The University of Manchester
School of Mechanical, Aerospace and Civil Engineering
Undergraduate Dissertation Project

Development of Multi Fidelity Neural Networks for Optimisation of CFD Simulations

A dissertation submitted to The University of Manchester for the degree of
Bachelors of Engineering in Aerospace Engineering
in the Faculty of Science and Engineering

YEAR OF SUBMISSION
2023

Emrecan Serin
ID 10536545
Supervisor: Dr Alex Skillen

Department of Mechanical, Aerospace and Civil Engineering
2022-23

Abstract

State-of-the-art CFD simulations play a crucial role in exploring new design hypotheses in the world of engineering. However, they are computationally expensive and time consuming to continuously run to test every possible design hypothesis. As a solution to this problem, Neural Networks (NNs) can be utilised to form surrogate models that can approximate a simulation's results in a faster and more efficient manner. Although, large quantities of hard to obtain high fidelity (HF) training data is needed to form accurate NN surrogate models, negating the advantages of using a NN model in the first place.

As a solution, Multi Fidelity Neural Networks (MFNNs) can be introduced to decrease the reliance of NNs on HF data with the use of low fidelity (LF) data sets. Effectively forming a relationship between the two fidelities of data to form a multi fidelity surrogate model, which can incorporate advantages of both fidelities while negating their disadvantages. Hence, decreasing the dependency on expensive HF data by forming models that are capable of replacing inefficient and costly simulations.

The aim of this project is to optimise the application of existing and emerging MFNN models to further decrease their dependency on HF data, further improve their accuracy and generalisability, to drive down costs in the engineering industry and academic research. To achieve this, a 2-step MFNN was formed from scratch using PyTorch framework and was benchmarked with the use of complex numerical examples taken from literature. It was found that the multi fidelity surrogate model had performed better than singular NN surrogate models during validation testing of the model. Additionally, proving its performance enhancing capabilities by forming predictions at locations in the parameter space away from the HF training data, with the guidance of the LF points, to give more accurate and flexible predictions.

Keywords: Machine Learning, Neural Networks, Multi Fidelity, Surrogate Models ,PyTorch

Acknowledgements

I would like to thank my project supervisor Alex Skillen for his great feedback and assistance during the time I worked on this project, I strongly believe that I would not have been able to gain this amount of knowledge outside of my course's scope without his help. Also, I would like to thank my parents Oğuz and Nergüz for believing in me and working day and night without hesitation, for me to have a great education and a bright future. But most of all, I would like to thank Nati for staying strong through the hardest times and being there for me.

Declaration

I Emrecan Serin declare that the work and codes presented in this report are all my own work. All work that has been taken from other resources, papers and projects have been correctly referenced. If any work is found that has not been referenced, contact me immediately and proper action will be taken immediately.

List of Figures

List of Tables

Contents

Nomenclature

C_L	Coefficient of Lift
C_m	Coefficient of Moment
Adam	Adaptive Moment Estimation
AoA	Angle of Attack
BNN	Bayesian Neural Network
CFD	Computational Fluid Dynamics
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DNN	Deep Neural Network
GPR	Gaussian Process Regression
GPU	Graphics Processing Unit
HF	High Fidelity
LES	Large Eddy Simulation
LF	Low Fidelity
MF	Multi Fidelity
MFNN	Multi Fidelity Neural Network
ML	Machine Learning
MMNN	Multi Model Neural Network
MSE	Mean Squared Error
NACA	National Advisory Committee for Aeronautics
NN	Neural Network
PDE	Partial Differential Equation
PINN	Physics-Informed Neural Network
RANS	Reynolds-Averaged Navier–Stokes
ReLu	Rectified Linear Unit

1 Introduction

1.1 Background & motivation

In a typical engineering design process, it is crucial to test numerous design hypotheses in order to identify the most optimal design that maximises its performance. A great method to test these hypotheses without building a full scale model is to use detailed computer simulations and experiments that can provide reasonably accurate results about the design's real-world performance. As a result, detailed simulations and experiments which can produce accurate or high fidelity[1] (HF) results are preferred during the initial design phase. However, these accurate techniques require large amounts of computational power and time to produce results, limiting the number of design hypotheses that can be tested within a short time frame or budget. Concurrently, cost and time are critical factors in both industry and academia, where limited budgets and tight time frames are often encountered.

In the case of an aircraft's manufacturing process, the aerodynamic properties of aerofoils are the most sensitive and important aspect of the design, where a high quantity of different hypotheses must be tested to find the most optimal design. To test these hypotheses, a major method is the utilisation of Computational Fluid Dynamics (CFD) simulations, which are widely used in the engineering industry. They are governed by Navier-Stokes equations which are based on mathematical models that describe the behaviour of fluids using partial differential equations (PDEs) and they make it possible to assess an hypothesis's viability without the need of a real-world scale model, reducing the overall cost of the design stage (Vinuesa and Brunton, 2021). However, complex simulations are still too computationally demanding to be able to test every design hypothesis in high detail, which creates a bottleneck in the design process.

As a solution to this bottleneck, Neural Networks (NNs) can be used to enhance the performance of computer simulations and their results, due to NNs strong ability to learn the abstract relationships commonly encountered in computer simulations such as CFD simulations. It means that they can process the results from a simulation to improve its scalability, negating the need to run another simulation for a slightly different hypothesis. Thus, modifying the structure of the design process to be more efficient by decreasing the dependency on complex simulations, improving the process's cost and time efficiency.

NN models achieve this by capturing relations between a simulation's input and output values through the model training stage, where the data runs the model through its layers altering its structure to fit the data that is running through it. Therefore, enabling the

[1] Fidelity defines the accuracy of a simulation or data when compared to the real world.

model to predict an output for an input that was never introduced to the model before. This property of NN makes it possible to run familiar design hypotheses through a pre-trained model, negating the need of a new computational simulation to be ran to obtain the desired output.

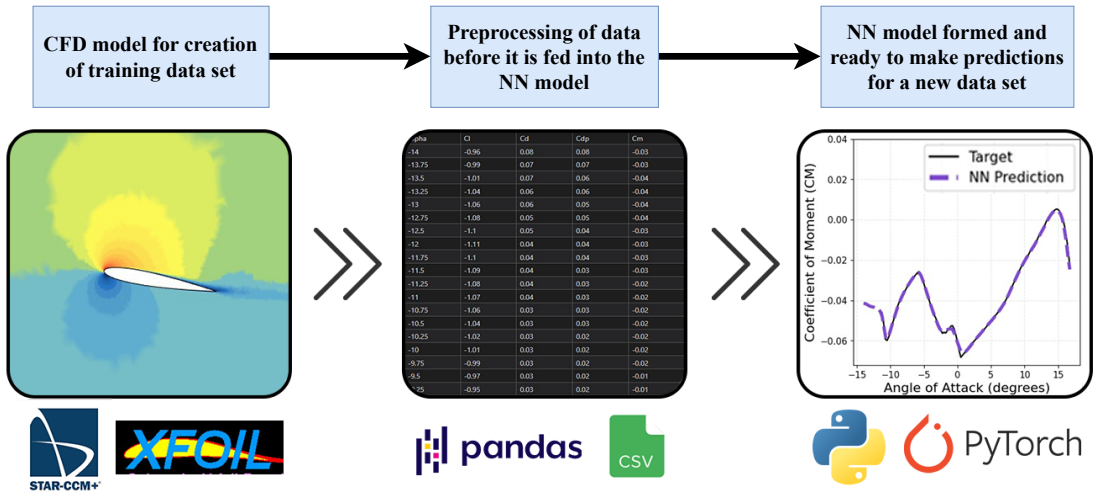


Figure 1: Training process of a Neural Network on the surface level

Figure 1 demonstrates the training process of a NN model on the surface level. The training process begins by feeding the training data obtained from a CFD simulation into the NN model. The model then gains the ability to generate predictions based on this data, allowing it to make predictions beyond the scope of the training data set and improving the scalability of the results. Moreover, the performance of a NN model is largely dependent on the quantity and fidelity of the training dataset. Meaning that, to form an accurate and effective model, a large quantity and high fidelity data is needed.

As previously mentioned, to train an accurate NN model, large quantity of HF data obtained from detailed simulations (e.g. LES) are needed. Although they are costly and time demanding to obtain due to the simulation's computational demands. Hence, negatively impacting the effectiveness and scalability of the model with the lack of data points that can be produced and negate the advantages of using a NN model in the first place. On the other hand, less accurate low fidelity (LF) data in higher quantity obtained from less accurate simulations (e.g. RANS) can be used to train the model, increasing its scalability while decreasing its accuracy. As a result, it is common practice for engineers to try and find a middle ground between accuracy and scalability while picking data set to train a model (Chen, Gao, and Liu, 2022).

To resolve this problem, Multi Fidelity Neural Networks (MFNNs) can be used in

implementing a bridge between these fidelities to create a multi fidelity surrogate model. Obtaining an accurate and a computationally light network by combining the advantages of multiple surrogate models. As a result, lowering the dependency on high fidelity data with minimal impact to the simulations accuracy and performance (Mole, Skillen, and Revell, 2020).

1.2 Aim & objectives

The aim of this project is to optimise the application of existing and emerging NN models by introducing MFNNs, to decrease NN models dependency on high fidelity data, improve their precision, speed and scalability, to drive down costs in the engineering industry and academic research.

The objectives enumerated below have been determined and completed to satisfy the aim:

1. Form and analyse the structure of Neural Networks with the help of existing documentation and academic papers.

2. Form Multi Fidelity Neural Networks and evaluate their efficiency compared to basic Neural Networks with the use of synthetic data sets.

