approach as a co-supervisor is the perfect equilibrium of letting the students learn by themselves and do the hard work, while still being there to guide when needed. Working with Igor has been a privilege and a blessing in my life, I am deeply appreciative of his expertise, guidance, and support.

I am also immensely grateful for the exceptional assistance provided by Carine Rebello, a researcher at the Chemical Engineering Department of NTNU. Carine's expertise in coding and her in-depth understanding of neural network concepts were invaluable throughout this thesis. Her guidance and support have been extremely helpful in the development of the technical aspects of this work.

While academic support is crucial, the emotional and moral support of loved ones is equally essential during the thesis process. I am deeply grateful to Miriam, my girlfriend, for her unmeasured support and encouragement throughout my journey. Her belief in me has been a constant source of strength. I am also grateful to my friends, Luiza and Vinicius, who offered their hospitality by providing me with a temporary home during the most intense phase of my thesis writing process. Their generosity and understanding were invaluable to my focused work.

Finally, I extend my heartfelt thanks to my parents and my older brother for their emotional and financial support throughout my master's studies and the thesis process. They have set a remarkable example of hard work and dedication that I aspire to emulate. Their sacrifices have made this achievement possible, and I am forever indebted to them.

To all those who have contributed to this journey, I am immensely grateful. Your support and encouragement have been essential in the completion of this thesis and have left an unerasable mark on my academic and personal growth.

Abstract

This master thesis explores the integration of Machine Learning (ML) techniques, particularly Neural Networks (NNs), in the field of Chemical Engineering, using solid-state batteries as a specific case study. The primary objective was to develop and evaluate NNs that can effectively predict data generated by a battery model developed by Iwakiri [1], thereby enhancing the modelling process in terms of flexibility and speed. A comparison was conducted between the NN models developed in this thesis and traditional modelling methods, specifically the Raichu model, developed by Iwakiri [1]. This comparison revealed a significant enhancement in terms of computational speed, where the NN models outperformed Raichu by several orders of magnitude, around 1475 times faster. This finding highlights the potential of NNs in revolutionizing battery modelling, particularly in scenarios requiring fast predictions, *e.g.* control or optimization routines. The research involved the development of three distinct NN models: the Key Performance Indicators (KPIs) Neural Network and two Potential Curve Neural Networks. These models were meticulously trained, tested, and optimized for high accuracy and efficiency. The KPIs Neural Network excelled in directly predicting the extraction and voltaic efficiencies of solid-state batteries, achieving mean absolute error (MAE) and mean squared error (MSE) in the order of 10^{-3} and 10^{-5} (dimensionless), respectively. The Potential Curve Neural Networks, developed for battery charging and discharging, showed great performance in accurately forecasting battery potential over time curves, with an MAE and MSE in the order of 10^{-2} V. These results underscore the reliability and precision of the developed models in predicting critical battery performance indicators. A comparative analysis to predict the extraction and voltaic efficiencies was done between the KPIs NN, which predicted the KPIs directly, and the Potential Curves NN, which KPIs were indirectly calculated through the predicted curves. It was found that training a NN specifically for predicting the Battery's KPIs will produce more accurate results than calculating the KPIs indirectly by the predicted Battery's Potential Curves.

Keywords: Scientific Machine Learning, Battery Modelling, Neural Networks, ML in Chemical Engineering

Resumo

Esta dissertação de mestrado explora a integração de técnicas de Aprendizado de Máquina (Machine Learning - ML), particularmente Redes Neurais (Neural Networks - NNs), no campo da Engenharia Química, usando baterias de estado sólido como um estudo de caso especifico. O objetivo principal foi desenvolver e avaliar modelos de Rede Neural que possam prever efetivamente os dados gerados por um modelo de bateria desenvolvido por Iwakiri [1], melhorando assim o processo de modelagem em termos de flexibilidade e velocidade. Foi realizada uma comparação entre os modelos de Rede Neural desenvolvidos nesta tese e os métodos de modelagem tradicionais, especificamente o modelo Raichu, desenvolvido por Iwakiri [1]. Esta comparação revelou um aumento significativo em termos de velocidade computacional, onde os modelos de Redes Neural superaram o Raichu por várias ordens de magnitude, cerca de 1475 vezes mais rápido. Este achado destaca o potencial das Redes Neurais em revolucionar a modelagem de baterias, particularmente em cenários que exigem previsões rápidas, *e.g.* rotinas de controle ou otimização. A pesquisa envolveu o desenvolvimento de três modelos distintos de Redes Neurais: a Rede Neural de Indicadores-Chave de Desempenho (Key Performance Indicators - KPIs) e duas Redes Neurais de Curvas de Potencial. Estes modelos foram meticulosamente treinados, testados e otimizados para alta precisão e eficiência. A Rede Neural de KPIs se destacou na previsão direta das eficiências de extração e eficiências voltaicas de baterias de estado sólido, alcançando um erro absoluto médio e um erro quadrático médio na ordem de 10^{-3} e 10^{-5} (adimensionais), respectivamente. As Redes Neurais de Curvas de Potencial, desenvolvidas para carregamento e descarregamento de baterias, mostraram grande desempenho em prever com precisão as curvas de potencial da bateria ao longo do tempo, com erro absoluto médio e erro quadrático médio na ordem de 10^{-2} V. Esses resultados ressaltam a precisão dos modelos desenvolvidos na previsão de indicadores críticos de desempenho da bateria. Uma análise comparativa para prever as eficiências voltaicas e de extração foi feita entre a Rede Neural de KPIs, que previa os KPIs diretamente, e a Rede Neural de Curvas de Potencial, cujos KPIs eram calculados indiretamente através das curvas previstas. Verificou-se que treinar uma rede neural especificamente para prever os KPIs da bateria produzirá resultados mais precisos do que calcular os KPIs indiretamente pelas curvas de potencial da bateria preditas.

Declaration

I hereby declare, under word of honour, that this work is original and that all non-original contributions is indicated and due reference is given to the author and source

Sign and date

Porto, 05/February/2024

Index

List of Figures

List of Tables

Notation and Glossary

List of Acronyms

ADAM	Adaptive moment estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
C-Rate	Charge or Discharge rate of the Battery
CNN	Convolutional Neural Network
FFNN	Feed Forward Neural Network
FNO	Fourier Neural Operator
GRU	Gated Recurrent Unit
KPI	Key Performance Indicator
LHS	Latin Hypercube Sampling
LSTM	Long Short-term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
NN	Neural Network
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation
PINN	Physics-Informed Neural Network
PINO	Physics-Informed Neural Operator
RNN	Recurrent Neural Network
SciML	Scientific Machine Learning
UDE	Universal Differential Equations

1. Introduction

This chapter presents the framing and the presentation of the work (Section 1.1), purpose and research motivation (Section 1.2), as well as the thesis structure (Section 1.3).

1.1. Framing and presentation of the work

Chemical Engineering is a data-rich subject. It heavily relies on the collection and analysis of data for tasks such as comprehending flow patterns, optimizing chemical reactions, and monitoring and controlling chemical processes and systems. However, despite its data-driven nature, the integration of Machine Learning (ML) into Chemical Engineering was relatively limited until recent years [2]. One of the contributing factors to this was the lack of computational power and infrastructure for data generation and storage. With the improvement of those areas in the last years, Scientific Machine Learning (SciML) is becoming a valuable tool in the toolbox of Chemical Engineers. This is even more valuable in the case of time-constraint applications or industries where speed will result in a competitive advantage, *e.g.*, planning and optimization in real-time. Such applications require high accuracy, and it is possible to deploy models that have learning capabilities which will detect out patterns and learn from data over time [2].

The increase in the interest in applying AI and ML in Chemical Engineering is visible in the evolution of the numbers of AI and ML publications within the Chemical Engineering domain shown in Figure 1. The Figure shows that following a substantial increase in publications there is a phase of disinterest, due to the limitations of early computing power and the complexities associated with developing effective ML algorithms [3]. Now, AI in chemical engineering came back again to be in a “hot” status and it is not clear if the curve will soon reach a condition in which it flattens out [2].

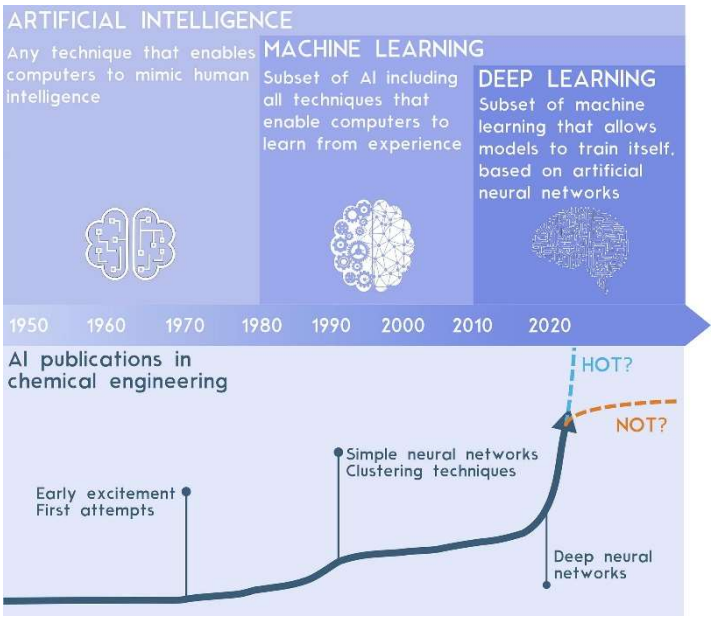

***Figure 1**: Evolution of the numbers of AI and ML publications within the Chemical Engineering domain [2].*

ML techniques are not new, they have been studied for several decades [4]. However, ML improved substantially in the past decade, not only because it is in more evidence in recent years but especially because of the development of better learning algorithms, availability of big data, easier access to better computational power, and its practical applications across real-world scenarios [5]. ML is showing remarkable success in various fields and enabling the extraction of crucial insights from data, whether is observational or simulated [6].

ML methods, especially with the rise of neural networks (NNs) and Deep Learning, are nowadays used widely in commercial applications [7], and have found profound utility in the engineering realm, revolutionizing how complex engineering problems are approached and solved. Engineers are increasingly leveraging ML techniques to model and optimize intricate systems, *e.g.*, designing more efficient aircraft structures [8], enhancing the performance of renewable energy systems, and predicting fuel properties [9]. In some domains, certain ML methodologies have proven to be not only more flexible and faster but also as accurate as traditional modelling techniques [10]. This success has led to a considerable uptake in the use of ML in many scientific areas, including Chemical Engineering.

Despite its great success, ML still has limits, especially when dealing with insufficient training data. To sum that, in some complicated physical systems in Chemical Engineering, collecting enough data can be a challenge sometimes due to issues such as cost or practicality. For example, obtaining enough data from a structure such as a riser can present substantial challenges in terms of both difficulty and expense [11]. To combat this problem, of finding

the solution of complex systems with limited data, several ideas and approaches were made, which, some of them, are going to be discussed later in this thesis.

1.2. Purpose and Research Motivation

This thesis explores how to use different methods from the realm of SciML in the field of Chemical Engineering, particularly emphasizing their application in phenomena-driven modelling and Partial Differential Equations (PDEs) regression. The primary objective was to develop and evaluate Neural Networks that can effectively predict data generated by a traditional phenomenological model, thereby, potentially enhancing the modelling process in terms of flexibility and speed. Replicating traditional models with a faster Neural Network can be particularly useful in scenarios requiring fast predictions, *e.g.* control or optimization routines. To demonstrate this concept a traditional model was required. Therefore, the Solid-state battery model developed by Iwakiri [1] was chosen, although any phenomenological model could have been employed. This thesis intends to use SciML techniques, such as Feed Forward Neural Networks (FFNNs) and Recurrent Neural Networks (RNN), to create surrogate models and to gain insights into their potential for enhancing the phenomena-driven modelling. This enhancement may manifest as faster execution and/or increased flexibility. Through this investigation, we aspire to contribute to the ongoing discussion surrounding the broader applicability of ML techniques in Chemical Engineering.

1.3. Organization of the dissertation

This dissertation structure is designed to present information in a way that facilitates the reader's comprehension of the study's context and methodology. The order of presentation is not necessarily the chronological order of what the experiments were executed but is crafted to optimize understanding of the research process and its findings.

The thesis starts with the “Context and State of Art” section. In it is shown the state of art in terms of Neural Networks used to regression, data prediction, and to learn and capture PDE’s behaviors, been this done through data or through the direct use of the PDE. Right after, the “Methods and Applications” section describes step by step the assembling and training of the specific Neural Networks used on this thesis, from the data production till the architecture selection. In the “Results and Discussions” section, the Training and evaluation of the Neural Networks are shown, as well as two interesting comparisons: One between the accuracy of the Neural Networks developed on this thesis, and the other between the computational speed of the Battery model proposed by Iwakiri [1] and the Neural Networks developed on this thesis. Finally, in the “Conclusion” the main thoughts, findings and conclusions that were exposed during the entire thesis are summarized.

2. Context and State of Art

In the past few years, Scientific Machine Learning (SciML), which is a field of research that combines the power of machine learning with the rigor of scientific modelling, has experienced rapid and significant growth. This has largely been driven by improvements in learning algorithms, which, combined with data-driven methods and the understanding of physical laws and mathematical principles, have pushed the field forward [6]. Several innovative ML methodologies have emerged and are transforming how complex physical systems are comprehended and modeled. These methods have proven particularly valuable in Chemical Engineering, which is a data-rich field [2]. One of these methods is the use of Artificial Neural Networks (ANNs) for PDEs regression and data prediction.

In the context of this thesis, as said priorly, the primary objective was to develop and evaluate Neural Networks that can effectively predict data generated by a traditional phenomenological model, effectively creating a surrogate model. To demonstrate this concept the Solid-state battery model developed by Iwakiri [1] was chosen. In this particular model it is possible to specify some characteristics of the battery, *e.g.*, electrolyte and cathode thickness, diffusion coefficient of Li+, the charge or discharge rate of the battery and more. Using PDEs, and initial and boundary conditions, that describe the Battery, the model can calculate and recreate the battery`s behavior through the application of numerical methods. The idea of this thesis is to recreate this battery model using ANNs. The goal is to increase the speed of the calculations without losing accuracy. The reason behind this objective is the fact that ANNs have already demonstrated great capabilities as a regression method for approximating solutions to different PDEs [12]. Researchers have developed several types of ANNs designed for specific scenarios and objectives [13]. In Table 1, a list of ANNs from various studies is presented, each uniquely designed to address specific challenges.

***Table 1**: A list of studies that applied ANNs within PDEs.*

NN Architecture	What it was used for	Reference
Conventional ANN	To accurately estimate the drag coefficient of a spherical particle as it moves through viscoelastic fluids.	[14]
Conventional ANN	Denoise data and approximate the governing PDEs of the systems. The methodology was on 3 PDE models for biological transport: advection-diffusion, classical Fisher-KPP, and nonlinear Fisher-KPP equations.	[15]
U-net, CNN	To predict the velocity and pressure fields around arbitrary 2D shapes	[16]
h-PINN	To leverage PINNs to solve topology optimization problems by imposing hard constraints using the penalty method and the augmented Lagrangian method.	[17]
Fourier Neural Operator (FNO)	To learn mappings between infinite-dimensional spaces of functions. They propose a neural operator architecture that directly approximates the operator of a PDE, allowing for faster and more efficient solutions compared to traditional solvers.	[18]
Parabolic and hyperbolic CNNs	Innovate on existing ResNet's architectures by introducing new classes of convolutional neural networks (CNNs) that are informed by PDE.	[19]

From this point, it is wise and facilitates comprehension to separate the ANNs used for the regression of PDEs by two parameters inherent to the specific problem: Availability of Data and the nature of the PDE being addressed, whether it is an individual PDE or a set/group of PDEs.

Multiple NNs were already applied for PDEs regression, which is the case of Convolutional Neural Networks (CNNs) [20], Mixture density networks (MDNs) [21], Bayesian Neural Networks (BNNs) [22], and similar models. Typically, the training and optimization of these NNs necessitate a substantial volume of data. However, there exist scenarios where data collection/availability is a challenge. Nevertheless, even in these instances, one still understands the underlying physical principles/phenomena and laws governing the system under study. Therefore, different strategies were created to incorporate these fundamental

laws into the training of the ANNs. A great example of this approach is the concept of Physics-informed Neural Networks (PINNs) [23, 24] and the Physics-informed Neural Operator (PINO) [25]. The specific mechanics behind how this integration is accomplished will be covered in later sections of this thesis.

One notable limitation of training a NN to solve a specific PDE is the computational intensity of the training phase. Moreover, after completing the training process, the resultant model is exclusively for the resolution of this particular PDE. If one wishes to modify any aspect, such as constants or initial conditions, it necessitates to retrain the NN [23]. This thought gave rise to the concept of Neural Operators. Unlike conventional NNs that map within finite-dimensional Euclidean spaces or sets, Neural Operators learn to map between infinite dimensional functional spaces [18]. Therefore, they possess the capacity to approximate to a complex nonlinear operator. Further explanation of the mechanics of Neural Operators, along with illustrative examples, will be presented in subsequent sections of this thesis.

This section has introduced the diverse range of NNs that can be employed in phenomena-driven modeling and PDEs regression. These networks vary in their applicability and underlying principles. The subsequent sections (2.1 – 2.4) delve into the details of some of the most prominent and utilized Neural Networks, highlighting their theoretical fundamentals, advantages, limitations, and presenting a comprehensive list of relevant research articles that have successfully employed them.

2.1. Feed Forward Neural Networks (FFNNs)

Feed Forward Neural Networks (FFNNs) are the classic network used in ML. They are widely used due to their simplicity and effectiveness in handling complex nonlinear relationships, they are considered one of the cornerstones in the ML field. A FFNN consists of numerous layers of interconnected artificial neurons, organized in a sequential way [26]. The data moves exclusively in a single direction, starting from the input layer, passing through the hidden layers, and ultimately reaching the output layer. Each neuron calculates a weighted sum of its inputs, applies an activation function, and transfers the output to the subsequent layer. The activation function introduces nonlinearity, enabling the network to capture intricate relationships, otherwise, the network would only be a fancy way of doing a linear combination.

In the training phase, the FFNN undergoes a learning process to fine-tune the weights and biases associated with each neuron, minimizing the disparity between its predicted outputs and the actual target values. This refinement is achieved through backpropagation, where the error is retroactively propagated through the network, and the weights are adjusted

accordingly [26]. Through iterative training, the network progressively enhances its ability to approximate to the desired function. A diagram of a simple FFNN is shown in Figure 2.

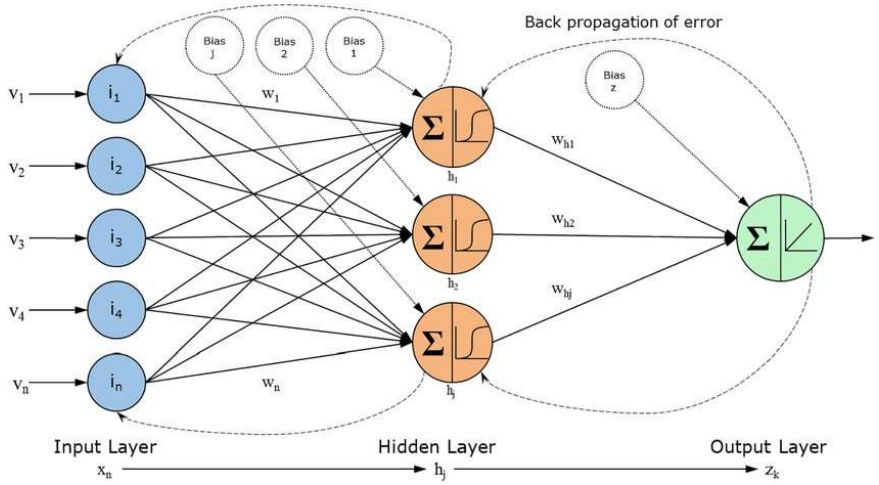

***Figure 2**: Workflow diagram of an artificial neural network architecture and algorithm [27].*

By training a neural network on a set of input-output pairs generated from a PDE, the network can learn to approximate to the underlying function [28]. The input data typically consists of variable values and possibly additional parameters, while the output data represents the solution of the PDE at those points.

2.2. Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of neural network architectures that are excellent at modelling sequential data by capturing temporal dependencies. Unlike FFNNs, RNNs have an internal memory, also called hidden state, that allows them to process and store information about past inputs. This memory enables RNNs to make predictions based not only on current inputs but also on the context of previous inputs [29]. As a result, RNNs have found widespread applications in solving problems related to time series analysis. During training, the RNN learns to adjust the weights associated with its recurrent connections and the input connections with backpropagation as well.

In the context of time series regression, RNNs are particularly well-suited for tasks such as forecasting, anomaly detection, and sequence generation [30]. By considering the temporal order of the data, RNNs can capture patterns, trends, and dependencies that exist within the time series. This makes them powerful tools for analyzing and predicting complex dynamical systems. In both PDE and time series regression, RNNs offer advantages over traditional numerical methods or FFNNs [30]. RNNs can handle variable-length inputs and outputs, adapt to changing or irregular time steps, and capture long-term dependencies in the data. A diagram of a simple RNN is presented in Figure 3.

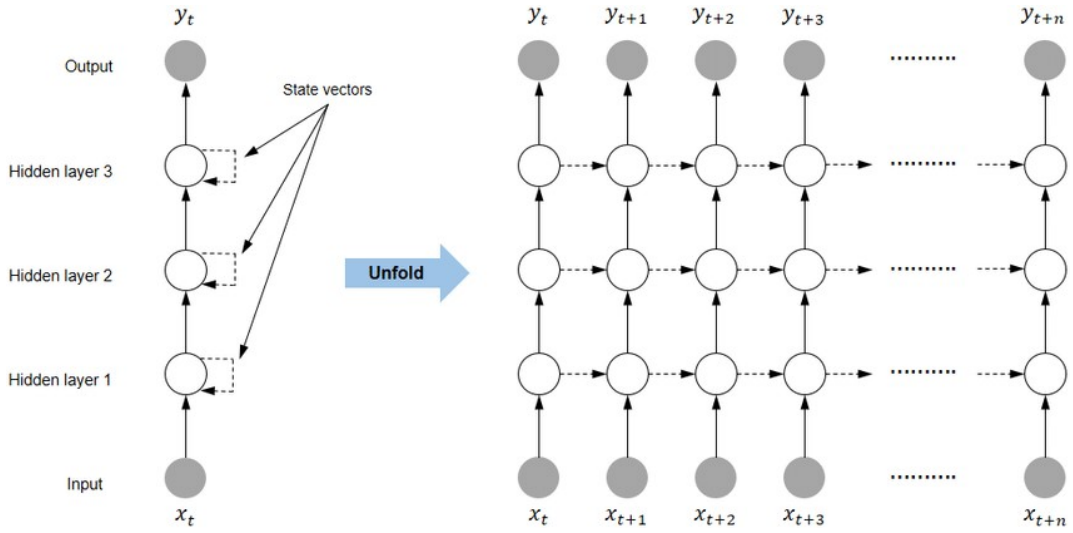

***Figure 3**: Diagram of a deep Recurrent Neural Network (RNN) and its unfolding in time [31].*

The most used RNNs are the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) [32], empirical studies have shown that both GRU and LSTM networks can achieve comparable performance on a wide range of tasks [33]. As said priorly, these types of RNNs are usually used for forecasting time-series, some examples of where they have been applied are the forecasting of the electric load in power grids [34], the prediction of dissolved oxygen in a fishery pond [35], and the forecasting of wastewater treatment plant key features, *i.e.* influent flow, influent temperature, influent biochemical oxygen demand (BOD), effluent chloride, effluent BOD, and power consumption [36].

2.3. Neural Operators

The concept of Neural Operators marks a significant shift in the field of ML, moving from conventional mappings between finite-dimensional spaces to the less explored realm of function spaces. When applied to PDEs, Neural Operators enable the direct learning of mappings from parameter dependencies to solution spaces [18]. Therefore, instead solving a specific PDE it can learn multiple PDEs at once. Several Neural Operators were developed throughout the years. The most prominent and utilized will be show in the subsequent sections (2.3.1 – 2.3.3).

2.3.1. Deep Operator Networks (DeepONets)

DeepONets have become a powerful tool in the world of SciML, utilizing NNs to accurately model complex, nonlinear continuous operators. What made DeepONets especially fascinating is the idea that an ANN with enough hidden layers can achieve remarkable precision in approximating any operator [37]. This idea is rooted in the universal approximation theorem [38]. The DeepONet was the first Neural Operator concept, proposed in 2019 [39]. Based on

the DeepONet concept, many others Neural Operators were created. One of the most influent is the Fourier Neural Operator (FNO), which will be covered in the subsequent section of this thesis.

DeepONets are not just theoretical concepts. They have already been applied and offer practical solutions for precise and efficient operator learning, even when working with relatively small data sets [39]. The DeepONets architecture is divided into two distinct parts: one for encoding input functions and the other for encoding output function locations. This structured approach has been thoroughly tested through simulations, and the results show a significant reduction in generalization errors compared to traditional fully connected networks [40].

To exemplify, an illustration of the architecture of a DeepONet, the shape of the data and the network setup is presented in Figure 4. Figure **A)** shows how the DeepONet seems to work from outside, a NN that receives two inputs $[\boldsymbol{u}(\boldsymbol{x_3}), \boldsymbol{u}(\boldsymbol{x_2}), \ldots, \boldsymbol{u}(\boldsymbol{x_m})]$ and $\boldsymbol{y}$, and then learn an operator $\boldsymbol{G}: \boldsymbol{u} \rightarrow \boldsymbol{G}(\boldsymbol{u})$. Figure **B)** presents the shape of the training data. For each input function $\boldsymbol{u}$ it is necessary to have the same number of "sensors", points of the $\boldsymbol{x}$ **domain** in which the function $\boldsymbol{u}$ was evaluated. However, it is not necessary to impose any constraints on the number of locations for the evaluation of the output functions. Figure **C)** illustrates an example of a "stacked" DeepONet, which main characteristic is to have one branch network for each input function $\boldsymbol{u}$. The final figure, Figure **D),** illustrates an example of an unstacked DeepONet, which is the most conventional. It has only one trunk network and one branch network.

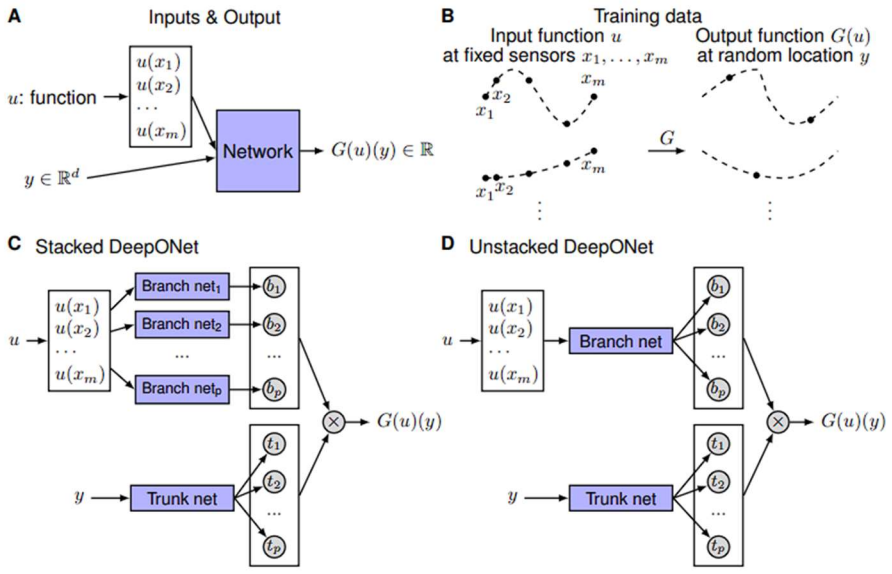

***Figure 4**: Illustrations of the architecture of a DeepONet, the shape of the data and the network setup [37].*

Through experimental validations, it has become clear that this approach is great in addressing complex challenges, including the modelling of turbulent flows [41]. Systematic simulations showed that DeepONets have capability to accurately and efficiently identify and approximate dynamic systems and PDEs. Furthermore, DeepONets showcase impressive error convergence rates, ranging from half order to fourth order, and can even achieve exponential convergence as the training data set size increases [41]. This extraordinary potential has profound implications for modelling complex systems in various domains. In Table 2 it is listed studies that applied the DeepONet, or a variation of it, to model different scientific problems.

***Table 2**: A list of studies that applied the DeepONet.*

DeepONet type	What it was used for	Reference
Conventional DeepOnet	To learn non-linear operators accurately and efficiently with low generalization error. It was tested the generalization error for 16 different diverse applications.	[42]
DeepM&Mnet	To predict accurately and efficiently 2D electroconvection fields from unseen electric potentials.	[43]
V-DeepONet	To propose a physics-informed variation of DeepONet to analyze brittle fracture.	[44]
Conventional DeepOnet	To predict the linear amplification of instability waves in high-speed boundary layers and to perform data assimilation.	[45]
Convolutional Autoencoder DeepOnet	To learn the dynamic development of a two-phase mixture and reducing solution time for predicting microstructure evolution.	[46]

2.3.2. Fourier Neural Operator (FNO)

The Fourier Neural Operator (FNO) is a great development within the field of SciML. Traditionally, neural networks have primarily focused on mapping between finite-dimensional Euclidean spaces. However, the FNO, as the DeepOnets, extends the capabilities of neural networks by enabling them to learn mappings between function spaces, particularly in the context of PDEs [18]. Unlike classical methods that solve a specific instance of a PDE, neural operators like the FNO have the ability to directly learn the mapping from any parametric dependence to the solution, essentially learning an entire family of PDEs.