3. Optimise Multi Fidelity Neural Networks to decrease their dependency on high fidelity data produced by complex simulations.

4. Improve the flexibility of Multi Fidelity Neural Networks to effectively handle the phase-shift between different data sets.

5. Illustrate the network's real-world engineering practice uses by applying the model on available aerofoil data available from the literature and organisations, proving their cost reducing capabilities.

Furthermore, the methods that have been used to satisfy the objectives are enumerated respectively below. Further detail will be provided in Section 4.

1. Form a multi-layer feed forward Neural Network with the use of Python and PyTorch Machine Learning Framework.

2. Perform tests on singular Neural Networks and Multi Fidelity Neural Networks using synthetic data sets consisting of complicated numerical examples to benchmark their performance.

3. Gather feedback from the performance tests and perform optimisation on the Multi Fidelity Neural Network's structure to improve its performance.

4. Use existing documentation and academic papers to improve network's phase-shift handling capabilities by making changes on its structure.

5. Collect pre-existing NACA[2] aerofoil simulation results from a reliable open-sourced database to test the model on real-world engineering applications.

1.3 Research questions

This section goes over the research questions asked throughout the time period where the student has worked on the project, with their respective answers referred within text.

- How can effective NNs be formed and optimised using PyTorch framework, to improve their accuracy and effectiveness? [Objective 1 & 2 and Section 4.5]
- How can the existing literature on NN and MFNN modelling and benchmarking techniques be used to develop and compare MFNNs performance to NNs? [Objective 3 and Section 5.5]
- How can MFNNs be optimized to reduce their reliance on high-fidelity data and reduce the need of running computationally demanding simulations? [Objective 3 and Section 4.7]
- What techniques can be utilised to improve the phase-shift handling capabilities of MFNN? [Objective 4 and Section 4.9]
- How can Multi Fidelity Neural Network models be enhanced and adapted to handle computer simulations to solve existing problems in engineering? [Objective 5 and Section 5.6]

1.4 Relevance and impact

1.4.1 Industrial and academic impact

As already mentioned in Section 1.1 processing power availability for CFD simulations is the main limiting factor for their use. To accommodate high numbers of simulations to be ran in a limited time frame, sacrifices in level of detail and realism are commonly made by applying assumptions into the simulations. Implying that CFD simulations are hard to replicate in larger amounts, as they require large amounts of power and time. This is one of the main reasons why NNs and specially MFNNs are starting to become a trend in today's scientific an industrial branches. Since, MFNNs make it possible for the simulations to run for a shorter period to produce the few required HF points to be aided by easily accessible LF points. Thus, decreasing the amount of energy and time used for the simulation and increasing efficiency.

Additionally, Moore's Law states that the number of transistors on a processor would double every two years. Thus reducing the size of the chip while increasing the

[2]Aerofoil shapes for aircraft wings developed by the National Advisory Committee for Aeronautics (NACA).

transistor count which improves the integrated circuit's functionality and performance while decreasing costs (Waldrop, 2016). With this piece of information, after a certain amount of years the most challenging CFD simulations could run effortlessly in a short period of time. Although, in reality the growth is not that straight forward and according to Moore's Law there's a long way for processors to reach that level.

Furthermore, cost and time in branches of industry and research are the two most important things, applying a low cost and an efficient model to simulations that can be run for a shorter duration significantly increases the yield of results. Making MFNNs more attractive for any engineering discipline or industrial application.

1.4.2 Sustainability impact

The adoption of MFNNs in industrial research and development has the potential to significantly impact sustainability in the industry which recently became a major point of interest. The use of MFNNs can allow for more efficient and streamlined simulations to be performed by reducing the dependency on HF data and reducing total simulation time. Resulting in noticeable energy savings during the design stage and reduce environmental impact associated with simulations that require a large amount of computational power and energy.

2 Background Theory

2.1 Chapter Overview

This section provides an overview of Neural Networks, covering their structures, training process and working principles. By providing the necessary background for the reader to better grasp the Literature Review section of this report, and build up to forming Neural Networks and Multi Fidelity Neural Networks.

2.2 Fundamentals of Neural Networks

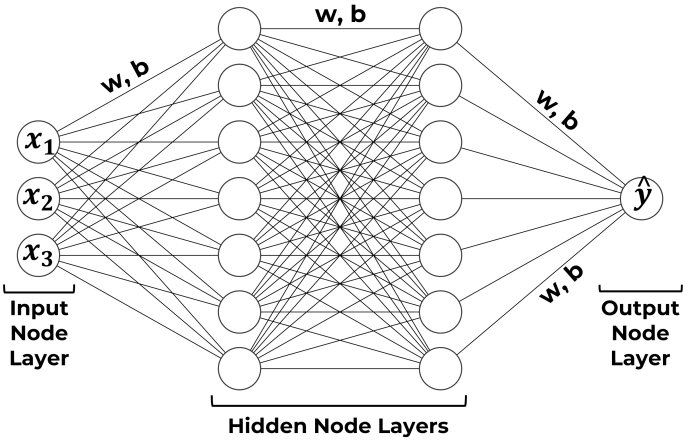


Figure 2: Deep Neural Network layers interconnected by weights and biases

Neural Networks are a collection of functions composed of neurons (nodes) interconnected with parameters (weights, $\mathbf{w}$ and biases, $\mathbf{b}$) to each other. A typical NN consists of at least three layers which are input, middle and output layers. Referring to Figure 2 the first layer is the input layer where data (x) is fed into the network and the last layer is the output layer that produces the final prediction ($\hat{y}$) of the network. In addition, the layer(s) residing between the input and output layers are refereed to as hidden (middle) layers (Ding, Su, and Yu, 2011). Typically, NNs that contain more than a single hidden layer are called Deep Neural Networks (DNNs).

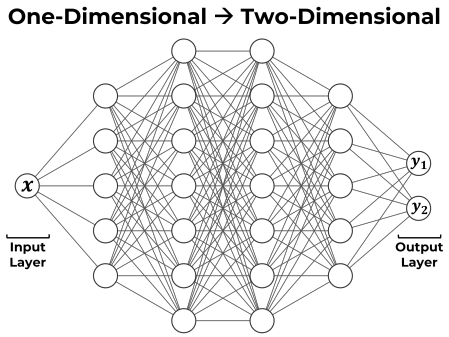
2.2.1 Nodes & layers

A NN's structure can be customised to include any numbers of layers and nodes inside it and altering these numbers can play a significant role in the network's learning capabilities and performance. Consequently, increasing the number of layers and nodes can result in better predictions with higher accuracy as the network can better identify smaller details and patterns in the data (Cilimkovic, 2010). Although, it is essential to experiment with

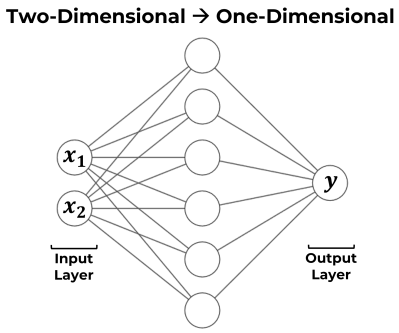
and adjust the number of neurons and hidden layers for each specific problem to optimise the network's computational performance and to prevent overfitting or underfitting in the model's predictions (Lecun et al., 1998). These limitations will be further discussed under 'Generalisation' in Section 2.3.7.

2.2.2 Multi-Dimensionality

A notable property of NNs is their ability to process multi-dimensional data sets[3]. This property makes NNs a versatile tool for analysing complex data sets, as the number of nodes at the input and output layers can be altered to accommodate any dimension (Altun, Bilgil, and Fidan, 2007).



(a) Deep Neural Network with a single input and two output nodes (1D to 2D)



(b) Basic Neural Network with two input and one output node (2D to 1D)

Figure 3: Two different structures of Neural Networks

Figure 3 illustrates two NN structures with different layer configurations as an example. Whereas, Figure 3a illustrates a DNN capable of generating a two-dimensional output from a one-dimensional input, and the network in Figure 3b illustrates a network capable of generating a one-dimensional output from a two-dimensional input. Representing that NNs can alter the dimensions of data that is running through them which is an important property that enables the development of Multi Fidelity Neural Networks.

Moreover, it is important to emphasise that the configuration of the hidden layers do not influence the dimensions of input and output layers. Implying that any configuration of hidden layers can be used without affecting the dimensions of the input and output data (Altun, Bilgil, and Fidan, 2007).

[3]e.g. data made of multiple variables, such as temperature, velocity and pressure at each point in a time domain.

2.3 Training Overview

Training a NN is an iterative process that involves adjusting the network's weights and biases to minimize the error between the predicted output of the model ($\hat{y}$) and target output (y) fed into the model. This iterative process consists of several key steps including, forward pass, loss calculation and backward pass to update the parameters (LeCun, Bengio, and Hinton, 2015).

Sections (2.3.1) to (2.3.6) will be covering the training process of a NN in detail, whereas an overview of the training process is provided in the flowchart in Figure (4).