One of the key innovations of the FNO compared to the DeepONet and others Neural Operators is its parameterization of the integral kernel directly in Fourier space [10]. This approach allows for an expressive and efficient architecture, making it particularly well-suited for solving complex physical systems. These representations are excellent at capturing the oscillatory patterns often inherent in physical systems. The model`s expressiveness enables it to faithfully represent intricate and complex phenomena, making it a valuable asset for understanding and simulating a wide range of natural phenomena [47].

The FNO's parameterization involves working with the Fourier transform of functions, which provides an efficient means of representation and computation. It can handle discretized domains and consistently delivers reliable performance across different input and output resolutions. One of its notable features is its invariance to discretization, this means that the Fourier Neural Operator can learn from and evaluate functions that have been discretized in various ways. This property enables zero-shot super-resolution and ensures consistent error levels across different resolutions [10].

To exemplify, an illustration of a FNO and of a Fourier Layer is shown in Figure 5. The figure **a)** presents the full architecture of the FNO. First, the input $\boldsymbol{a}(\boldsymbol{x})$ is transmitted to a Dense layer **P**, from there the data goes through **T** Fourier layers. The final Dense layer **Q** transforms the data back to the target dimension, and its output is $\boldsymbol{u}(\boldsymbol{x})$. Figure **b)** illustrates what the Fourier layer actually looks like. Inside the Fourier layer the input goes through two processes. In the top one, the Fourier transform is applied to the input, then a linear transform **R** is applied on the lower Fourier modes, which also filters out the higher modes, and finally the inverse Fourier transform is applied to make the input return back to the original dimension. In the bottom one, the input just goes through a linear transformation.

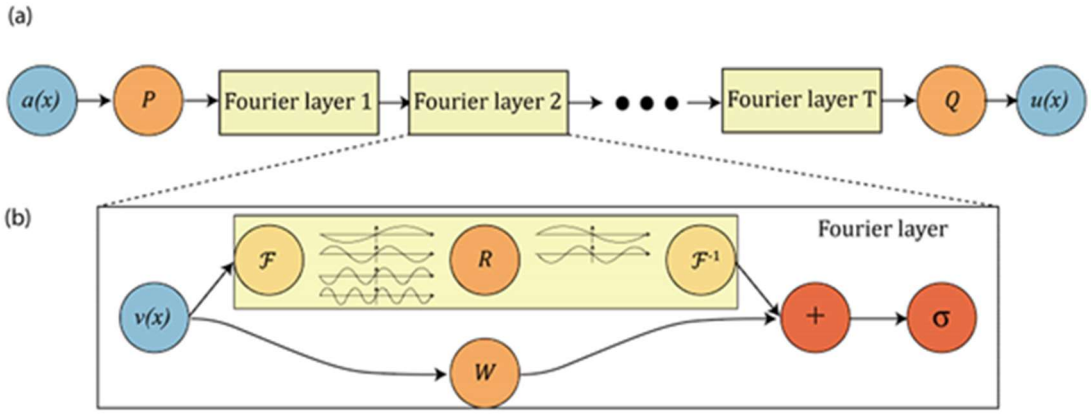

***Figure 5**: Illustration of a FNO and of a Fourier Layer [10].*

What sets the FNO apart is its exceptional computational efficiency. It can perform computations with remarkable speed, often outpacing traditional numerical solvers for PDEs

by up to three orders of magnitude [10]. This efficiency is particularly valuable in scenarios where rapid solutions are imperative, such as real-time simulations and optimization tasks. Despite its remarkable speed, the FNO maintains an impressive level of accuracy [48]. This precision is crucial, especially in scientific and engineering applications where even minor deviations from reality can have significant consequences. Moreover, the versatility of the FNO is a noteworthy aspect. It has demonstrated its effectiveness across multiple domains and applications, such as fluid dynamics [49], enhancing image super-resolution [50] and signal processing [51]. In Table 3 are listed studies that applied the FNO, or a variation of it, to model different scientific problems.

***Table 3**: A list of studies that applied the FNO.*

FNO type	What it was used for	Reference
Conventional FNO	To solve and generate simulations of the 2D photoacoustic wave equation in a homogeneous medium.	[52]
IF-NO	To predict how a material behaves by mapping the applied loads with the resulting changes in displacement or damage within the material. Hyper elastic, anisotropic and brittle materials were used as test examples.	[53]
FourCastNet	To create a weather forecasting model that offers precise short to medium-range predictions on a global scale. Effectively predicts high-resolution, rapidly changing variables such as surface wind speed, precipitation, and atmospheric water vapor.	[54]
Conventional FNO	To predict the velocity spectrum, vorticity and velocity increment probability density functions (PDFs), and instantaneous flow structures based on filtered velocity fields.	[55]
U-FNO	To solve highly complex CO_2-water multiphase problems with wide ranges of permeability and porosity heterogeneity, anisotropy, reservoir conditions, injection configurations, flow rates, and multiphase flow properties.	[49]

2.3.3. Physics-Informed Neural Operators (PINO)

Physics-Informed Neural Operator (PINO) represents a hybrid approach that combines training data and physics constraints to learn the solution operator of a given family of PDEs [25]. PINO was another variation of DeepONets, which innovation was to use the PDEs as constraints to guide the learning and to optimize hyperparameters of the Neural

Operator, ensuring that it satisfies the given physical equations at specific points in the domain (boundary conditions) and within the domain (initial conditions). Through this approach it is possible to combine lower-resolution training data with higher-resolution PDE constraints, which allows to accurately approximate solution operators for a wide range of PDE families even in the cases where acquiring data is a challenge [56]. Therefore, PINO is useful when it comes to zero-shot super-resolution, where it can predict outcomes even when there's a low amount of relevant training data available.

PINO builds on the strong foundation of the FNO and DeepONet frameworks, providing highly accurate reconstructions of true operators [25]. It is extremely useful in situations where training data is scarce or entirely absent [56], demonstrating its reliability and adaptability. In contrast to previous methods like PINNs, which struggled with optimization challenges in complex multi-scale dynamic systems [57, 58], PINO emerges as a solution, opening new possibilities for tackling intricate modelling tasks.

To exemplify, an illustration of architecture and setup of a PINO is shown in Figure 6. First, the input a is "lifted", increased in dimension, and the output v_0 goes through a linear integral operator and then to a non-linear function. The output of the non-linear function and the query points x are inputted in a chain of **L** linear integral operators. Finally, the output V_L is projected by a pointwise projection operator to output the function u, from it is possible to calculate the data loss. Due to the fact that both u and V_L are functions, it is possible to calculate the derivative of u using the chain rule.

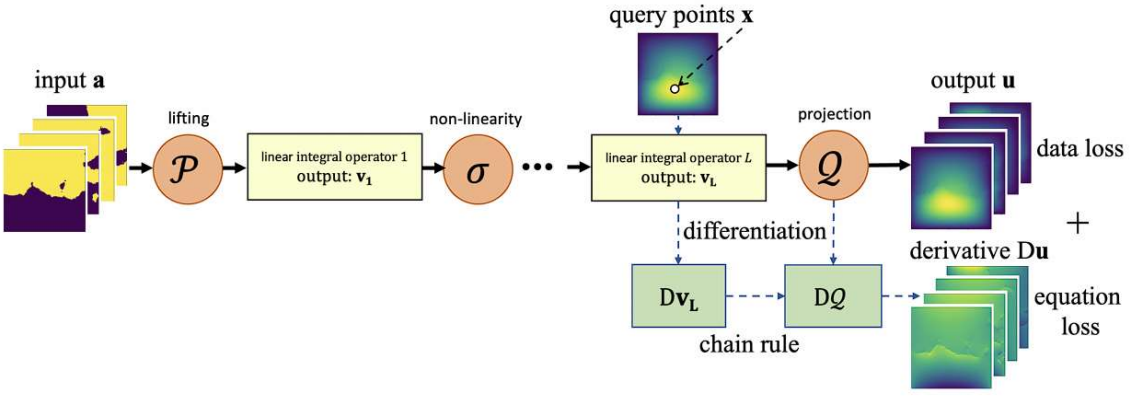

***Figure 6**:Illustration of architecture and setup of a PINO [25].*

While the idea of using PINO is appealing, particularly for its potential to work with less data, it still has its limitations and drawbacks. Similar to other Physics-Informed approaches, training PINOs can be computationally demanding, especially when dealing with high-dimensional

problems [56]. This increased computational load is primarily due to the need for numerical integration to enforce the constraints of the PDEs.

Several studies used and proved the capabilities of PINOs. One highlighted their usefulness in diverse applications in computational mechanics, including porous media, fluid mechanics, and solid mechanics [59]. Another employed PINOs in the context of high-dimensional Bayesian inverse problems, which traditionally necessitate an enormous, and frequently infeasible, number of forward models [60]. In the latter the PINO ability of zero-shot super-resolution allowed them to construct accurate approximations of the full posterior, irrespective of the number of observations points and the magnitude of the observation noise [60]. They demonstrate the efficacy and accuracy of the PINO methodology in the context of inverse problems for three benchmarks: an anti-derivative equation, reaction-diffusion dynamics, and flow through porous media. In Table 4 it is listed studies that applied the PINO, or a variation of it, to model different scientific problems.

***Table 4**: A list of studies that applied the PINO.*

PINO type	What it was used for	Reference
Conventional PINO	To solve multiple types of PDEs, including the 1D wave equation, 1D Burgers equation, 2D Burgers equation in the scalar, inviscid, and vector types, and the 2D linear and nonlinear shallow water equations.	[56]
FC-PINO	To employ the Fourier continuation (FC) to enable the exact gradient method for PINO in nonperiodic scenarios, through this method is possible to accurately predict the third-order derivatives of non-smooth solution functions.	[61]
Conventional PINO	To solve parametrized boundary value problems without labeled data. The network is showcased using 2D Laplace equation, 2D Bi-harmonic equation and 3D Helmholtz equation as examples.	[62]
Conventional PINO	To prove the efficacy of PINOs by solving the 2D Navier-Stokes equation for Long temporal transient flow, Chaotic Kolmogorov flow and Lid cavity flow.	[63]
Conventional PINO	To model 2D incompressible magnetohydrodynamics simulations.	[64]

2.4. Universal Differential Equations (UDE)

Universal Differential Equation (UDE) is a powerful mathematical framework for modelling physical simulations without the need for explicit full ODE/PDE definitions. UDEs aim to merge the mechanistic models, which rely on prior structural knowledge and are often used in scientific domains due to limited training data, and data-driven machine learning models, which are more flexible but require large amounts of training data [65].

In scientific scenarios, there are instances where only partial knowledge about certain factors that play a role in an equation is known. However, there are other factors that remain unknown, and a clear methodology to define them is missing. Rather than discarding the incomplete information or resorting to guesswork, UDEs utilize ANNs to take charge of these unknown elements. ANNs fill these gaps and provide a more comprehensive understanding of the problem, effectively completing the missing components and shedding light on the hidden aspects of the underlying physics of this equation. Therefore, the ANN acts like a parameter in the function.

To exemplify, an illustration of how an UDE is applied is shown in Figure 7. In the top of the schematic it is possible to see two fictional equations, they are going to be the target equations. Imagine that a specific physical phenomenon happened, and it is known that somehow these two equations describe it. Unfortunately, only a part of the equation is known. The NNs will be used then to "complete" this equation and replace the unknown part. The last thing to be done is to train and fit these NNs with the data collected from the physical phenomenon.

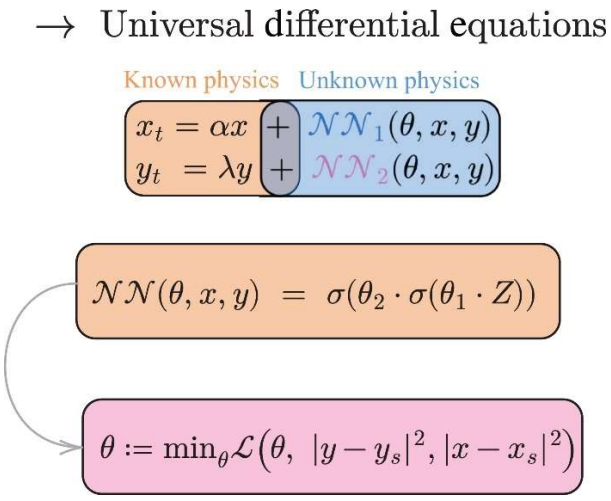

***Figure 7**: Illustration of universal differential equations (UDE) building and fitting procedure [66].*

In summary, UDEs are differential equations defined, in full or in part, by a universal approximator [38]. A universal approximator is a parameterized object capable of representing any possible function within certain limits. Examples of universal approximators include Fourier or Chebyshev expansions in low dimensions and neural networks in high dimensions [67]. UDEs can be formulated as forced stochastic delay PDEs with embedded universal approximators. The working principle of UDEs involves training the parameters of the universal approximator to minimize a cost function [68], a similar approach of what is used in the PINO [63]. This optimization process typically carried out using techniques like stochastic gradient descent, Adam [69], or L-BFGS [70], aims to find the best parameters that fit the given data and satisfy the differential equation. By leveraging the mathematical properties of UDEs and the tools provided by the SciML software ecosystem [71], such as efficient solvers and adjoint techniques, UDEs enable efficient handling of various applications, including biological mechanism discovery [72], solving high-dimensional equations like the Hamilton-Jacobi-Bellman equations [73] and solve multicomponent competitive adsorption systems [66]. In Table 5 it is listed studies that applied the UDE, or a variation of it, to model different scientific problems.

***Table 5**: A list of studies that applied the UDE.*

UDE type	What it was used for	Reference
Conventional UDE	Created a computationally efficient hybrid model to predict sorption uptake kinetics in complex non-linear advection-diffusion-sorption systems.	[74]
UODE	To solve the multicomponent separation by adsorption in a fixed bed column.	[66]
Multiple UDE types	To offer a showcase of how the UDE framework can handle a wide range of problems, *e.g.*, the Linear Quadratic Gaussian (LQS) problem, the 1D Fisher-KPP (Kolmogorov–Petrovsky–Piskunov) PDE and more.	[65]
SymODEN	To introduce a deep learning framework that infers the dynamics of physical systems described by ODEs from observed state trajectories.	[73]

3. Methods and Applications

The solid-state battery model proposed by Iwakiri [1] served as a foundation for this thesis, which aimed to replicate certain features of it. This chapter addresses the different steps taken to build the Neural Networks that will be used to replicate what these features do as similarly as possible. To enhance reading clarity and minimize redundancy, the solid-state battery model proposed by Iwakiri [1] will be referred to as “Raichu”.