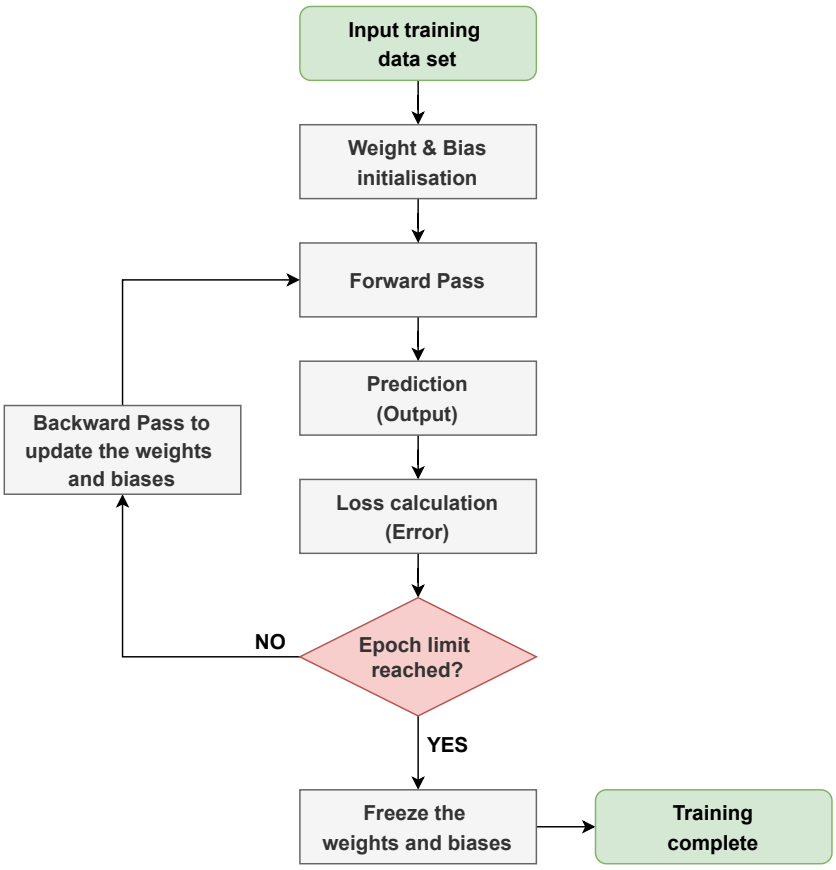


Figure 4: Neural Network training overview flowchart

2.3.1 Weight and Bias initialisation

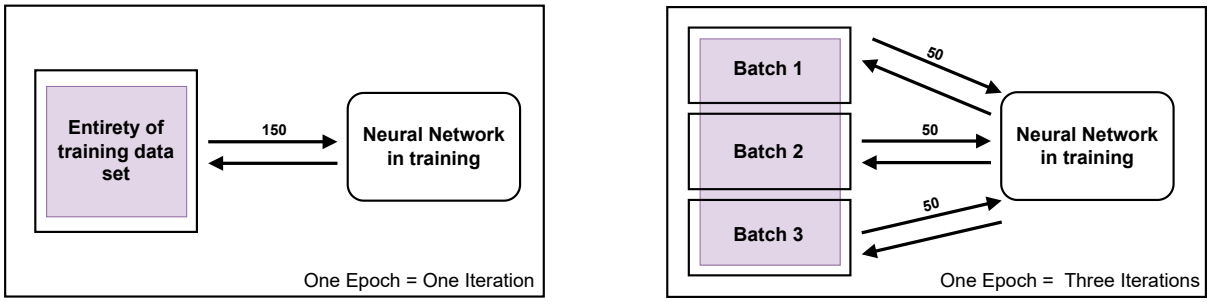
Before the training begins, weights and biases are initialised with small random numerical values. This approach helps breaks the symmetry in the hidden layers of the network, allowing for a more computationally cheaper and flexible model capable of learning more features in the data (Goodfellow, Yoshua Bengio, and Courville, 2017).

2.3.2 Epochs and Batches

Epochs are one of the various hyperparameters used in configuring NNs. They determine how many times the entire training data set will be passed through the network. Allowing the model to adjust its parameters with every pass and improve its prediction capabilities.

Moreover, a single epoch is only considered complete when the **whole** data set passes through the network **once** and it is common for large numbers of epochs to be used during training which can range from hundreds to thousands. This ensures that the network can effectively learn the patters and features lying within the data. However, just like any hyperparameter it is important to properly adjust them to prevent overfitting or underfitting in the data.

Figure 5 illustrates two examples of NN training processes with and without batches, with a single pair of arrows representing a single iteration.



(a) Training without batches

(b) Training with three batches

Figure 5: Effects of batches on number of iterations per epoch

Batch size is a similar hyperparameter to epochs, which determines how the training data is going to be divided into smaller subsets during training. Typically, a training dataset can be divided into one or more batches determined by the batch size hyperparameter.

As an example, referring to Figure 5, choosing a batch size of 50 out of the 150 available training samples, will mean that the network will pick 50 samples on each iteration, until three iterations are complete to finalise an epoch.

Important: An iteration and an epoch take different meanings when batches are introduced into the system. In this context, an iteration refers to a single batch or subset of data running through the network, whereas an epoch (which may consist of multiple batches) represents the point when a complete pass of the **entire** dataset is made (Brownlee, 2018).

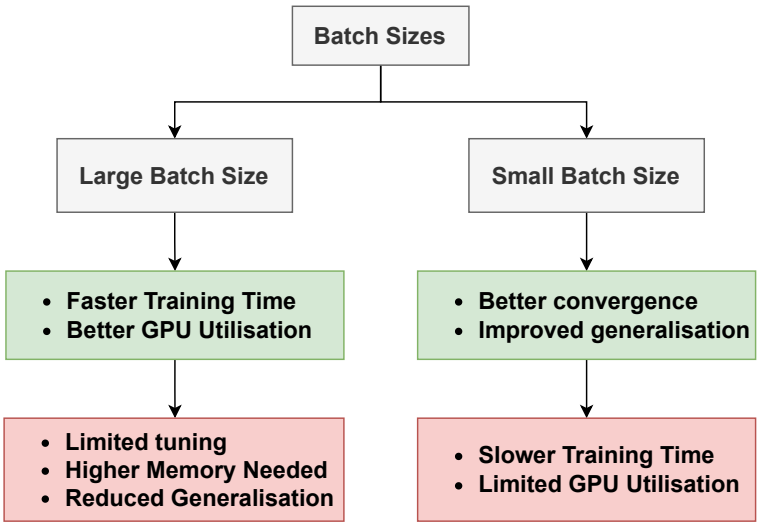


Figure 6: Advantages and Disadvantages of Utilising Different Batch Sizes

Different batch sizes may produce different results and the choice of size typically depends on the structure of the network, complexity of the data set and the available hardware resources. Additionally, the use of various batch sizes contribute greatly in determining the speed and learning capabilities of the network. Moreover, similar to any other hyperparameter, an optimal batch size is often determined through experimentation and fine-tuning and requires trade-offs to be made on the performance of the network, as illustrated in Figure 6.

2.3.3 Forward Pass and Loss Functions

Firstly, during forward pass, input data (x) divides into chunks and sequentially travels through every neuron in the network. Where, each neuron modifies the chunk of data passing through it using its corresponding parameters and activation functions. Then, this process continues from input to output layer until the final prediction is produced by the output layer(s) (LeCun, Bengio, and Hinton, 2015). Figure 7 illustrates the flow of data during the forward pass.

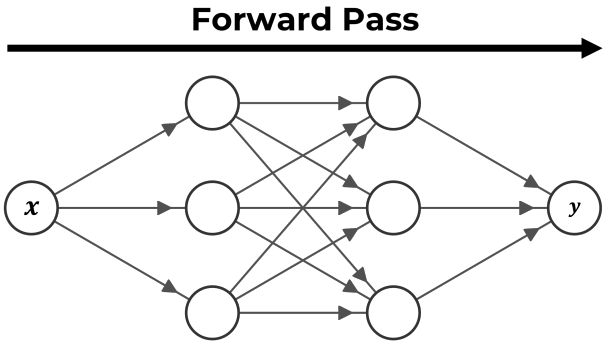


Figure 7: Forward pass in a Neural Network with the flow of data represented by arrows

Secondly, to train an effective NN, the loss (error) between the predicted output ($\hat{y}$) and the actual target output (y) which is fed into the network must be minimised. To achieve this, a loss function which is a mathematical function can be used to calculate their difference.

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

where n is the number of samples, y_i is the target output for the i-th sample, and $\hat{y}_i$ is the predicted output for the i-th sample.

Mean Squared Error (MSE) loss function, shown in Equation 1 is a common example used for regression problems. It determines the loss by calculating the average squared distance between the predicted and the target output (Haykin, 1998).

2.3.4 Backward Pass (Backpropagation)

Following forward pass and loss calculation, the backward pass begins at the output layer and works its way through the network to the input layer. As it passes through the network, it calculates the 'gradient' of the calculated loss concerning each parameter of the network. Then, the optimiser uses these gradients to update the values of the weights and biases of every neuron to minimise the loss on the next forward pass. As a result, this creates an iterative system where the forward pass computes the loss, and the backward pass calculates the gradients for the optimiser to make changes on the parameters, reducing the loss (Hecht-Nielsen, 1992).

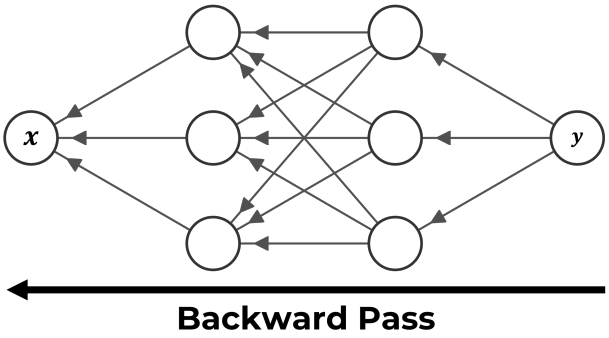


Figure 8: Backward pass updating weights and biases in a Neural Network

Firstly, gradient represents the rate of change of the loss function with respect to the parameters, informing the optimiser in which direction the parameters should be updated to minimise the error on the next iteration (Karpathy, Johnson, and Fei-Fei, 2016).

Secondly, optimisers are algorithms capable of updating the parameters of a NN based

on the information obtained from the gradient computed during backpropagation. With each iteration, they update the weights and biases using preset hyperparameters such as 'learning rate' to minimise the loss function during training (Ruder, 2016).

Furthermore, there are various optimiser algorithms capable of minimising the loss in different fashions. Some of the most common used algorithms for regression networks are Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) algorithms. Additionally, these iterative optimisation algorithms use first-order differentials to find local minimums of differentiable functions using the chain rule (Burkov, 2019).

2.3.5 Activation Functions

Activation functions are mathematical functions applied to the ends of **each** neuron in the network to introduce non-linearity into the model, enabling the network to learn more complex patterns and relationships in the data. They are fundamental parts of NNs allowing them to function and compute predictions (Feng and Lu, 2019).

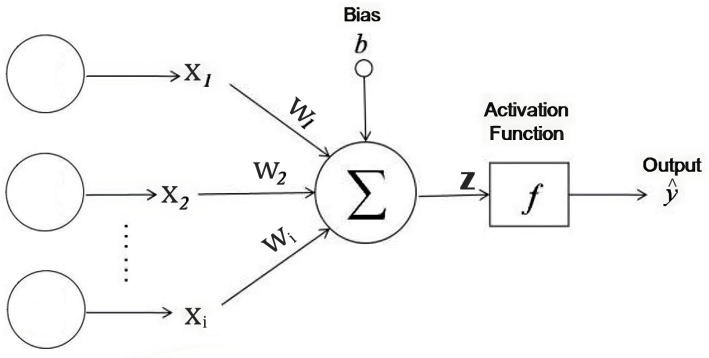


Figure 9: Close up of a single neuron's structure (Grigoryan, 2021)

$$z = \sum_i w_i \cdot x_i + b \tag{2}$$

$$\hat{y} = f(z) \tag{3}$$

where z is the sum of the product; w_i represents the weights; x_i is the input(s) from the previous neurons; b is the bias applied; $\hat{y}$ is the output prediction of the NN; and f is the activation function applied to z respectively.

As illustrated in Figure 9, activation functions are applied at the end of each neuron, following the calculations of weights and biases. By referring to Equations 2 & 3, initially, the weights are summed and a bias is added to the input (x_i) to yield output Z. Subsequently, Z is processed by an activation function f to generate the output $\hat{y}$ of the

neuron. This output is then fed to the next neurons as an input again. Additionally, this process takes place at every single neuron throughout the network.

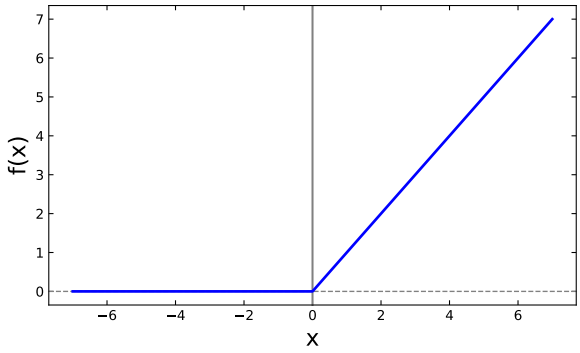


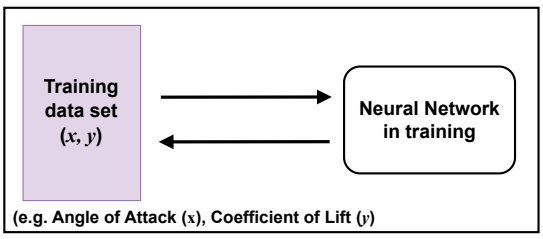
Figure 10: ReLu activation function

$$f(x) = \max(0, x) \tag{4}$$

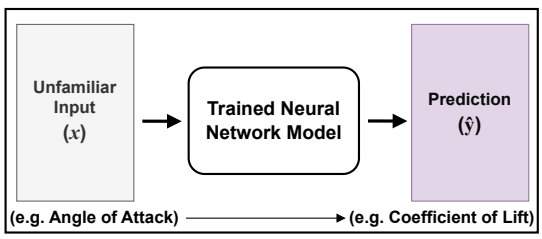
A great activation function example is the Rectified Linear Unit (ReLU) function, that transforms the input by setting all negative values to zero while keeping positive values unchanged, illustrated in Figure 10 and Equation 4, which is one of the many available activation functions that can be used to introduce non-linearity to or scale the data flowing through the network. Moreover, depending on the type of problem, functions such as sigmoid and Hyperbolic Tangent (tanh) functions can also be used with each having their unique properties and uses (Feng and Lu, 2019).

2.3.6 Freezing weights and biases

As the final step, as discussed in Section 2.3.3, the network training process ends when the loss is converged, or when the pre-determined number of epochs is reached by the network. As a result, the NN becomes ready to be used for making predictions and ultimately turn into a surrogate model.



(a) Network in training



(b) Network predicting

Figure 11: Training and Predicting Phases

2.3.7 Generalisation

Generalisation is the model's capability to adapt to and provide accurate predictions from previously unseen, new input data. In other words, referring to Figures 11a and 11b it represents the network's abilities to apply the knowledge it obtained during training (11a) to new data sets to produce outputs during prediction stage (11b).

Moreover, generalisation is an important property of NNs that shall not be overlooked, as it impacts the models prediction flexibility and abilities and an improved generalisation ultimately means that the model can be more flexible and be better at predicting unseen data. Finally, generalisation can be improved by avoiding overfitting and underfitting during training as previously mentioned in Section 2.2.1 (Goodfellow, Yoshua Bengio, and Courville, 2017).

2.3.8 Training, Testing and Validation data sets

To train an effective Neural Network model that can perform well, it is essential to divide the input data set into bits that will be used in training, setting the correct hyperparameters and lastly testing the performance of the NN before the training is complete.

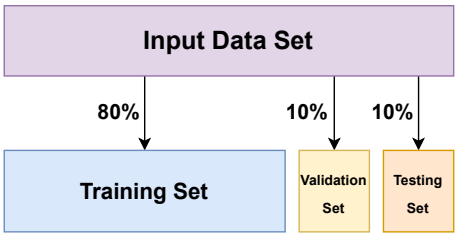


Figure 12: Training, testing and validation split of the input data set

In a typical NN training process, the input data set is split into three major sets. Firstly, the training set which usually consists of the 80% of the data set that will be used to train the model. Secondly, the validation set (10% of total data) to set the hyperparameters of the trained model and evaluate its performance. Lastly, the testing set (10% of total data) where the model's final performance such as is generalisation is tested. The percentage of the set sizes can change from problem to problem to find the best for a given problem (Burkov, 2019).

2.4 Summary

To summarise this section, Neural Networks are Machine Learning models capable of serving as surrogate models, providing a cost effective alternative to complex computer

simulations.

They are capable of handling multi-dimensional data and they come in any shape or form with varying hyperparameters to suit any given problem during training, ultimately making them capable of making predictions as surrogate models.

3 Literature Review

3.1 Chapter overview

Building up from the Background Theory section, the literature review section goes over the uses of singular NN surrogate models and demonstrates their capabilities in the engineering industry and academia. Also, while highlighting the points where they lack in performance and reliability. Additionally, illustrating how challenging it is to strike the right balance between the hyperparameters to optimise a NN designed for a specific purpose.

Then, it introduces fidelity in modelling & simulation illustrating why HF data is harder to obtain compared to LF data from experiments and simulations. Moreover, it introduces different types and structures of MFNNs, demonstrating their capabilities by giving examples from literature with synthetic mathematical data sets, and real-world uses with LES and RANS simulations. Hence, highlighting their ability to balance accuracy and efficiency while minimising their dependency on expensive HF data.

Lastly, the section goes over possible improvement methods such as using embedding theory and MF data aggregation using CNNs to form stronger relationships between the LF and HF data to achieve better prediction accuracy and overall MFNN model performance.

3.2 Fidelity in modelling & simulation

In modelling & simulation, fidelity defines the accuracy of a simulation or data when compared to the real world. Where, high fidelity (HF) data represents results that are closer to real-world results from experimental tests, and low fidelity (LF) represents results that are not as closely aligned with real-world experiments and less accurate compared to HF data.

Table 1: HF against LF Data Pools with advantages highlighted in green

	High Fidelity Data	**Low Fidelity Data**
Data Availability		
Accuracy		
Cost		
Prediction Range		

As mentioned earlier in Section 1, to train an accurate NN model, large quantity of HF data from detailed simulations such Large Eddy Simulations (LES) or experimental data from real-world experiments must be collected. Naturally, resulting in more

computationally and time demanding process to collect HF data, to form an accurate surrogate model. Alternatively, LF data can be used to train a less accurate NN model with a higher quantity of data points, which can increase the prediction range of the model by running more hypotheses through it, but by resulting in a lower accuracy. Additionally, a trade-off between HF and LF data pools are shown in Table 1 for the reader to make a clearer trade-off between the two fidelities of data.

Lastly, it is important to note that multiple fidelities of data must originate from the same problem that is being aimed to be solved. As an example, LES and RANS must be performed on the same aerofoil geometry to combine the two fidelities together, in order to form a better relationship between them.