The initial idea was to create two NNs, the First to predict the Key Performance Indicators (KPIs) of the battery, Extraction and Voltaic efficiencies, and the Second to predict the battery potential through time curve. Therefore, the first NN predicts 2 float numbers, while the second predicts a time-series. The two NN models followed a similar process during their development. To avoid redundancy, a summary of the general steps is presented here:

1. **Input data sampling:** To select the input variables data sets (also called features or predictors) from each variable ranges the Latin Hypercube Sampling (LHS) method was implemented [75], the same selected data sets were used in both NN’s trainings.
2. **Data Generation:** The Raichu was used to generate the required output data (also called the target or label).
3. **Data Treatment:** The features and targets where subjected to preprocessing and standardization to ensure its suitability for each of the NN trainings.
4. **NN Architecture Selection:** Various NN’s configuration were explored to identify the optimal architecture for the tasks. This involved considering factors like network depth, layer types, and activation functions. The hyperband search algorithm was implemented in this step [76].
5. **NN Training:** The selected NN architecture was trained on the prepared data set using an appropriate optimization algorithm [69]. This process involves iteratively adjusting the network's weights and biases to minimize the loss, mean squared error (MSE) or mean absolute error (MAE), between predicted and actual outputs. The TensorFlow python package [77] was used to perform the NN’s training.
6. **NN Model Evaluation and Export:** After training, the NN's performance was evaluated on a separate validation data set to assess its generalization ability. If satisfactory, the trained model was exported and saved for future use.
7. **Data Handling and Plotting:** The processed data was post-processed and visualized to gain insights into the relationships between input and output variables. This involved creating charts, graphs, and other visual representations.

This summary contains the essential steps involved in developing and evaluating both NN models, highlighting the fundamental processes without excessive detail. In the subsequent sections (3.1 – 3.4) the details will be presented.

3.1. Data Collection and Preparation

Data collection and preparation are the fundamental initial steps in any of the state-of-the-art ML processes presented previously. The data preparation significantly impacts the performance of NN models [78]. Therefore, this section provides a detailed explanation of the data preparation phase, outlining the reasoning behind each key decision.

3.1.1. Phenomenological Model

Delving into the intricate details of Iwakiri's [1] battery model, its operational principles and explaining its electrochemical aspect is not in the scope of this Thesis. Therefore, the differential equations expressing its physicochemical phenomena will not be approached. Nevertheless, a brief explanation of which Raichu's input and output variables were used and their meaning is crucial to understand its role in the NN's training process.

As previously stated, the primary purpose of this thesis was to replicate some features of the solid-state battery model developed by Iwakiri [1]. The Raichu produces multiple outputs, including extraction efficiency, voltaic efficiency, battery potential profiles, the evolution of the ion concentration profile over time, and further data elements. Given time constraints, the initial idea was to implement two NNs: One to predict battery KPIs, extraction and voltaic efficiencies, and the other to predict the battery potential over time curve. From this point, the first NN will be addressed as the KPIs Neural Network, and the second as the Potential Curve Neural Network.

The Key Performance Indicators (*i.e.* extraction efficiency and voltaic efficiency), as the name state, will represent the overall performance of the solid-state battery. The extraction efficiency refers to the battery ability to effectively utilize the available chemical energy in it, while the voltaic efficiency indicates how effectively the battery can be charged and discharged without losing energy to side reactions or other inefficiencies. Both KPIs range from 0% to 100%, indicating the degree of efficiency. The Battery Potential profile is a time-series that can be graphically represented, it shows how the voltage of a battery behave during charging and discharging cycles.

Developing two single-task NNs instead of one multitask NN was chosen due to the fundamentally different data types involved. Battery KPIs are float numbers, while battery potential curves represent time-series data, therefore both require distinct neuron types for optimal prediction. Furthermore, no performance advantage was observed with multitask NNs compared to single-task NNs [79], and their benefits tend to diminish with larger training datasets [80].

Now, already knowing which elements of data are going to be predicted (*i.e.* KPIs and the Battery Potential profile), it is also important to know which variables they rely on to be calculated, *i.e.* which are the input variables. The Raichu accepts multiple different inputs to generate data, however, only the most relevant ones were chosen for this thesis while the others were kept constant. The Raichu's input variables whose values varied in this thesis were 14:

1. Electrolyte thickness, which is a layer of LiPON (Lithium phosphorus oxynitride).
2. Cathode thickness, which is a layer of $LiCoO_2$ (Lithium cobalt oxide).
3. Diffusion scaling constants of the Li^+.
4. Diffusion scaling constants of the electrons in the LCO electrode.
5. Maximum concentration of $LiCoO_2$.
6. Diffusion coefficient of Li^+.
7. Diffusion coefficient of the uncompensated negative charge in LiPON electrolyte.
8. Maximum concentration of Lithium host sites in electrolyte.
9. Fraction of Li in LiPON that resides in the mobile state.
10. Lithium ion recombination reaction rate constant.
11. Standard reaction rate constant for the decomposition of $LiCoO_2$.
12. Charge-transfer coefficient for the positive electrode.
13. Standard reaction rate constant for decomposition of Lithium in Li^+ and e^-.
14. The charge or discharge rate of the battery. A positive C-rate means that the battery is charging, while the negative means it is discharging.

The abbreviation shown in Table 6 will be used to avoid long names, to facilitate comprehension, and because it was the one used in the code. Additionally, each of these variables had their specific range of values that they varied, the ranges are presented in Table 6 as well. The variables are presented in the same order as the previous list.

Table 6: Input variables abbreviation, values range, and dimension.

Variable Number	Abbreviation	Value Range	Dimension
1	thickness_electrolyte	Min. value = 10^{-6}, Max. value = 10^{-5}	m
2	thickness_cathode	Min. value = 10^{-6}, Max. value = 10^{-5}	m
3	DAB_cathode	Min. value = 10^{-13}, Max. value = 10^{-11}	$m^2 \cdot s^{-1}$
4	DB_cathode	Min. value = 10^{-13}, Max. value = 10^{-11}	$m^2 \cdot s^{-1}$
5	cmax_cathode	Min. value = 10^{2}, Max. value = 10^{5}	$mol \cdot m^{-3}$
6	DA_electrolyte	Min. value = 10^{-16}, Max. value = 10^{-14}	$m^2 \cdot s^{-1}$
7	DB_electrolyte	Min. value = 10^{-18}, Max. value = 10^{-14}	$m^2 \cdot s^{-1}$
8	c0_electrolyte	Min. value = $5 \cdot 10^{4}$, Max. value = 10^{6}	$mol \cdot m^{-3}$
9	Frac_electrolyte	Min. value = 0.4, Max. value = 0.99	Dimensionless
10	ka_electrolyte	Min. value = 10^{-10}, Max. value = 10^{-6}	$m^3 \cdot mol^{-1} \cdot s^{-1}$
11	k_positive	Min. value = 10^{-13}, Max. value = 10^{-9}	$m^{.5} \cdot mol^{-0.5} \cdot s^{-1}$
12	alfa_positive	Min. value = 0.4, Max. value = 0.6	Dimensionless
13	k_negative	Min. value = 10^{-11}, Max. value = 10^{-7}	$m \cdot s^{-1}$
14	C-rate	Min. value = -15, Max. value = 25	h^{-1}

To effectively sample points from the specified ranges the Latin Hypercube Sampling (LHS) method was implemented [75]. Utilizing the ranges from Table 6, 1300 sample points were generated and subsequently introduced in the Raichu model to gather the output data from it.

3.2. Data Treatment

The Raichu outputs 3 files for each sample point. Each file type was placed in a specific list:

- One file contains the information, characteristics, and conditions of the battery. These files were placed in a list arbitrarily named "*variable_files*".
- Another file contains all the data that the Raichu calculated, *e.g.*, battery potential over time curve, the evolution of the ion concentration profile over time, and further data elements. These files were placed in a list arbitrarily named "*dynamic_files*".
- The last file contains the 2 KPIs: extraction efficiency and voltaic efficiency. These files were placed in a list arbitrarily named "*efficiency_files*".

The code for the files separation is presented in Figure 8.

```
all_files = []
for file in os.listdir(os.getcwd() + "\\todos_dados"):
    all_files.append(file)

# Separate the files into variable_files, dynamic_files and efficiency_files
variable_files = [file for file in all_files if file.endswith("_info.csv")]
dynamic_files = [file for file in all_files if file.endswith("_data.csv")]
efficiency_files = [file for file in all_files if file.endswith("_KPI.csv")]
```

***Figure 8**: The code used to import and separate the files by type.*

The approach taken for each of the NNs deviates after this step. Therefore, a dedicated section describing the specific procedures taken for each of them is needed.

3.2.1. Data Treatment – KPIs Neural Network

The 14 input variables values were collected from the *variable_files* and then placed in a list for each variable, therefore, 14 lists were created. The last thing was to place all the 14 lists in a list called *x_data*. The *x_data* creation is presented in Figure 9.

```
x_data = [thickness_electrolyte_list, thickness_cathode_list, DAB_cathode_list,
        DB_cathode_list, cmax_cathode_list, DA_electrolyte_list, DB_electrolyte_list,
        c0_electrolyte_list, Frac_electrolyte_list, ka_electrolyte_list,
        k_positive_list, alfa_positive_list, k_negative_list, C_rate_list]
```

***Figure 9**: The code used to create the nested list "x_data".*

For the NNs training, it is necessary to have training, validation, and test data sets. The common practice in ML is to separate your data into 70% training data, 15% validation data, and 15% test data. Following the separation, all data sets were normalized using the training data set. This practice is used because it is important to ensure the integrity of the training data and prevent information leakage from the test data to the training data. After separating and normalizing the data, the training (*X_train*), test (*X_test*), and validation (*X_valid*) data sets are ready for use.

For the KPIs NN, the extraction efficiencies and voltaic efficiencies were collected from the *efficiency_files* and placed in a list for each variable. The 2 lists are then placed in the *y_data* list. The *y_data* list creation is presented in Figure 10. Similar to *x_data*, the *y_data* is separated into training, validation, and test data sets in the 70%, 15%, 15% proportion.

```
# y_data creation
y_data = [Extraction_Efficiency_list, Voltaic_Efficiency_list]
```

***Figure 10**: The code used to create the nested list "y_data".*

3.2.2. Data Treatment – Potential Curve Neural Network

For the Potential Curve NN, while it shares some similarities for the *x_data* creation with the KPIs NN, *e.g.*, the normalization of the 14 variables, it still involves a slightly more intricate approach. Instead of the NN input being a 2D matrix (14 columns and 1300 · 75% rows), it will be a 3D matrix (15 columns, 201 rows and 1300 · 75% layers), and, instead of 14 input variables, the Potential Curve NN uses 15 input variables: the 14 original input variables plus the time step that the potential was measured.

The Raichu's battery potential curve outputs don't have the same maximum time from sample to sample. Consequently, even though all battery potential curves were discretized with 201 points in time, the time steps are of different length from sample to sample. Fortunately, one of the battery's pieces of information registered in the *variable_files* is the "Time_max / s", which means the maximum time registered in the Battery potential curve. Therefore, a possible way to solve this issue was to find the longest maximum time of all the training samples and normalize all the other maximum times using it.

Understanding the steps to the *x_data* creation is facilitated by returning to the section 2.2 from the state of art. This particular NN will predict a time series (the battery potential through time curve), consequently a Recurrent Neural Network (RNN) is preferable to be used. Therefore, visualizing how the input of a RNN looks is essential. A diagram of one layer of the 3D matrix that is inputted in the RNN is shown in Figure 11.

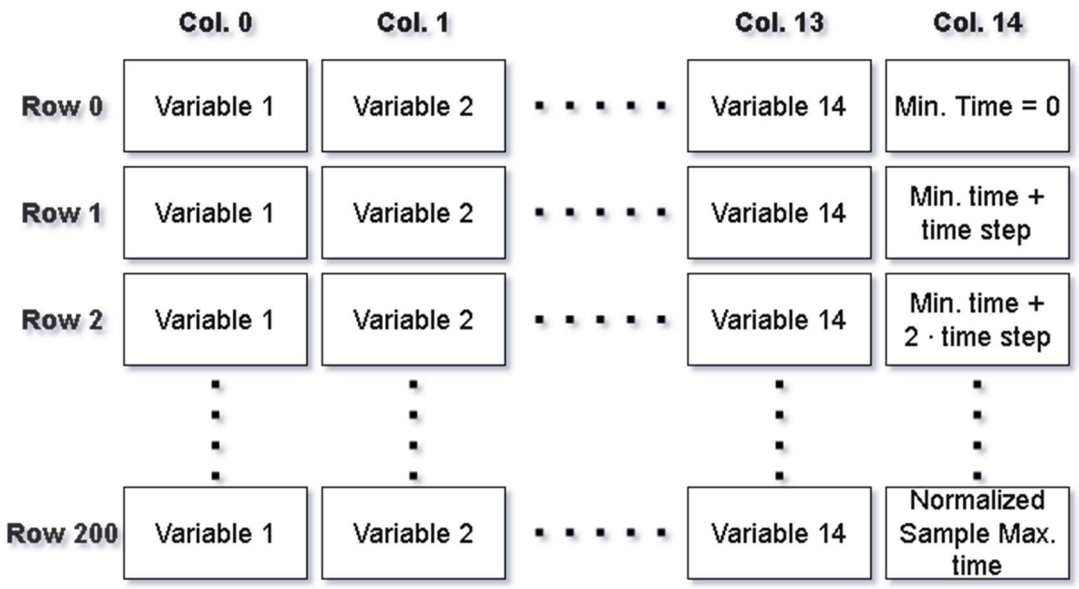

***Figure 11**: Diagram of a layer of a 3D input matrix for a RNN.*

The input of a RNN is a 3D matrix, and each layer of this matrix is one sample. The first 14 columns of this layer are the original 14 variables, and the last column is the time that the potential value was measured. There are 201 rows (the number of points in time measured by the Raichu). The first line of the first 14 columns will repeat itself 200 more times, while the last column will go from 0 (in the first row) to the normalized maximum time of that sample (in the last row). By creating one layer for each of the 1300 samples, it is possible to separate the training, validation, and test data sets by collecting 75% of the layers to the training set, 15% of the layers to the validation set, and the last 15% to the test set.

For the Potential Curve Neural Network, the Battery potential value points were extracted from the *dynamic_files*, each sample had 201 points in time. Therefore, the *battery_potential_list* is a nested list (list of lists), the outer list has 1300 lists, and the inner list has 201 numbers. Due to the nature of the battery potential and the calculation from Raichu, there were some points in time when the potential was negative, -inf or was a NaN. These values are impractical to work in a NN training. As a result, all the negative and NaN value points were turned into zero. The code performing this is presented in Figure 12. The final *y_data* will be a nested list, with each inner list containing 201 Battery Potential data points.

```
# Making all negative values in zero
for battery_potential in battery_potential_list:
  battery_potential[battery_potential <= 0] = 0
  battery_potential[battery_potential == np.nan] = 0
```

***Figure 12**: The code part where all the negative and NaN potential values were turned into zero.*

Having outlined the theoretical approach for utilizing all samples, the actual implementation involved a similar methodology with one key modification: instead of collectively analyzing the samples, they were divided into two groups based on their C-rates, *i.e.* positive or negative. The reason for this is that the different groups have very distinct patterns/curves, especially at the extremes of time, which makes it difficult for NNs to find weights and biases that can accurately fit so extreme points. Additionally, once the data is partitioned, the samples within each group will have much more similar patterns/curves, which makes it easier for the neural network to learn the relationships between the data points. An example of a Battery potential curve of both groups is presented in Figure 13. The left curve originated from a sample with a negative C-rate, and the right one originated from a sample with a positive C-rate.

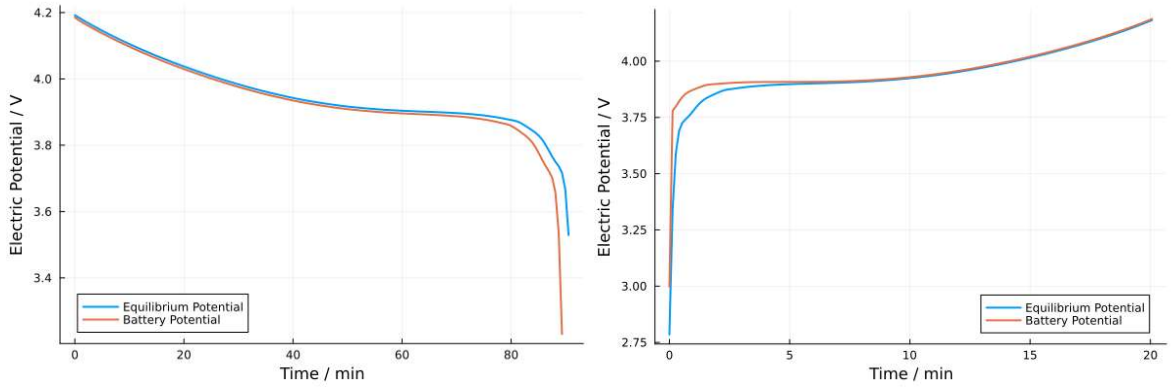

***Figure 13**: Two Battery potential and Equilibrium potential over time curves. **The left** originated from a sample with a **negative C-rate**, and **the right** one originated from a sample with a **positive C-rate**.*

Given the formation of two distinct groups of *x_data*, it is now necessary to train 2 different NNs, one specialized to predict curves with positive C-rates and the other for negative C-rates. Once the data has been grouped, it is still necessary to divide each group into three separate sets: training, validation, and testing.

3.3. NN Architecture Selection

The architecture of a NN, specifically the number of neurons, number of layers and the choice between GRU and LSTM layers (in the RNNs case), play a significant role in the model's capacity and expressiveness. Increasing the number of neurons and layers allows the network to capture complex patterns and relationships, however, this comes with the risk of overfitting. Smaller architectures require less computational power and time but may sacrifice representation power. Choosing the right balance between model complexity and generalization is crucial for successful NNs. Activation functions are another crucial aspect of the neural network architecture. These functions introduce nonlinearity, enabling the network to approximate

and represent nonlinear relationships between inputs and outputs. The choice of activation functions can impact the learning speed and convergence behavior of the model. Commonly used activation functions include sigmoid, tanh, and ReLU (Rectified Linear Unit), each with its advantages and drawbacks in specific applications [81].

Due to the vast number of variables that need to be carefully controlled and optimized, identifying manually the optimal hyperparameter for your specific data set can be a challenging and time-consuming task. As a result, multiple search algorithms were created, *e.g.*, grid search, random search, and Bayesian optimization. In this thesis, the hyperband search algorithm was implemented [76].

Hyperband is a meta-algorithm for hyperparameter optimization that has gained significant popularity in recent years. It is a hybrid approach that combines elements of random search and bandit-based optimization to effectively search for the best hyperparameter configuration. It is known for its efficiency in terms of computational resources, and it aims to identify promising configurations quickly and allocate more resources to those that show potential. Hyperband can provide over an order-of-magnitude speedup over the popular Bayesian optimization methods on a variety of deep-learning and kernel-based learning problems [76].

3.4. NN Training

The NN training process involves iteratively adjusting the network's weights and biases to minimize the loss, which can be the MSE or MAE, between predicted and actual outputs. In this thesis, the weights and bias optimization algorithm implemented was the adaptive moment estimation (Adam) [69], and the loss was the MAE.

The NNs architectures and the learning rate for the Adam optimizer were already defined in the previous step, therefore, the most important variables that needed to be set during the training process were the number of epochs and batch size. An epoch represents a single complete pass through all the training data set. This means that all the data samples in the training set have been presented to the neural network for the purpose of learning patterns and updating the NNs weights and biases. Setting the numbers of epochs excessively high or excessively low can lead to suboptimal training outcomes. Setting an extremely high number of epochs, such as 10^{80}, will result in the NN converging to a result way before the training ends. This will waste computational resources and time as the NN continues to train, even though it has already reached its optimal performance. Conversely, setting an extremely low number of epochs, such as 10, may prevent the NN from reaching its full potential. The NN

may not have enough training iterations to converge to the lowest possible loss, resulting in a less accurate and effective model.

The number of epochs for the KPIs NN, and the 2 Potential Curve NNs, were set to 2000. Additionally, an early stop was placed. The early stop will track the variation of the validation Loss throughout the iterations, if the Loss value stops to change significantly or starts to increase substantially an early stop will happen in the training process, this will prevent the NN from overfitting in the training set and prevent from unnecessary computational time.

The batch size is the number of training samples processed before the model's weights and biases are updated. It can play a significant role in the training process, influencing the convergence speed, generalization ability, and computational efficiency of the network. The choice of batch size significantly impacts the training dynamics. Smaller batch sizes tend to lead to more gradual changes in the model's parameters, potentially promoting better generalization [82]. However, they also require more training iterations, which can be computationally expensive. Larger batch sizes, on the other hand, can accelerate training by reducing the number of iterations required [83]. However, they may introduce more noise into the parameter updates, potentially leading to overfitting. It was found that the optimal batch size can range between 16 and 256 depending on the loss function and the data set [84]. After comprehensive testing, a batch size of 64 demonstrated the most favorable performance on the data set employed in this thesis. Therefore, this value was adopted for computations. The results of the three Neural Networks training, including their evaluation metrics and graphical representations, will be presented in the next section, "Results and Discussion".

4. Results and Discussion

This section delves into the specific findings for each NN, providing a comprehensive analysis and interpretation of the results. Towards the end of the section, an interesting comparison between the networks' performance is presented, as well as a performance comparison between the Raichu and the NNs. The TensorFlow python package [77] was used to perform all the NN's training, including the NN construction, Adam optimizer placement and hyperparameters iterations.

4.1. NN Architecture Selection Results

This section provides the search area and results of the Hyperband algorithm implementation.

4.1.1. NN Architecture Selection Results – KPIs Neural Network

The hyperparameters, as well as the hyperparameters search area used in the Hyperband algorithm for the KPIs Neural Network, are presented in Table 7. The hyperband algorithm code used was made available by the Keras python package [85].

***Table 7**: Hyperparameters and search area used in the Hyperband search algorithm for the KPIs Neural Network.*

Hyperparameters	Search area
Number of Layers	Min value = 1, Max Value = 4, step = 1
Number of neurons	Min value = 14, Max Value = 56, step = 14
Activation function	ReLU, tanh
Learning rate of the Adam optimizer	Min value = 10^{-4}, Max Value = 10^{-2}, sampling = "log"

The Hyperband had the objective of finding the hyperparameters that produced the lowest Mean squared error (MSE) in a maximum of 50 epochs using the training and validation data sets. In Table 8, the hyperparameters of the best performing (lowest MSE) architecture when the search was finished are presented.

***Table 8**: The hyperparameters of the best performing (lowest MSE) architecture for the KPIs NN when the Hyperband search was finished.*

Hyperparameter	Value
Layer 1	42 neurons, tanh
Layer 2	28 neurons, tanh
Layer 3	14 neurons, tanh
Layer 4	14 neurons, tanh
Learning rate	$5.9 \cdot 10^{-3}$

It is possible to see in Table 8 that the best architecture has 4 hidden layers, and it decreases the number of neurons layer by layer, which is in fact a common approach due to the concept of feature extraction. As the network goes deeper, the higher-level features can be represented by a smaller number of neurons. This allows the network to learn more abstract and complex representations of the input data, which can lead to better generalization and performance [86]. Additionally, reducing the number of neurons in deeper layers can help prevent overfitting and improve computational efficiency [87].

4.1.2. NN Architecture Selection Results – Potential Curve Neural Network

The hyperparameters, as well as the hyperparameters search area used in the Hyperband algorithm for the Potential Curve Neural Network, are presented in Table 9.

***Table 9**: Hyperparameters and search area used in the Hyperband search algorithm for the Potential Curve Neural Network.*

Hyperparameters	Search area
Type of RNN Layer	GRU, LSTM
Number of RNN Layers	Min value = 1, Max Value = 4, step = 1
Number of RNN neurons	60, 80, 100, 120, 130
Number of Dense neurons	50, 60, 80, 100
Activation function	ReLU, tanh
Learning rate of the Adam optimizer	Min value = 10^{-4}, Max Value = 10^{-2}, sampling = "log"

The Hyperband had the objective of finding the hyperparameters that produced the lowest Mean squared error (MSE) in a maximum of 100 epochs using the training and validation data sets. The hyperparameters of the best performing (lowest MSE) architecture when the search was finished are presented in Table 10. It is possible to see in the Table that the best

architecture has 4 Gated Recurrent Unit (GRU) hidden layers and it almost follows the concept of feature extraction as well.

***Table 10**: The hyperparameters of the best performing (lowest MSE) architecture when the Hyperband search was finished.*

Hyperparameter	Value
GRU Layer 1	130 neurons, tanh
GRU Layer 2	100 neurons, tanh
GRU Layer 3	60 neurons, tanh
GRU Layer 4	80 neurons, ReLU
Dense Layer	50 neurons, ReLU
Learning rate	$2.3 \cdot 10^{-3}$

4.2. KPIs Neural Network Training and Results

Before delving into the training and outcomes of the neural network, it is insightful to first examine the characteristics of the test data. A summary of the key statistical measures for the test data, extraction and voltaic efficiencies, that were generated by the LHS, is provided in Table 11.

***Table 11**: Key statistical measures for the test data extraction and voltaic efficiencies.*

Measure	Extraction efficiency	Voltaic efficiency
Maximum value	0.990	0.999
Minimum value	0.807	0.845
Range	0.183	0.154
Mean value	0.969	0.978
Median value	0.989	0.984
Variance	$1.43 \cdot 10^{-3}$	$4.74 \cdot 10^{-4}$
Standard deviation	$3.79 \cdot 10^{-2}$	$2.18 \cdot 10^{-2}$

The data in Table 11 shows that the extraction and voltaic efficiencies are relatively consistent across different samples. This is indicated by the small ranges (0.183 for extraction and 0.154 for voltaic efficiency), low variances, and standard deviations. Moreover, the median values for both efficiencies are closer to the maximum values, suggesting a left-skewed distribution, with most data points concentrated towards the higher end of the efficiency scale. The KPIs NN's training parameters that were used are presented in Table 12.

***Table 12**: Values used for the KPIs NN training parameters.*

Parameter	Value
Epochs	2000
Batch size	64
Adam optimizer learning rate	$5.9 \cdot 10^{-3}$

The evolution of MAE between the NN prediction and the real values as a function of the training epochs is shown in Figure 14. The neural network predicts Extraction and Voltaic efficiencies, therefore, the MAE values presented in Figure 14 represent the average of the MAE values of each efficiency, it is important to remember that the efficiencies are dimensionless.

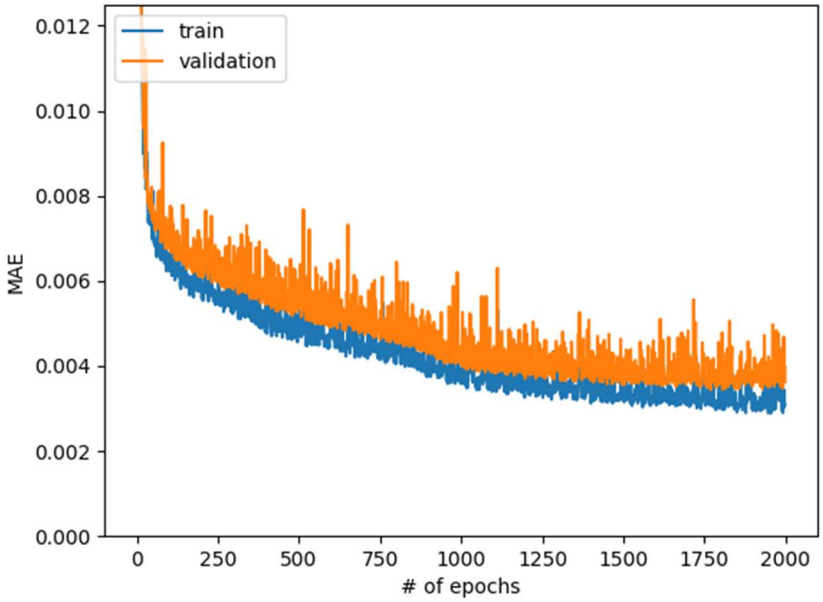

***Figure 14**: Evolution of train and validation MAE through epochs for the KPIs NN training.*

Both the training and validation sets exhibited a convergence trend in the MAE values towards the end of the training process. The trained NN model was then applied to predict the KPIs from the unseen test data. A summary of the MAE and MSE for the training and test data sets is presented in Table 13.

***Table 13**: Summary of the mean absolute error (MAE) and mean squared error (MSE) of the training and test data for the KPIs NN model.*

Measure	Training Data	Test Data
Extraction Efficiency MAE	$3.38 \cdot 10^{-3}$	$3.93 \cdot 10^{-3}$
Extraction Efficiency MSE	$4.31 \cdot 10^{-5}$	$5.46 \cdot 10^{-5}$
Voltaic Efficiency MAE	$2.80 \cdot 10^{-3}$	$2.77 \cdot 10^{-3}$
Voltaic Efficiency MSE	$1.84 \cdot 10^{-5}$	$1.96 \cdot 10^{-5}$

It's insightful to compare the NN model's MAE to the data set's standard deviation. The NN model's MAEs, $3.93 \cdot 10^{-3}$ for the Extraction efficiency and $2.77 \cdot 10^{-3}$ for the Voltaic efficiency, are both one order of magnitude lower than the data set's standard deviation, which means that the model's prediction has less deviation from the true values compared to the overall variability in the data set, *i.e.* the model is making informed estimations, not simply selecting random numbers within the dataset's boundaries.

Once trained, the KPIs NN model is able to predict the Extraction efficiency and Voltaic efficiency of solid-state batteries using any set of the 14 input variables in matter of milliseconds, with a MAE in the order of 10^{-3}. NNs typically exhibit reduced performance when predicting using input values outside the range of their training and test data [88]. Therefore, to ensure optimal functionality, all 14 input variables must be within the range specified in Table 6.