3.3 Surrogate models

Surrogate models are simplified representations of complex simulations designed to approximately mimic the behaviour of the simulation it was trained on, aiming to reduce the computational costs and time spent on a design's analysis by replacing the complex models.

They can take various forms and shapes such as, Gaussian Process Regression, Polynomial Regression, Kriging, Radial Basis Functions, and NNs, with each model having its advantages and disadvantages per a type of problem. Among these options, NN surrogate models are widely preferred due to their scalability, adaptability and faster computing potential using parallel processing (Peter and Marcelet, 2008).

In this report's context, the characteristics of both singular and multi fidelity surrogate models will be reviewed in detail to illustrate their differences in performance.

3.4 Uses of Singular NN surrogate Models

NNs can be used to post-process results obtained from CFD simulations to later be used as surrogate models as an alternative to running new simulations. Hence, in addition to obtaining a more efficient model, this introduces many advantages, such as, expanding the prediction range of the model compared to the range of the original data set (scalability) and making use of historical data a viable option to eliminate the need of running another simulation for a slightly new design hypothesis.

3.4.1 Case 1: Using Historical data sets

NNs can be trained on pre-existing data sets, containing tens or hundreds of pre-ran simulation results of familiar simulations. Making it possible to form a surrogate model

capable of making an accurate prediction on a design that is newly introduced to the model by using the pre-existing data. Hence, eliminating the need to run another simulation, saving valuable simulation time and cost.

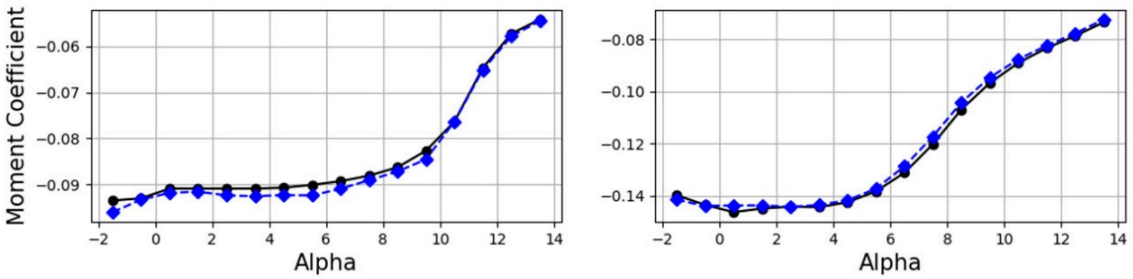


Figure 13: Predicted values against the actual Moment Coefficient over AoA of two different aerofoils (Bakar et al., 2022)

Figure (13) illustrates an example taken from literature. In this study, a Convolutional Neural Network (CNN)[4] was used to analyse 2300 aerofoil geometries fed into the network along with their respective aerodynamics parameters. After training the model, random aerofoils had been picked from the validation data set to validate the model's accuracy. As it can be seen, predictions (dashed blue lines) in comparison to the target values (black lines) are good in agreement with each other per aerofoil. Hence, indicating that the network had been successful in predicting the desired aerodynamic parameters within an acceptable degree of accuracy (Bakar et al., 2022).

3.4.2 Case 2: Improving prediction range and scalability

NN surrogate models, given the right data set can produce accurate results without deviating from the target at most of the cases depending on the problem, although, compared to computer simulations, they are fragile models that are more prone to failing and giving inaccurate results highly depending on their quality and depth of training and structure.

As an example, a NN can be trained using the extracted data from an aerofoil simulation to learn the relationship between the aerodynamic parameters with the changing Angle of Attack (AoA). By doing so, a surrogate model can be formed that is capable of making predictions on a wider range of AoAs beyond the initially taught data (e.g., predicting C_L at AoAs from 0 to 20 degrees while the training data set only ranges from 0 to 8 degrees). As a result, this increases the prediction range of the model, making it a more flexible tool against conventional computer simulations. This is a big advantage, as they

[4]A class of DNNs that functions differently, but can be used to achieve similar goals.

can provide predictions for points outside the range of the training set without the need of any additional data.

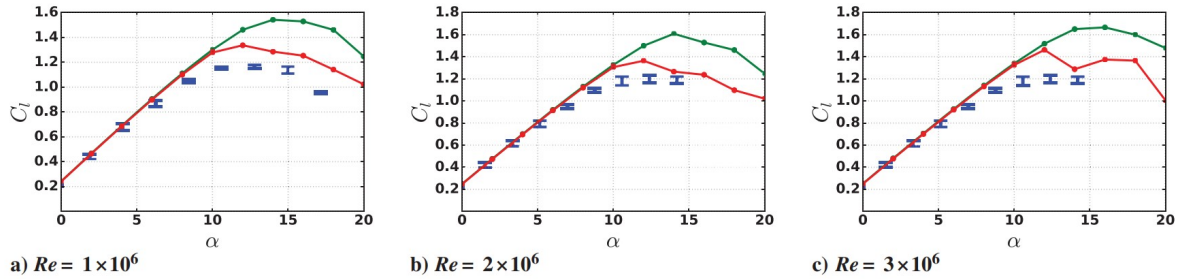


Figure 14: Comparison of experimental, simulation and NN model predictions of an aerofoil at three different Reynolds numbers (Singh, Medida, and Duraisamy, 2017)

Figure (14) demonstrates the Coefficient of Lift (C_L) over AoA for an S809 aerofoil tested at three different Reynolds numbers. The figure displays real-world experimental values (blue), simulation results (green), and NN predictions (red) which were made by an NN model trained using the experimental data. The simulation result were obtained from Spalart–Allmaras Turbulence Model, and the NN's structure was sourced from the Fast Artificial Neural Network Library (FANN) in ANSI C by Steffan Nilsen (Singh, Medida, and Duraisamy, 2017).

As can be seen from Figure 14a, the NN model trained using the experimental data initially performs well, matching the target (experiment data) until 8 degrees of AoA. As the AoA increases the NN model starts overshooting its target, although still giving a better prediction than the simulation. In addition, a similar result can be observed in Figure 14b, where the model outperforms the simulation at AoAs outside of the training range (after 14 degrees). However, in Figure 14c, when the Reynolds number is increased again, the NN model fails and fails to provide accurate predictions beyond 12 degrees of AoA and causes a significant change of C_L by the end of the range. Lastly, it must be noted that, despite the NN model's limitation, it still outperforms the simulation results.

As a conclusion, the study indicates that NN surrogate models can be used as an alternative for computer simulations, although they might require a large amount of HF data to produce a reliable and an accurate model. Hence, indicating the need of more HF training data which is expensive to obtain or a more efficient method that can utilise both HF and LF data sets.

3.4.3 Limitations of singular NNs

NNs have shown that they can provide more advantages in few scenarios compared to computer simulations either by making use of pre-existing data sets or increasing the range and scalability of the experiment that it being used on. Although, they still do have significant drawbacks such as not being reliable and accurate enough in comparison to complex simulations. It is extremely important to point out that a NN model is as good as its training data.

A great example to cover the limitations of NN models would be to refer to Case 1, to point to their prediction capabilities in terms of accuracy and reliability. As it can be seen from Figure 13 the NNs' results are not perfectly accurate, showing slight inaccuracies in its prediction of Moment Coefficient between the AoAs of 2 and 8 degrees. This flaw can depend on several factors including the structure of the NN or the quality of the training data.

In addition to this, the flaws encountered on Case 2 can also be mentioned under this section. As it can be seen from Figure 14c, between the AoAs of 12 and 20 degrees, where no training data exists. The model starts to produce predictions of Coefficient of Lift (C_L) with abrupt and unrealistic changes, indicating that the model is more likely to fail at points with no training data.

Therefore, an alternative method can be introduced to address the challenges associated with singular NN surrogate models not being accurate and reliable enough. Hence, Multi Fidelity Neural Networks can be introduced to make NN based surrogate models a more viable option in industry and academia by reducing their dependency on large quantities of expensive HF data (Objective 3) and their flaws in making predictions.

To conclude this sub-section, singular NN models can be a great tool in the field of engineering to decrease the demand of running a high number of simulations. Although, they are not effective enough themselves to negate the dependency on complex computer simulations yet. Hence, the use of MFNNs must be introduced to offer a solution to these problems to open up the path to use Machine Learning in engineering industry and academia.

3.5 Multi Fidelity Neural Networks

Multi Fidelity Neural Networks function similarly to singular NN models. Although, instead of running a single fidelity of data, they run minimum of two data sets with varying fidelities to form a relationship between them. Forming this relationship, makes it possible for the model to combine the information from all fidelities of data sets, and enable each fidelity level to contribute to the overall understanding of the problem that needs to be solved. Thus, creating a more accurate and flexible surrogate model, rather than using either of the fidelities alone by themselves (Meng and Karniadakis, 2020).

3.5.1 Types of Multi Fidelity Neural Networks

Multi Fidelity Neural Networks typically consist of minimum of two singular NN models that are interconnected. Although, they can come in a variety of configurations, with each method having its own benefits. Offering the user a greater flexibility in selecting the most suitable model for a specific problem.