4.3. Potential Curves Neural Network Training and Results

As said previously, for the Battery potential curves two Neural networks were developed, one specialized in predicting when the Battery will be charged (*i.e.*, positive C-rate) and the other in predicting when the Battery will be discharged (*i.e.*, negative C-rate), therefore, it is necessary to present the results and findings separately.

4.3.1. Positive C-rate Neural Network

A summary of the key statistical measures for the test data extraction and voltaic efficiencies that were generated by the LHS is provided in Table 14.

Table 14: *Key statistical measures for the Battery potentials test data.*

Measure	Battery Potential / V
Maximum value / V	6.065
Minimum value / V	2.784
Range / V	3.281
Mean value / V	4.009
Median value / V	3.981
Variance / V^2	0.059
Standard deviation / V	0.242

The training parameters that were used for the Positive C-rate NN are presented in Table 15.

***Table 15**: Values used for the positive C-rate Potential Curves NN training parameters.*

Parameter	Value
Epochs	2000
Batch size	64
Adam optimizer learning rate	$2.3 \cdot 10^{-3}$

The evolution of MAE between the NN prediction and the real values as a function of the training epochs is shown in Figure 15.

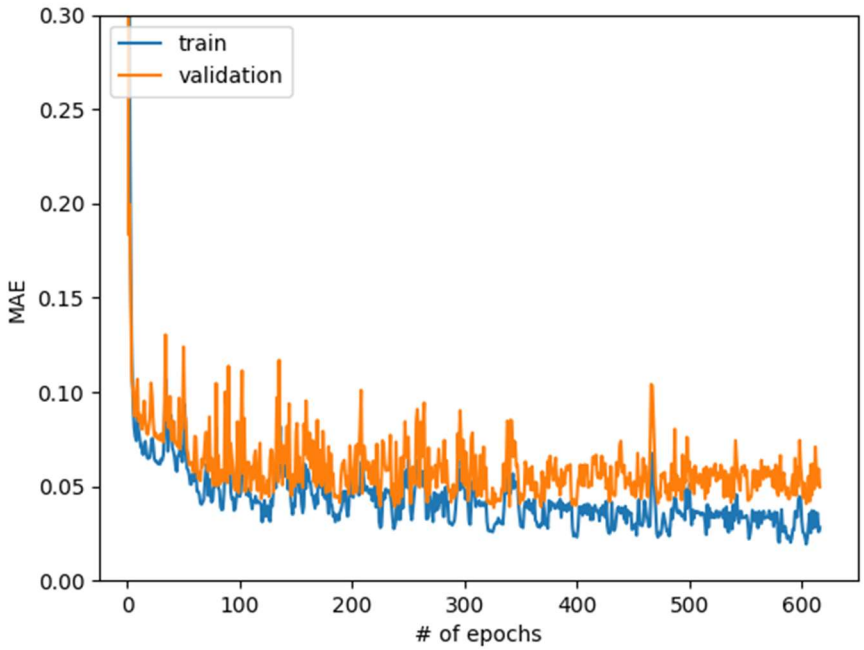

***Figure 15**: Evolution of train and validation MAE through training epochs for the positive C-rate Potential Curves NN.*

As can be seen from Figure 15, the model was trained for a total of 617 epochs, as the early stopping mechanism, which was explained previously, interrupted the process before reaching the target of 2000 epochs. To investigate the effects of prolonged training, and out of curiosity, the early stopping mechanism was temporarily disabled, and the training was extended to the original target of 2000 epochs. Figure 16 shows the evolution of MAE throughout the rest of the training.

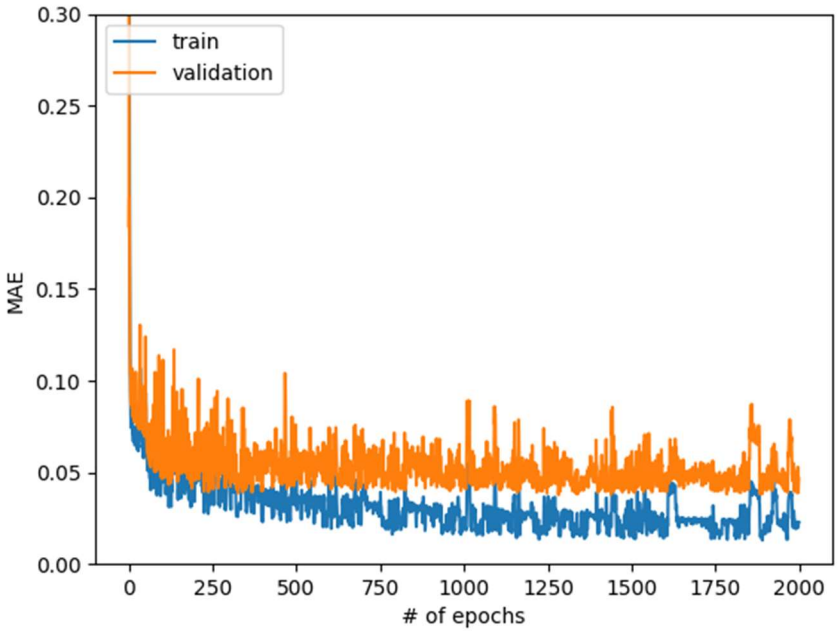

***Figure 16**: Evolution of train and validation MAE through 2000 training epochs for the positive C-rate Potential Curves NN, the training was resumed after the original stop at epoch 617.*

From Figure 16 it is possible to conclude that the MAE values reached a plateau after the epoch 500 region, indicating that further training would not yield significant improvements. Therefore, stopping the training process at epoch 617 was a wise decision to avoid wasting time and computer power. Following the 2000 training epochs, the NN was then applied to predict the potential through time curves from the unseen test data. A summary of the MAE and MSE for the training and test data sets is presented in Table 16.

***Table 16**: Summary of the mean absolute error (MAE) and mean squared error (MSE) of the training and test data for the Potential Curves NN model for positive C-rates.*

Measure	Training Data	Test Data
Mean absolute error (MAE) / V	$3.14 \cdot 10^{-2}$	$5.57 \cdot 10^{-2}$
Mean squared error (MSE) / V	$2.25 \cdot 10^{-2}$	$4.13 \cdot 10^{-2}$

It's insightful to compare the NN model's MAE to the data set's standard deviation. The NN model's MAE, $5.57 \cdot 10^{-2}$ V, is one order of magnitude lower than the data set's standard deviation, 0.242 V, which means that the model's prediction has less deviation from the true values compared to the overall variability in the data set, *i.e.* the model is making informed estimations, not simply selecting random numbers within the dataset's boundaries. To illustrate the range of performance achieved by the NN model, three representative predicted

curves were selected based on their individual MAE values: the "Curve 1", the "Curve 2", and the "Curve 3". These terms were chosen arbitrarily and represent, respectively, the curve with the lowest MAE, $2.92 \cdot 10^{-3}$ V, the curve with a MAE closest to the MAE of the test data, $5.70 \cdot 10^{-2}$ V, and the curve with the highest MAE, $1.57 \cdot 10^{-1}$ V. The curves are presented in Figure 17.

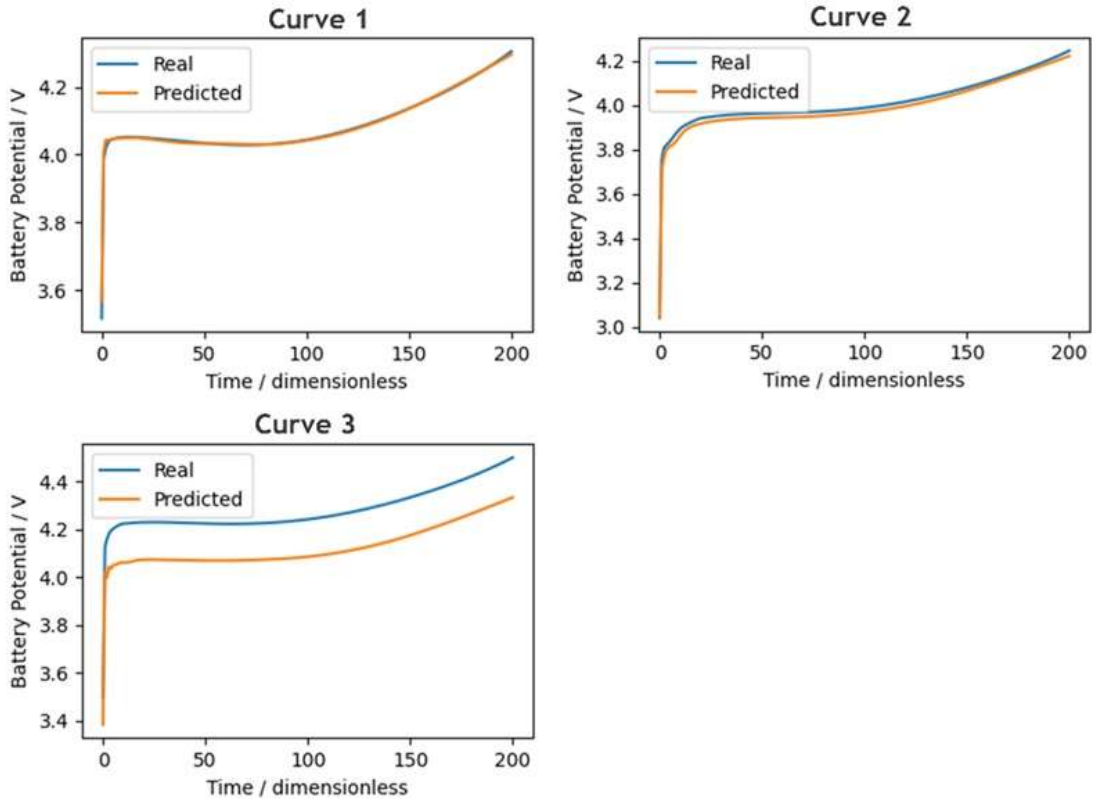

***Figure 17**: Representation of the Battery potential "Curve 1", "Curve 2", and " Curve 3". These terms were chosen arbitrarily and represent, respectively, the curve with the lowest MAE, 2.92·10-3 V, the curve with a MAE closest to the MAE of the test data, 5.70·10-2 V, and the curve with the highest MAE, 1.57·10-1 V.*

Figure 17 shows that the predicted "Curve 1" is extremely accurate with real curve, while the predicted "Curve 3" seems far for the real values. However, despite the deviation in the predicted "Curve 3", the overall shape of it remains remarkably similar to the actual curve. Therefore, even though the network didn't predict the values of the "Curve 3" well, it was still able to predict the curve pattern, *i.e.* the rate of change of the potential. While this may seem like a positive attribute, it can actually be more detrimental than helpful in real-world applications. This is because users will not have access to the ground truth potential curve for comparison. As the predicted curve closely resembles the actual curve, users may mistakenly believe that the NN has made an accurate prediction. However, if the curve with the highest mean absolute error were clearly erroneous, it would be easier to identify a suboptimal prediction.

4.3.2. Negative C-rate Neural Network

A summary of the key statistical measures for the test data extraction and voltaic efficiencies is provided in Table 17.

***Table 17**: Key statistical measures for the Battery potentials test data.*

Measure	Battery Potential
Maximum value / V	4.192
Minimum value / V	0
Range / V	4.192
Mean value / V	3.609
Median value / V	3.885
Variance / V^2	1.017
Standard deviation / V	1.008

The training parameters that were used for the NN are presented in Table 18.

***Table 18**: Values used for the positive C-rate Potential Curves NN training parameters.*

Parameter	Value
Epochs	2000
Batch size	64
Adam optimizer learning rate	$2.3 \cdot 10^{-3}$

The evolution of MAE between the NN prediction and the real values as a function of the training epochs is visualized in Figure 18.

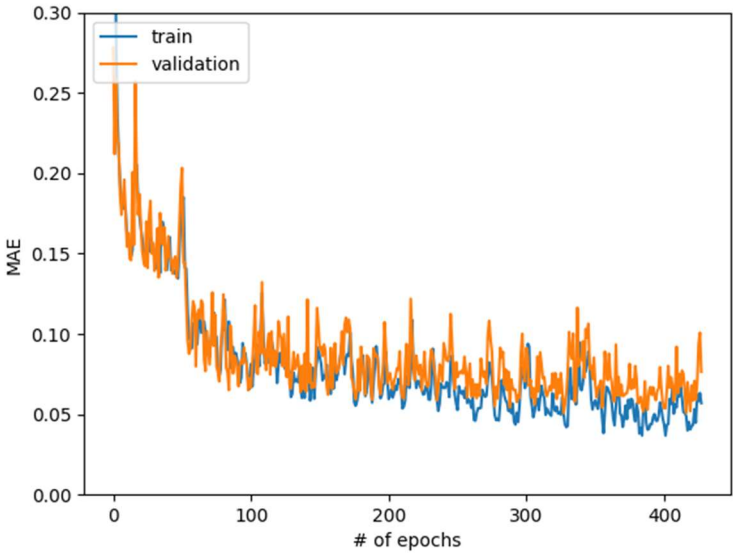

***Figure 18**: Evolution of train and validation MAE through epochs for the KPIs NN training.*

As can be seen from Figure 18, the model was trained for a total of 428 epochs, as the early stopping mechanism interrupted the process before reaching the target of 2000 epochs. The trained NN model was then applied to predict the potential through time curves from the unseen test data. A summary of the MAE and MSE for the training and test data sets is presented in Table 19.

Table 19: *Summary of the mean absolute error (MAE) and mean squared error (MSE) of the training and test data for the Potential Curves NN model for negative C-rates.*

Measure	Training Data	Test Data
Mean absolute error (MAE) / V	$3.76 \cdot 10^{-2}$	$6.73 \cdot 10^{-2}$
Mean squared error (MSE) / V	$7.22 \cdot 10^{-2}$	$1.77 \cdot 10^{-1}$

Initially, it's insightful to compare the NN model's MAE to the data set's standard deviation. The NN model's MAE is one order of magnitude lower than the data set's standard deviation, which means that the model's prediction has less deviation from the true values compared to the overall variability in the data set, *i.e.* the model is making informed estimations, not simply selecting random numbers within the dataset's boundaries. To illustrate the range of performance achieved by the NN model, 3 representative predicted curves were selected based on their individual MAE values: the "Curve 1", the "Curve 2", and the "Curve 3". These terms were chosen arbitrarily and represent, respectively, the curve with the lowest MAE, $3.98 \cdot 10^{-3}$ V, the curve with an MAE closest to the MAE of the test data, $6.35 \cdot 10^{-2}$, and the curve with the highest MAE, $5.8 \cdot 10^{-1}$ V. The curves are presented in Figure 19.

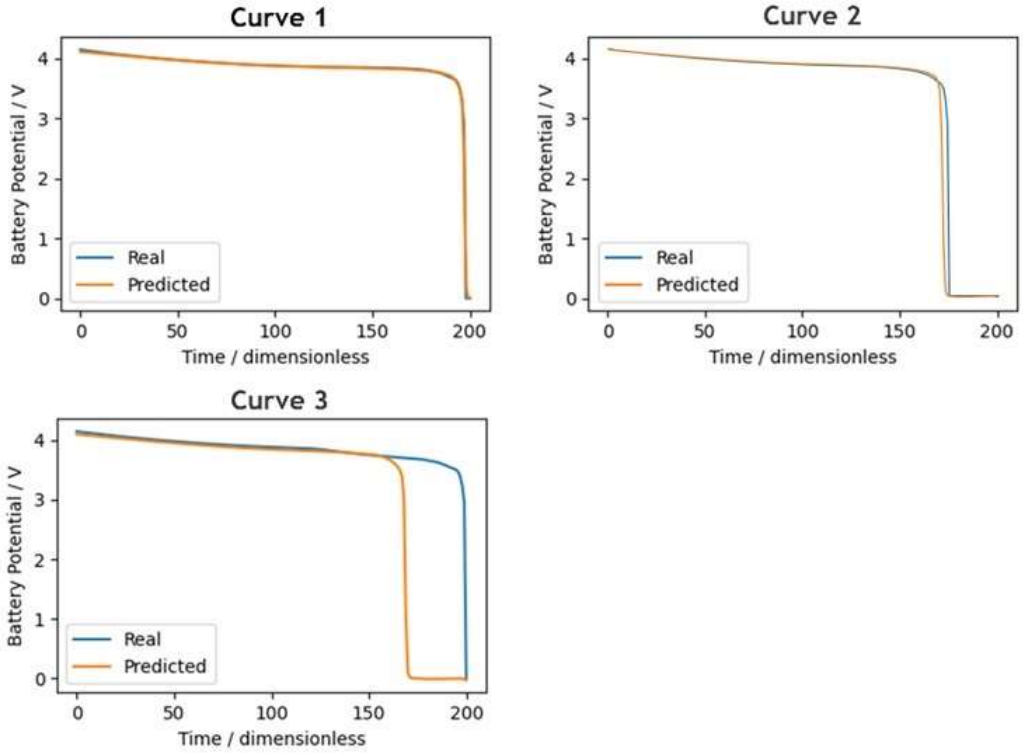

***Figure 19**: Representation of the Battery potential "Curve 1", "Curve 2", and " Curve 3". These terms were chosen arbitrarily and represent, respectively, the curve with the lowest MAE, 3.98·10-3 V, the curve with an MAE closest to the MAE of the test data, 6.35·10-2, and the curve with the highest MAE, 5.8·10-1 V.*

Figure 19 shows that the predicted "Curve 1" is extremely accurate to the real curve, while the predicted "Curve 3" seems far for the real values. However, in this case not in terms of the values of the Potential but in time. Nevertheless, despite the deviation in the predicted "Curve 3", the overall shape of it remains remarkably similar to the actual curve before the potential drop. Therefore, even though the network didn't predict the timing of the "Curve 3" well, it was still able to predict the curve pattern. Similarly to the previous case, while this may seem like a positive attribute, it can actually be more detrimental than helpful in real-world applications. This is because users will not have access to the ground truth potential curve for comparison. As the predicted curve closely resembles the actual curve, users may mistakenly believe that the NN has made an accurate prediction. However, if the curve with the highest mean absolute error were clearly erroneous, it would be easier to identify a suboptimal prediction.

4.4. Neural Networks Accuracy Comparison

Using the data predicted by the Potential Curves NNs it is possible to calculate the voltaic efficiency of the Battery. Therefore, an interesting experiment is to compare the KPIs NN and

the Potential curves NNs, to determine the most effective method for predicting battery voltaic efficiency. This experiment will compare the accuracy of a NN that directly predicts efficiency to the accuracy of a NN that indirectly calculates it based on the predicted battery potential curve.

Using the test data to generate predicted data and comparing them with the real values, the MAE and MSE shown in Table 20 were calculated. Since the test data space was divided into positive and negative C-rates for the Potential Curve NNs, a column for the mean of these NN's values was added to Table 20.

***Table 20**: MAE and MSE values for voltaic efficiencies calculated/predicted by the positive C-rates Potential Curve NN, negative C-rates Potential Curve NN, mean of both, and the KPIs NN.*

Measure	Positive C-rate NN	Negative C-rate NN	Potential Curve NN mean	KPIs NN
MAE	$9.32 \cdot 10^{-3}$	$2.73 \cdot 10^{-2}$	$1.83 \cdot 10^{-2}$	$2.77 \cdot 10^{-3}$
MSE	$1.43 \cdot 10^{-3}$	$1.49 \cdot 10^{-2}$	$8.16 \cdot 10^{-3}$	$1.96 \cdot 10^{-5}$

As shown in Table 20, the KPIs NN exhibits a MAE that is one order of magnitude lower than that of the Potential Curve NN. This can be attributed to the fact that the Potential Curve NN already exhibited higher MAE and MSE values when predicting potential profiles, as it can be seen in tables 16 and 19 (*i.e.*, in the order of 10^{-2}), compared to the MAE and MSE values of the KPIs NN, as it can be seen in table 13 (*i.e.*, in the order of 10^{-3} and 10^{-5} respectively). Since KPI values are calculated through potential profiles, the error tends to be in the same range. Furthermore, the KPIs NN's MSE is two orders of magnitude lower than that of the Potential Curve NN, again due to the same underlying reason. Therefore, it is evident that developing a NN specifically designed to predict battery KPIs will lead to more precise results than indirectly calculating them through a NN that predicts battery potential curves.

4.5. Raichu and Neural Networks Speed Comparison

As said in the "Purpose and Research Motivation" section of this thesis, one of the primary objectives of this study was to examine the potential of NNs to enhance the modelling of technologies such as solid-state batteries, and this enhancement could manifest as faster execution. Therefore, this section compares the speed of generating the battery's potential curves and KPIs data using Raichu to the speed of generating them using the NN models developed in this thesis. While the comparison focuses on predicting battery potential curves and KPIs, it is important to note that Raichu generates a broader range of data. This analysis is particularly relevant for those seeking to utilize battery models solely for the battery potential curves and KPIs generation. Additionally, it serves as a basis for suggesting that

constructing multiple neural networks, each trained to predict distinct data generated by Raichu, may potentially outperform, in terms of speed, utilizing Raichu directly.

First, 5 samples were generated using the LHS method using the same 14 input variables' ranges as presented in Table 6, therefore, the data points are new for both Raichu and the NNs. Then, the time for Raichu to generate the data from these 5 samples was measured. Next, using the same samples, the time to the Potential Curve NN to predict the battery potential and the time to the KPIs NN to predict the KPIs was measured.

The entire Raichu process took approximately 19 minutes and 10 seconds. This translates to approximately 230 seconds per sample. The Potential Curve NN and KPIs NN took a combined total of 773 milliseconds, 511 milliseconds from the Potential Curve NN and 262 milliseconds from the KPI NN, predicting the same 5 samples, which equates to around 155 milliseconds per sample. Therefore, this is equivalent to around 1475 times faster. Table 21 presents the process's duration.

***Table 21**: Total time and time per sample comparison between the Raichu and the NNs for a 5 data samples set.*

	Raichu	KPIs NN + Potential Curve NN
Total time	19 min	773 ms
Time per sample	230 s	155 ms

Finally, the MAE and MSE of the Battery Potential Curves and KPIs predicted for the 5 samples were calculated. The results are shown in Table 22.

***Table 22**: MAE and MSE of the Potential Curve NN and KPIs NN for the 5 data samples set.*

Measure	Potential Curve NN	KPIs NN
MAE	$8.97 \cdot 10^{-2}$ V	$3.97 \cdot 10^{-2}$
MSE	$4.62 \cdot 10^{-2}$ V^2	$2.75 \cdot 10^{-5}$

To further illustrate the predicted capacity of the NNs, Table 23 contains two of the five samples done on this section. Table 23 shows real KPIs values, the predicted sample with the highest MAE value (Sample 5) and the predicted sample with the lowest MAE value (Sample 2).

Table 23: Comparison of the Real and predicted values for Extraction and Voltaic efficiencies of the sample with the highest MAE value (Sample 5) and the lowest MAE value (Sample 2).

	Real Sample 5	Pred. Sample 5	Real Sample 2	Pred. Sample 2
Extraction Efficiency	0,989	0,983	0,989	0,986
Voltaic Efficiency	0,965	0,979	0,996	0,996
MAE	$9{,}49 \cdot 10^{-3}$		$1{,}75 \cdot 10^{-3}$	

Additionally, Figure 20 contains the predicted battery's potential curves that got the highest and lowest MAE. The Left curve had a MAE of 0.07 V, while the Right curve had a MAE of 0.02 V.

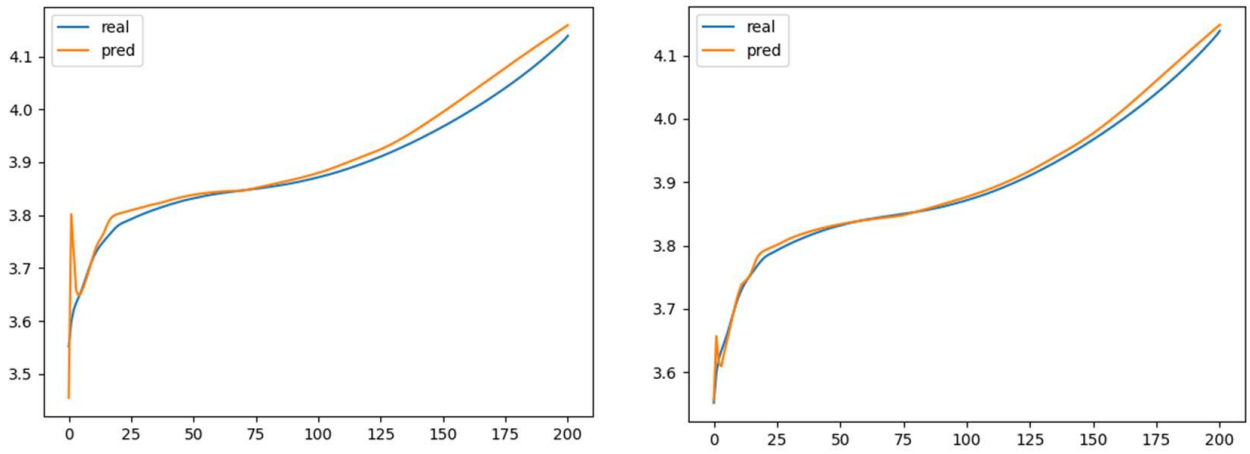

Figure 20: Representation of the predicted battery's potential curves that got the highest and lowest MAE. The Left curve had a MAE of 0.07 V, while the Right curve had a MAE of 0.02 V.

Analyzing Tables 21, 22, and 23 and Figure 20 it is possible to conclude that the NNs developed in this thesis demonstrated a very good accuracy in predicting both KPIs and Battery Potential curves, while significantly outperforming the Raichu in terms of computational time in several orders of magnitude, around 1475 times faster. This comparison highlights the immense potential that NNs and ML techniques hold in modeling solid-state batteries and advancing the Chemical Engineering field as a whole.

5. Conclusions

In conclusion, this master thesis has explored the integration of SciML techniques in Chemical Engineering and its potential applications. The study has highlighted the increasing importance of ML in tackling complex engineering problems and optimizing chemical processes. Throughout this research, the primary focus was on the development and evaluation of NN models to predict data generated by a solid-state batteries model created by Iwakiri [1]. Three distinct NN models were developed: the Key Performance Indicators (KPIs) Neural Network and two Potential Curve Neural Networks. These models were meticulously trained, tested, and optimized to achieve high accuracy and efficiency in the predictions.

The KPIs NN demonstrated great capability in directly predicting the extraction and voltaic efficiencies of solid-state batteries. The results showed that this model achieved a MAE in the order of 10^{-3}, and a MSE in the order of 10^{-5} on test data. This level of accuracy highlights the model's reliability in providing precise predictions of the Battery's KPIs. Similarly, the Potential Curve Neural Network, developed in two variants (*i.e.*, chargers and discharges of the Battery), effectively predicted the battery potential over time curves. Both Potential Curve NNs achieved an MAE and MSE in the order of 10^{-2} V. However, the comparison of the predicted curves against the real ones in Figures 17 and 19 reveals that, if used alone, the Potential Curves NNs should not be relied upon for accurate battery potential curves, particularly when seeking extremely precise predictions. Nevertheless, due to their exceptional computational speed, and with acceptable generalization performance, they can be valuable tools for comparing batteries with distinct properties. If further accuracy is required, users can always cross-check the information with a more precise model, such as Raichu.

Two comparison experiments were done. The first, between the KPIs NN and the Potential Curves NNs, showed that Battery's KPIs predicted by a NN dedicated exclusively for that is more accurate than KPIs calculated indirectly by predicted Battery's Potential curves. The KPIs NN achieved a MAE of $2.77 \cdot 10^{-3}$ while the Potential Curves NNs achieved a MAE of $1.83 \cdot 10^{-2}$. The second comparison was between traditional methods and the NN models developed in this thesis, it revealed that the NNs outperforms Raichu in generating battery's data, with a marked increase in speed by several orders of magnitude, around 1475 times faster. This comparative analysis served not only to validate the models' performance but also to provide insights into the most effective approaches for specific prediction tasks within the solid-state battery domain.

These NN models will be useful for the internship institution, the Norwegian University of Science and Technology (NTNU), by the fact that the models have shown remarkable accuracy

and efficiency, specially the KPIs Neural Network. The developed NNs can be valuable assets for researchers and practitioners in the field, enabling future studies to leverage and enhance them for diverse battery model analyses. The outcome of this study is to help encourage the exploration and innovation of new options in the realm of Chemical Engineering.

6. Assessment of the work done

6.1. Objectives Achieved

The main objective of this thesis was to explore the implementation of SciML techniques within the domain of Chemical Engineering. Utilizing the data generated by Iwakiri's [1] solid-state battery model, three Neural Networks were successfully constructed, replicating key features of the original model. Both Feed Forward Neural Networks (FFNNs) and Recurrent Neural Networks (RNNs) were extensively evaluated throughout the research. The aim of improving the modelling process was achieved, resulting in a significant enhancement in computational speed, which improved in several orders of magnitude, around 1475 times faster.

6.2. Final Assessment

Overall, the developed work was a success. It showed the feasibility of implementing SciML techniques in the realm of Chemical Engineering. The Neural Networks constructed in this dissertation perform effectively at some level and can be employed in future projects and investigations. However, it still had some weaknesses. The primary limitations of this dissertation were time related. It was feasible to generate more data than the 1300 samples utilized, and training the NNs with more data may have yielded superior outcomes. Moreover, it would have been beneficial to the discussion to have more NNs for comparative analysis or to implement an Operator in some manner. However, generating each sample was time-consuming, and achieving convergence to a good NN output was not as straightforward as it may seem. Future works can address these limitations by generating additional data samples using Raichu and developing surrogate NNs for other Raichu's outputs.