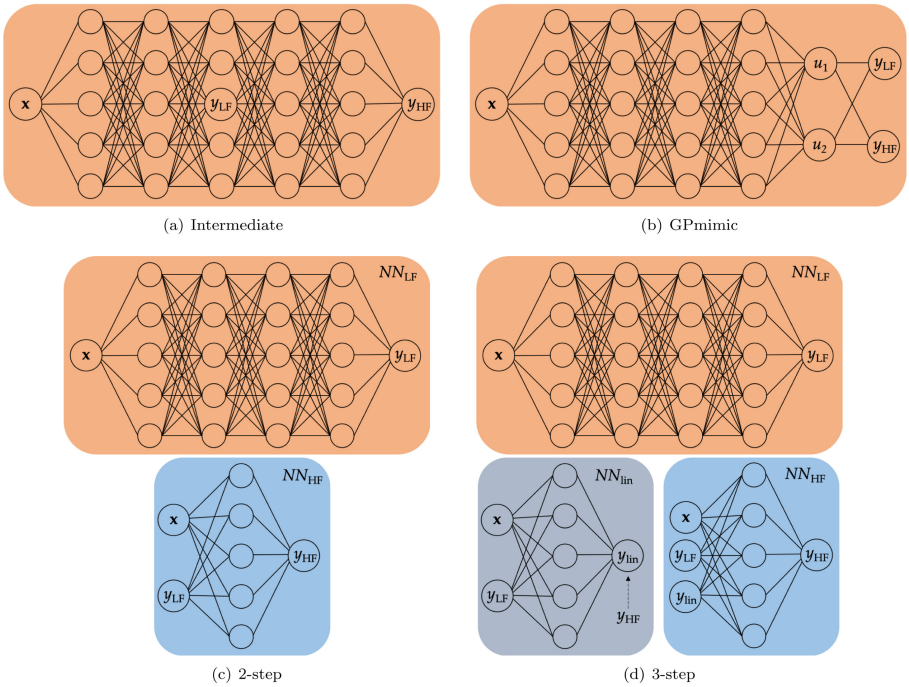


Figure 15: Different types of MFNNs (Meng and Karniadakis, 2020)

As it can be seen from Figure 15, MFNNs can take various forms and configurations such as, (a) intermediate, (b) linear model of coregionalization or (c) 2-step and (d) 3-step networks with each having their own benefits depending on the problem that is desired to be solved (Meng and Karniadakis, 2020). This report will be focusing on 2-step models to explain MFNNs.

3.5.2 2-Step Multi Fidelity Neural Networks

A 2-step MFNN uses two separate NNs to process the two fidelities of data sets (LF and HF) running through them. The MFNN is formed by running these data through each model in a specific order to map the relationship between them. As a result, enabling the MFNN to understand the relationship between the two fidelities of data, to generate a MF prediction accordingly.

Additionally, before a MFNN can any make predictions, the individual NNs inside the model must have already been completely trained (e.g., a LF Model (F_L) in a MFNN must be completely trained before the MFNN can be used to make predictions). The reader can refer back to Section 2.3.6 for further clarification on this concept.

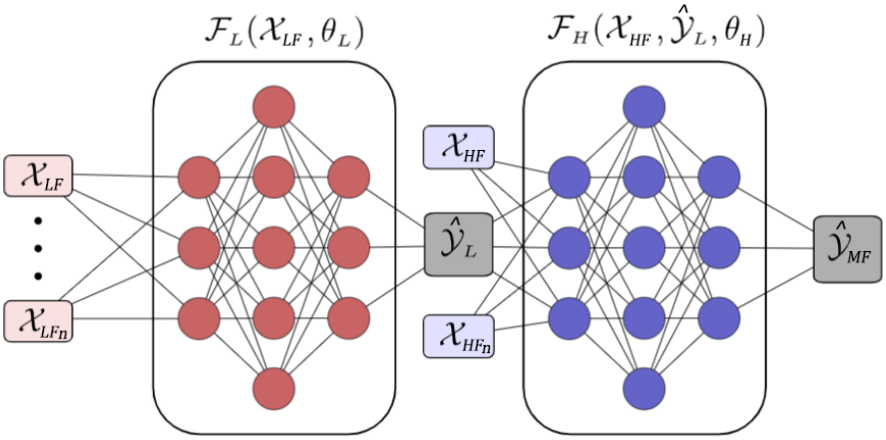


Figure 16: A MFNN consisting of two Neural Networks (Mole, Skillen, and Revell, 2020)

Figure 16 illustrates a MFNN model consisting of two interconnected LF and HF networks. With the colour red representing the LFNN model (F_L) and its LF input (X_{LF}). Whereas, the colour blue denotes the HFNN model (F_{HF}) and its HF input (X_{HF}). Furthermore, ($\hat{y}_L$) and ($\hat{y}_{MF}$) represents the predictions of LFNN and HFNN/MFNN models respectively.

Additionally, θ_L and θ_H represents the weights and biases of the LF and HF model respectively (Mole, Skillen, and Revell, 2020). As discussed in Section 2.3.4, this implies that when either X_{LF} or X_{HF} passes through the networks, they will be transformed according to the respective weights and biases, θ of the model. For instance, passing X_{LF} through θ_H will help the model form a relationship between LF data and the HF model.

In Figure 17 below, the MFNN model above has been illustrated in the form of a flowchart, for the reader's better understanding of the model.

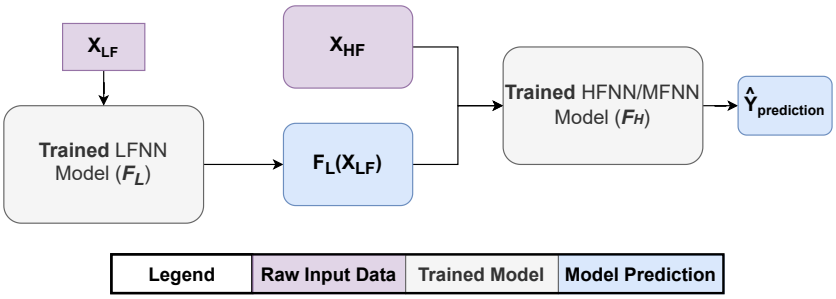


Figure 17: MFNN structure flowchart

Firstly, the MFNN prediction process starts when LFNN forms a LF prediction ($F_{L(X_{LF})}$) from the LF input (X_{LF}) passing through it. Secondly, the LF prediction is then combined with the HF input (X_{HF}) for the model to form a relation between the two fidelities of data. Finally, the combined input is then passed through the HFNN/MFNN network to form the final Multi Fidelity prediction ($\hat{y}_{prediction}$). Thus, forming a relation between the, the input values and predictions of each network to form the MFNN model.

3.5.3 Multi Fidelity Neural Network Surrogate models

Multi Fidelity Neural Networks have the ability to effectively combine multiple NN surrogate models to form a MF model with improved the accuracy and range. Hence, enabling the model to achieve a balance between accuracy and efficiency while minimising the model's dependency on expensive HF data to drive down costs (Mole, Skillen, and Revell, 2020).

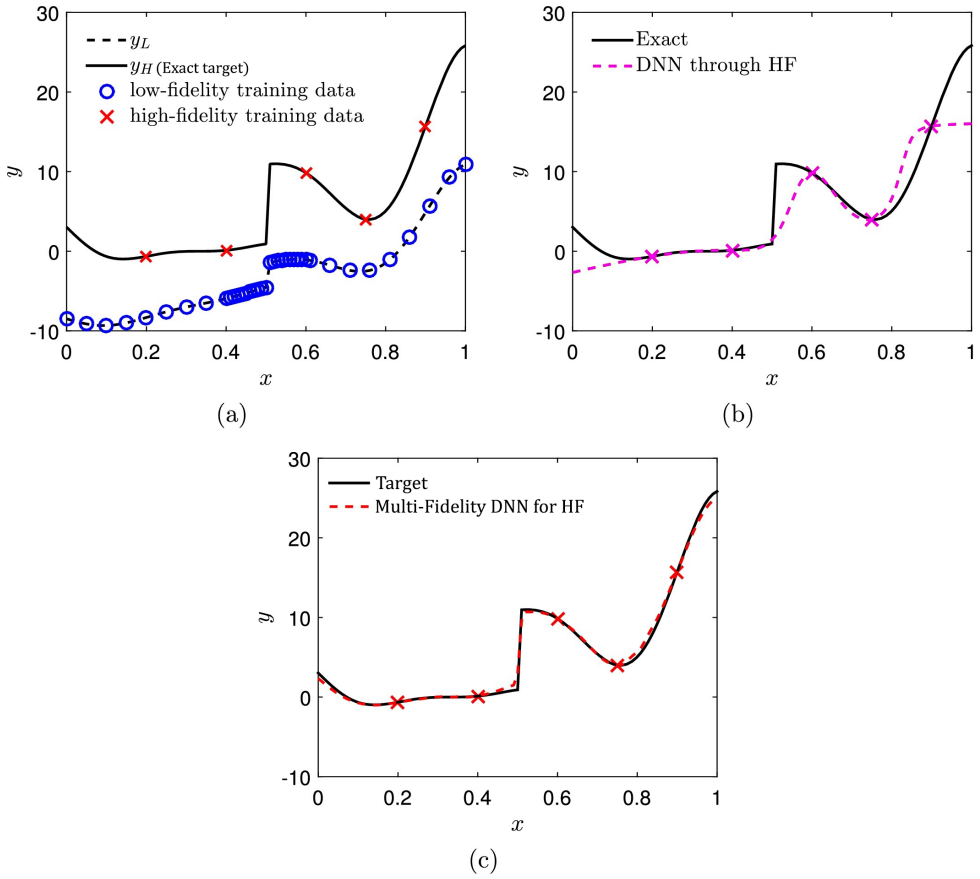


Figure 18: MFNN prediction of a discontinuous function (Meng and Karniadakis, 2020)

Figure 18 illustrates, the LFNN, HFNN and MFNN surrogate model's predictions from Figure 18a to 18c respectively. The aim of these networks is to predict the discontinuous function, Y_H Exact Target (black line) with a high accuracy, the red crosses represents the accurate HF data points situated right on top of the target, whereas the blue dots represents the inaccurate LF data points, situated around the target that form a similar shape to the Y_H Exact Target. Additionally, the dashed blue line represents the LFNN prediction, the magenta represents the HFNN prediction and the red line represents the MFNN prediction from Figure 18a to 18c respectively. Furthermore, there are 38 LF and 5 HF data points used as to train the for LFNN and HFNNs respectively.