7. References

1. Iwakiri, I.G.I., *Solid-state Battery Model [Internal NTNU report]*. 2023: Trondheim.
2. Dobbelaere, M.R., et al., *Machine learning in chemical engineering: strengths, weaknesses, opportunities, and threats*. Engineering, 2021. **7**(9): p. 1201-1211.
3. Venkatasubramanian, V., *The promise of artificial intelligence in chemical engineering: Is it here, finally?* AIChE Journal, 2019. **65**(2): p. 466-478.
4. McCulloch, W.S. and W. Pitts, *A logical calculus of the ideas immanent in nervous activity*. The bulletin of mathematical biophysics, 1943. **5**: p. 115-133.
5. Pugliese, R., S. Regondi, and R. Marini, *Machine learning-based approach: Global trends, research directions, and regulatory standpoints*. Data Science and Management, 2021. **4**: p. 19-29.
6. Roscher, R., et al., *Explainable machine learning for scientific insights and discoveries*. Ieee Access, 2020. **8**: p. 42200-42216.
7. Bertolini, M., et al., *Machine Learning for industrial applications: A comprehensive literature review*. Expert Systems with Applications, 2021. **175**: p. 114820.
8. Brotherton, T. and T. Johnson. *Anomaly detection for advanced military aircraft using neural networks*. in *2001 IEEE Aerospace Conference Proceedings (Cat. No. 01TH8542)*. 2001. IEEE.
9. de Oliveira, F.M., et al., *Predicting cetane index, flash point, and content sulfur of diesel–biodiesel blend using an artificial neural network model*. Energy & Fuels, 2017. **31**(4): p. 3913-3920.
10. Li, Z., et al., *Fourier neural operator for parametric partial differential equations*. arXiv preprint arXiv:2010.08895, 2020.
11. Adumene, S. and H. Ikue-John, *Offshore system safety and operational challenges in harsh Arctic operations*. Journal of safety science and resilience, 2022. **3**(2): p. 153-168.
12. Tiryaki, S., Ş. Özşahin, and İ. Yıldırım, *Comparison of artificial neural network and multiple linear regression models to predict optimum bonding strength of heat treated woods*. International Journal of Adhesion and Adhesives, 2014. **55**: p. 29-36.
13. Pratama, D., A, M. A. Bakar, and N. Ismail, *ANN-based methods for solving partial differential equations: a survey*. Arab Journal of Basic and Applied Sciences, 2022. **29**(1): p. 233-248.
14. Faroughi, S.A., A.I. Roriz, and C. Fernandes, *A meta-model to predict the drag coefficient of a particle translating in viscoelastic fluids: a machine learning approach*. Polymers, 2022. **14**(3): p. 430.

15. Lagergren, J.H., et al., *Learning partial differential equations for biological transport models from noisy spatio-temporal data*. Proceedings of the Royal Society A, 2020. **476**(2234): p. 20190800.
16. Chen, J., J. Viquerat, and E. Hachem, *U-net architectures for fast prediction of incompressible laminar flows*. arXiv preprint arXiv:1910.13532, 2019.
17. Lu, L., et al., *Physics-informed neural networks with hard constraints for inverse design*. SIAM Journal on Scientific Computing, 2021. **43**(6): p. B1105-B1132.
18. Smith, J.D., Azizzadenesheli, K., & Ross, Z.E. *Fourier Neural Operators for Parametric Partial Differential Equations*. in *International Conference on Learning Representations (ICLR)*. 2021
19. Ruthotto, L. and E. Haber, *Deep neural networks motivated by partial differential equations*. Journal of Mathematical Imaging and Vision, 2020. **62**: p. 352-364.
20. Borisova, J., et al., *Forecasting of Sea Ice Concentration using CNN, PDE discovery and Bayesian Networks*. Procedia Computer Science, 2023. **229**: p. 177-187.
21. Bishop, C.M., *Mixture density networks*. 1994: Birmingham.
22. Foong, A.Y., et al., *'In-Between' Uncertainty in Bayesian Neural Networks*. arXiv preprint arXiv:1906.11537, 2019.
23. Raissi, M., P. Perdikaris, and G. Karniadakis, *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. arXiv preprint arXiv:1711.10561, 2017.
24. Raissi, M., P. Perdikaris, and G. Karniadakis, *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. arXiv preprint arXiv:1711.10566, 2017.
25. Li, Z., et al., *Physics-informed neural operator for learning partial differential equations*. arXiv preprint arXiv:2111.03794, 2021.
26. Svozil, D., V. Kvasnicka, and J. Pospichal, *Introduction to multi-layer feed-forward neural networks*. Chemometrics and intelligent laboratory systems, 1997. **39**(1): p. 43-62.
27. Zafeiris, D., S. Rutella, and G.R. Ball, *An artificial neural network integrated pipeline for biomarker discovery using Alzheimer's disease as a case study*. Computational and Structural Biotechnology Journal, 2018. **16**: p. 77-87.
28. Pedro, J.B., J. Maroñas, and R. Paredes, *Solving partial differential equations with neural networks*. arXiv preprint arXiv:1912.04737, 2019.
29. Sherstinsky, A., *Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network*. Physica D: Nonlinear Phenomena, 2020. **404**: p. 132306.

30. Hewamalage, H., C. Bergmeir, and K. Bandara, *Recurrent neural networks for time series forecasting: Current status and future directions*. International Journal of Forecasting, 2021. **37**(1): p. 388-427.
31. Moradi, M. and M. Samwald, *Deep learning, natural language processing, and explainable artificial intelligence in the biomedical domain*. arXiv preprint arXiv:2202.12678, 2022.
32. Cahuantzi, R., X. Chen, and S. Güttel. *A comparison of LSTM and GRU networks for learning symbolic sequences*. in *Science and Information Conference*. 2023. Springer.
33. Chung, J., et al., *Empirical evaluation of gated recurrent neural networks on sequence modeling*. arXiv preprint arXiv:1412.3555, 2014.
34. Kumar, S., et al. *Energy load forecasting using deep learning approach-LSTM and GRU in spark cluster*. in *2018 fifth international conference on emerging applications of information technology (EAIT)*. 2018. IEEE.
35. Li, W., et al., *Prediction of dissolved oxygen in a fishery pond based on gated recurrent unit (GRU)*. Information Processing in Agriculture, 2021. **8**(1): p. 185-193.
36. Cheng, T., et al., *Forecasting of wastewater treatment plant key features using deep learning-based models: A case study*. IEEE Access, 2020. **8**: p. 184475-184485.
37. Lu, L., P. Jin, and G.E. Karniadakis, *Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*. arXiv preprint arXiv:1910.03193, 2020.
38. Baker, M.R. and R.B. Patil, *Universal approximation theorem for interval neural networks*. Reliable Computing, 1998. **4**: p. 235-239.
39. Lu, L., P. Jin, and G.E. Karniadakis, *Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators*. arXiv preprint arXiv:1910.03193, 2020.
40. Lin, C., et al., *Operator learning for predicting multiscale bubble growth dynamics*. The Journal of Chemical Physics, 2021. **154**(10).
41. Lu, L., et al., *A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data*. Computer Methods in Applied Mechanics and Engineering, 2022. **393**: p. 114778.
42. Lu, L., et al., *Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators*. Nature machine intelligence, 2021. **3**(3): p. 218-229.
43. Cai, S., et al., *DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks*. Journal of Computational Physics, 2021. **436**: p. 110296.

44. Goswami, S., et al., *A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials*. Computer Methods in Applied Mechanics and Engineering, 2022. **391**: p. 114587.
45. Di Leoni, P.C., et al., *Deeponet prediction of linear instability waves in high-speed boundary layers*. arXiv preprint arXiv:2105.08697, 2021.
46. Oommen, V., et al., *Learning two-phase microstructure evolution using neural operators and autoencoder architectures*. npj Computational Materials, 2022. **8**(1): p. 190.
47. Kovachki, N., et al., *Neural operator: Learning maps between function spaces*. arXiv preprint arXiv:2108.08481, 2021.
48. Jiang, P., et al., *Digital Twin Earth--Coasts: Developing a fast and physics-informed surrogate model for coastal floods via neural operators*. arXiv preprint arXiv:2110.07100, 2021.
49. Wen, G., et al., *U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow*. Advances in Water Resources, 2022. **163**: p. 104180.
50. Guibas, J., et al., *Adaptive fourier neural operators: Efficient token mixers for transformers*. arXiv preprint arXiv:2111.13587, 2021.
51. Zha, X., et al., *Specific emitter identification based on complex Fourier neural network*. IEEE Communications Letters, 2021. **26**(3): p. 592-596.
52. Guan, S., K.-T. Hsu, and P.V. Chitnis, *Fourier neural operator networks: A fast and general solver for the photoacoustic wave equation*. arXiv preprint arXiv:2108.09374, 2021.
53. You, H., et al., *Learning deep implicit Fourier neural operators (IFNOs) with applications to heterogeneous material modeling*. Computer Methods in Applied Mechanics and Engineering, 2022. **398**: p. 115296.
54. Pathak, J., et al., *Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators*. arXiv preprint arXiv:2202.11214, 2022.
55. Li, Z., et al., *Fourier neural operator approach to large eddy simulation of three-dimensional turbulence*. Theoretical and Applied Mechanics Letters, 2022. **12**(6): p. 100389.
56. Rosofsky, S.G., H. Al Majed, and E. Huerta, *Applications of physics informed neural operators*. Machine Learning: Science and Technology, 2023. **4**(2): p. 025022.
57. Krishnapriyan, A., et al., *Characterizing possible failure modes in physics-informed neural networks*. Advances in Neural Information Processing Systems, 2021. **34**: p. 26548-26560.
58. Wang, S., X. Yu, and P. Perdikaris, *When and why PINNs fail to train: A neural tangent kernel perspective*. Journal of Computational Physics, 2022. **449**: p. 110768.

59. Goswami, S., et al., *Physics-informed Deep Neural Operator Networks*. arXiv preprint arXiv:2207.05748, 2022.
60. Kaltenbach, S., P. Perdikaris, and P.-S. Koutsourelakis, *Semi-supervised invertible neural operators for Bayesian inverse problems*. Computational Mechanics, 2023: p. 1-20.
61. Maust, H., et al., *Fourier Continuation for Exact Derivative Computation in Physics-Informed Neural Operators*. arXiv preprint arXiv:2211.15960, 2022.
62. Fang, Z., S. Wang, and P. Perdikaris, *Learning Only on Boundaries: A Physics-Informed Neural Operator for Solving Parametric Partial Differential Equations in Complex Geometries*. Neural Computation, 2024. **36**(3): p. 475-498.
63. Li, Z., et al., *Physics-informed neural operator for learning partial differential equations*. arXiv preprint arXiv:2111.03794, 2023.
64. Rosofsky, S.G. and E. Huerta, *Magnetohydrodynamics with Physics Informed Neural Operators*. arXiv preprint arXiv:2302.08332, 2023.
65. Rackauckas, C., et al., *Universal differential equations for scientific machine learning*. arXiv preprint arXiv:2001.04385, 2020.
66. BR Nogueira, I., et al., *Using scientific machine learning to develop universal differential equation for multicomponent adsorption separation systems*. The Canadian Journal of Chemical Engineering, 2022. **100**(9): p. 2279-2290.
67. Chen, T. and H. Chen, *Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems*. IEEE transactions on neural networks, 1995. **6**(4): p. 911-917.
68. Vortmeyer-Kley, R., P. Nieters, and G. Pipa, *A trajectory-based loss function to learn missing terms in bifurcating dynamical systems*. Scientific Reports, 2021. **11**(1): p. 20394.
69. Kingma, D.P. and J. Ba, *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.
70. Byrd, R.H., et al., *A limited memory algorithm for bound constrained optimization*. SIAM Journal on scientific computing, 1995. **16**(5): p. 1190-1208.
71. Rackauckas, C., et al., *DiffEqFlux.jl - A julia library for neural differential equations*. arXiv preprint arXiv:1902.02376, 2019.
72. Mangan, N.M., et al., *Inferring biological networks by sparse identification of nonlinear dynamics*. IEEE Transactions on Molecular, Biological and Multi-Scale Communications, 2016. **2**(1): p. 52-63.
73. Zhong, Y.D., B. Dey, and A. Chakraborty, *Symplectic ode-net: Learning hamiltonian dynamics with control*. arXiv preprint arXiv:1909.12077, 2019.

74. Santana, V.V., et al., *Efficient hybrid modeling and sorption model discovery for non-linear advection-diffusion-sorption systems: A systematic scientific machine learning approach*. arXiv preprint arXiv:2303.13555, 2023.
75. McKay, M.D., R.J. Beckman, and W.J. Conover, *A comparison of three methods for selecting values of input variables in the analysis of output from a computer code*. Technometrics, 2000. **42**(1): p. 55-61.
76. Li, L., et al., *Hyperband: A novel bandit-based approach to hyperparameter optimization*. Journal of Machine Learning Research, 2018. **18**(185): p. 1-52.
77. Abadi, M., A. Agarwal, and P. Barham, *TensorFlow: Large-scale Machine Learning on Heterogeneous Systems, v1. 14*. 2015.
78. Taye, M.M., *Understanding of Machine Learning with Deep Learning: Architectures, Workflow, Applications and Future Directions*. Computers, 2023. **12**(5): p. 91.
79. Nunes, M., et al., *A comparison of multitask and single task learning with artificial neural networks for yield curve forecasting*. Expert Systems with Applications, 2019. **119**: p. 362-375.
80. Ciliberto, C., et al. *Convex learning of multiple tasks and their structure*. in *International Conference on Machine Learning*. 2015. PMLR.
81. Ramachandran, P., B. Zoph, and Q.V. Le, *Searching for activation functions*. arXiv preprint arXiv:1710.05941, 2017.
82. Masters, D. and C. Luschi, *Revisiting small batch training for deep neural networks*. arXiv preprint arXiv:1804.07612, 2018.
83. Smith, S.L., et al., *Don't decay the learning rate, increase the batch size*. arXiv preprint arXiv:1711.00489, 2017.
84. Shi, X., W. Cao, and S. Raschka, *Deep neural networks for rank-consistent ordinal regression based on conditional probabilities*. Pattern Analysis and Applications, 2023. **26**(3): p. 941-955.
85. Chollet, F., *Keras: The python deep learning library*. Astrophysics source code library, 2018: p. ascl: 1806.022.
86. Schmidhuber, J., *Deep learning in neural networks: An overview*. Neural networks, 2015. **61**: p. 85-117.
87. Hu, H., et al., *Network trimming: A data-driven neuron pruning approach towards efficient deep architectures*. arXiv preprint arXiv:1607.03250, 2016.
88. Mi, X., et al., *Testing the generalization of artificial neural networks with cross-validation and independent-validation in modelling rice tillering dynamics*. Ecological Modelling, 2005. **181**(4): p. 493-508.