As can be seen from Figure 18b, the HFNN used to predict the target significantly loses its accuracy and significantly fails at the beginning and the end of the line, as there are not enough data points for it to form an accurate prediction. Additionally, as the discontinuous function has a sudden change of y-values from $0.4 < x < 0.6$, the singular HFNN model without enough data points significantly fails again to predict the Exact Target.

Subsequently, the LFNN with its 38 inaccurate but correctly organised data points can

be used to guide the HFNN's accurate 5 data points. Thus, a MFNNs can overcome this issue with its ability to **enrich the HF model by using LF model** to provide the supporting information and trends to the model for it to produce an accurate prediction as it can be seen in Figure 18c. Lastly, the MFNN can also successfully handle the rapid change within the $0.4 < x < 0.6$ range, leading to more accurate predictions of the MF values in that range (Meng and Karniadakis, 2020).

3.5.4 Enhancing CFD simulations using MF surrogate models

In the field of fluid dynamics, models such as MF Multi Layer Perceptron (MF-MLP) and Multi Fidelity Gaussian Process Regression (MF-GPR) have previously demonstrated their performance improving capabilities on various CFD simulations such as Navier–Stokes (RANS) and Large Eddy Simulation (RES).

An example study of "Multi Fidelity Surrogate Modelling of Wall Mounted Cubes" has been presented in this sub-section to provide further insights about real-word capabilities and advantages of utilising MF-surrogate models compared to singular surrogate models and standalone CFD simulations.

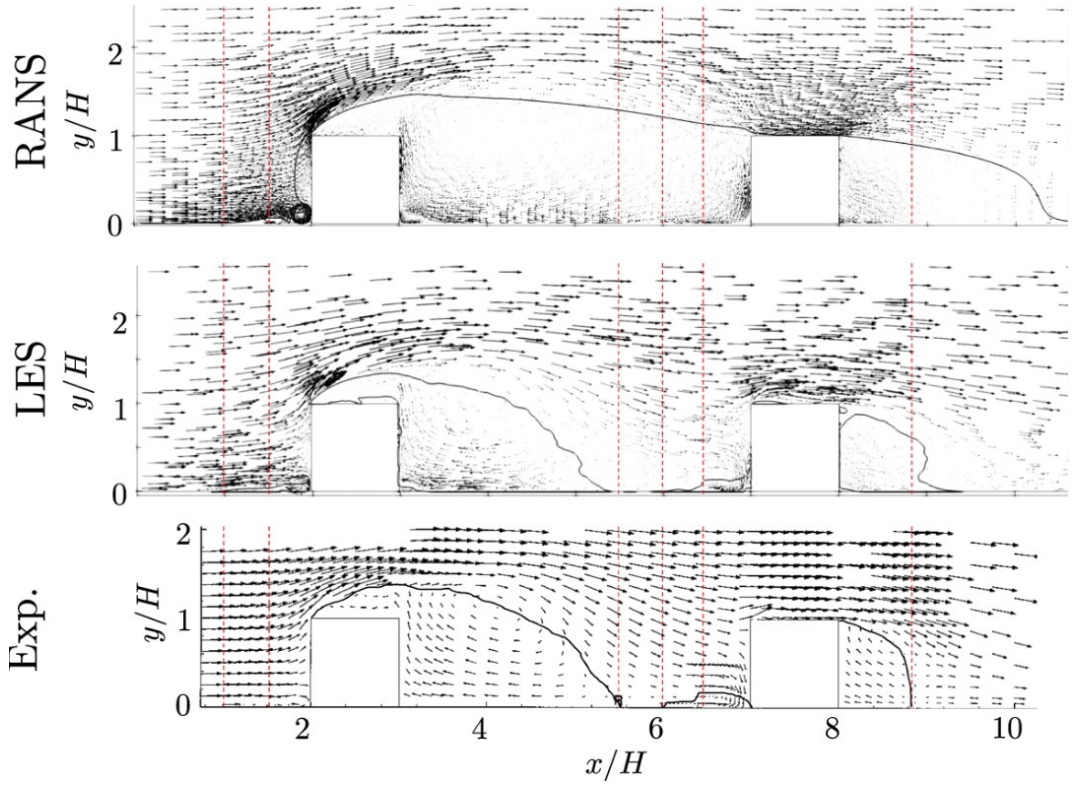


Figure 19: Comparison of RANS, LES and experimental results over the sliced flow field of two wall mounted cubes (Mole, Skillen, and Revell, 2020)

Figure 19 illustrates comparison of fluid flow over two cubes from a side view, with results produced by RANS, LES and a real-world experiment from top to bottom respectively. As can be seen, the HF LES model accurately matches the experimental results, demonstrating that running LES which is an expensive and computationally demanding model can be a great alternative to running real-word experiments. On the other hand, the LF RANS which is cheaper and quicker to run, fails to capture the fine details captured by the HF model, such as the flow separations and re-attachments indicated by the red vertical lines in the figure.

Subsequently, due to the high cost of LES and the low accuracy of RANS, each model presents its set of advantages and disadvantages, requiring a trade-off to be made, resulting in either a loss in accuracy to increase the simulation's parameter space by using RANS or vice versa with LES. As a solution to this problem, a MF surrogate model can be utilised to combine these two simulations to form a both accurate and flexible model. Ultimately improving the efficiency and cost-effectiveness of the modelling process which is crucial in the engineering academia and industry (Mole, Skillen, and Revell, 2020).

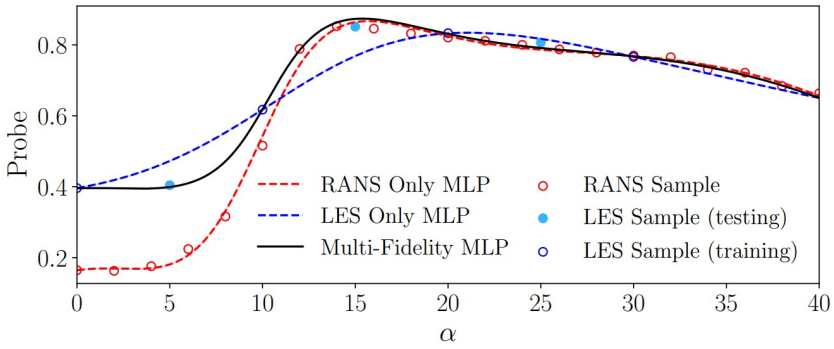


Figure 20: MFNN predicting numerical analysis (Mole, Skillen, and Revell, 2020)

Referring to Figure 20 which represents the LF, HF and MFNN predictions, similar to the figure in Section 3.5.3 can be seen. In this figure, the velocity probe over the angle of attack has been illustrated, with the raw data and single fidelity models plotted for RANS (LF), LES (HF) and the MF-MLP surrogate model (MFNN). Additionally, 21 and 4 training samples have been used to train the LF and HF models respectively.

Firstly, the LF model presented by the red dashed line, performs well in terms of forming predictions on the LF points (red dots) it was trained on, as there were enough data points (21) for the model to capture the relationship in the data. Although, the LF model significantly deviates from the targeted LES Sample testing points, presented by the light blue circles, primarily because the target points were not included in the LF model's training data pool.

Secondly, the HF model presented by the blue dashed line and trained on 4 HF data points, presented by dark blue circles, fails to produce an accurate prediction that align well with the target testing points due to the underfitting of the model caused by the lack of training points. Resulting in a model that cannot properly capture the relationship in the data, leading to inaccurate predictions.

On the other hand, the MF-MLP (MFNN) model, presented by the black line, performs significantly better than both models, forming an accurate prediction that passes through target points while not underfitting. This is due to MF surrogate model forming a relationship between the HF and LF data for the model to better capture the underlying patterns lying between the data. Hence, demonstrating the advantages of using MF surrogate models in achieving improved prediction accuracy and decreasing the reliance on HF data points by balancing the trade-off between the model's training cost and performance in an engineering application based example (Mole, Skillen, and Revell, 2020).

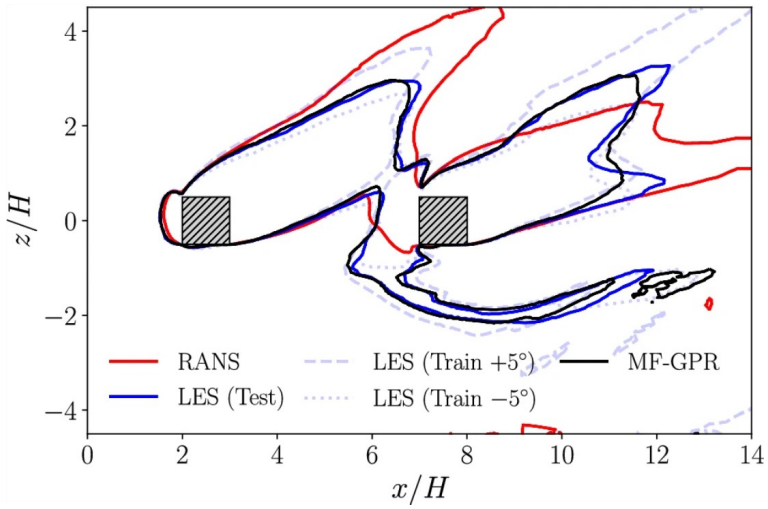


Figure 21: MFNN predicting numerical analysis (Mole, Skillen, and Revell, 2020)

Additionally, referring to Figure 21, the results obtained from the MFNN model can be shown using line contours of the velocity in the x direction, where HF, LF and MF-GPR (MFNN) model's contours are shown at a specific flow angle and height, with additional HF (LES) contours (blue dashed lines) of +/- 5 degrees added at surrounding training locations. As a result, the figure illustrates the MF model's improved prediction range and capabilities at locations in the parameter space away from the HF training data, by using the guidance of the LF points to predict results at flow angles not included in the HF training data set (Mole, Skillen, and Revell, 2020).

Proving that MF surrogate models can be more effectively utilised compared to singular

surrogate models and standalone simulations, in order to increase the accuracy and extend the parameters space of the simulation they are being trained on.

3.5.5 Additional methods to improve MFNNs

Introducing MFNNs to form surrogate models, introduces various new approaches to enhance the performance of NN surrogate models to further improve their accuracy and efficiency. Such as the incorporation of embedding theory and use of CNNs to form stronger relationships between the LF and HF data in a MF surrogate model.

A. Embedding Theory

Firstly, implementing embedding theory makes it possible for the MFNN to learn more complex non-linear correlations between the fidelities. Thus, increasing the MFNN model's prediction accuracy and capability by forming a better correlation between the LF and HF data sets. An illustrative diagram on how embedding theory can be implemented into a 2-step model is illustrated in Figure 22 below (Meng and Karniadakis, 2020).

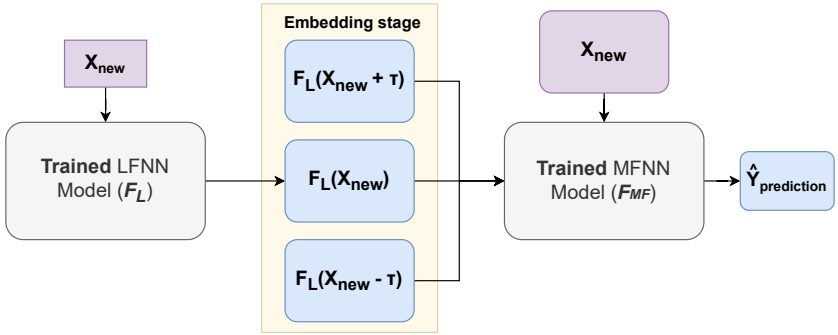
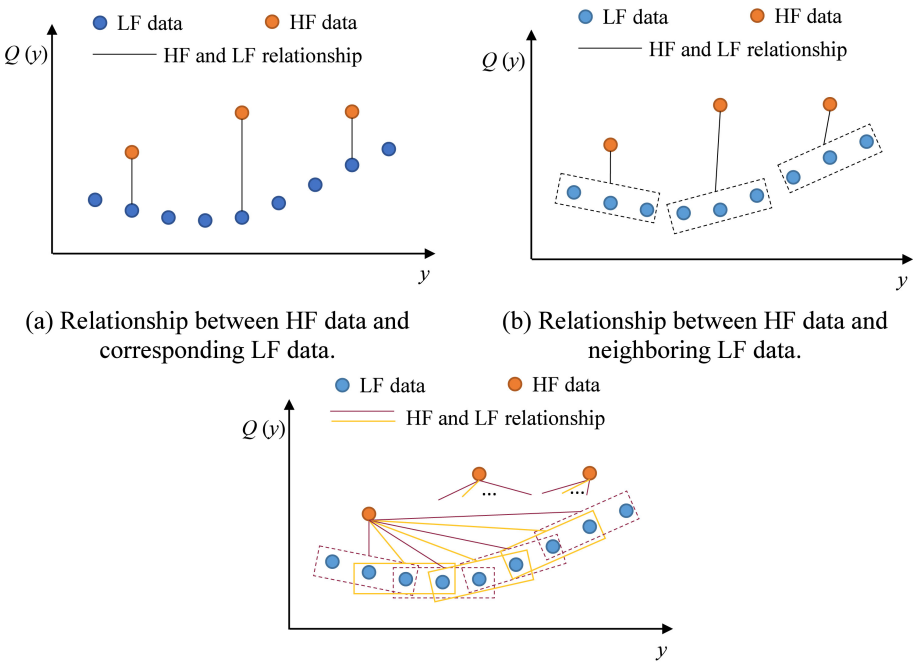


Figure 22: On surface illustration of embedding theory

Embedding theory can be introduced by adding a new hyperparameter into the MFNN network, called 'shift in the parameter space' τ (Mole, Skillen, and Revell, 2020). This hyperparameter can then be added or subtracted from two additional LFNN predictions before they are fed into the MFNN as additional inputs, hence increasing the model's prediction flexibility and range. As an example, embedding theory have been used in previously mentioned literature in Section 3.5.3 for the network to produce more accurate predictions.

B. MF data aggregation using CNNs

Secondly, MF data aggregation using Convolutional Neural Networks (CNNs) can be used to form a stronger relationship between the LF and HF data sets. This can be achieved by using a convolutional layer to form a stronger relationship between a single HF point and multiple LF points.



(a) Relationship between HF data and corresponding LF data.

(b) Relationship between HF data and neighboring LF data.

(c) Relationship between HF data and every LF datum.

Figure 23: (Chen, Gao, and Liu, 2022)

Figure 23a represents point to point relationships formed between HF and LF data in a typical MFNN model, this configuration is the norm for MFNNs. Although, referring to Figure 21b, a convolutional layer (represented by rectangular box) can be used to link a single HF point to multiple LF points in its neighbourhood, making it possible for the HF data to develop a better relationship with the LF data. Additionally, in Figure 23c, this rectangular box can then be sequentially moved through the entire LF data set for each individual HF point to form relationships with every LF datum in the training data set. As a result, forming a better performing MFNN, similar to the example given from literature in Section 3.5.3 which also utilises CNNs (Chen, Gao, and Liu, 2022).

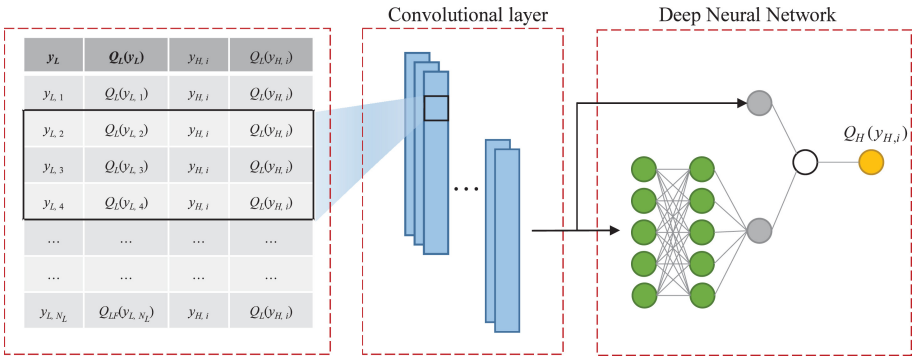


Figure 24: Multi Fidelity Data Aggregation using Convolutional Neural Networks (Chen, Gao, and Liu, 2022)

Referring to Figure 24, methodological application wise, this can be achieved by compiling the input LF and HF data in a specific order. Then, by running a convolutional layer over the compiled data to be fed into the MFNN network, to form the desired and better trained NN model.

3.6 Summary

To summarise the literature review section, it demonstrated the capabilities of using singular surrogate models by providing further detail about NN surrogate models and their uses in engineering. Moreover, going through their flaws and how they could be improved with the use introduction of MF surrogate models.

Then, fidelity in modelling & simulation has been introduced with its importance in achieving accurate and flexible surrogate models, and how MF surrogate models could be utilised to enhance their range of use in industry. Furthermore, multiple types of MFNNs to form MF surrogate models such as 2-step models were introduced and proper examples from the literature using mathematical functions and its possible uses in engineering have been provided. Ultimately, pointing out their ability to reduce the dependency on expensive HF data points in the engineering industry and academia.

Also, additional methods of enhancing MFNNs such as implementation of embedding theory and use of CNN models to form stronger relationships between varying fidelities of data have been discussed.

Lastly, one of the main goals of this section was to ensure that the reader fully understands what Multi Fidelity Neural Network models are, the founding steps needed to build them, how they function and how they can be used in real-world applications, to positively impact the sustainability and efficiency in the world of engineering.

4 Methodology

4.1 Chapter overview

The methodology section goes over the crucial steps in forming NNs and MFNNs, with the use of helpful code snippets and detailed explanations. Following their formation, methods to benchmark and validate the model using numerical examples are shown. Subsequently, building up to solving aerodynamic properties of NACA aerofoils by introducing a new method of utilising the MFNN as a multi model network.

Focusing on the less technical and more practical part of the Methodology, the student's learning process of ML and NNs are briefly explained with proper examples for the reader to take as an example and follow through. Additionally, the software and hardware needed to form the MFNN models are discussed in detail. Lastly, all the code snippets mentioned in this section are available in Appendix B as complete codes for the reader's better understanding and use to test the model.

4.2 Learning Machine Learning

Machine Learning is a branch that is not covered in the Aerospace Engineering curriculum at The University of Manchester, hence the student had to learn Machine Learning individually from scratch using external sources and with the aid of his academic supervisor throughout academic year.

The learning process was a step by step process where, the student first gained a basic understanding of Machine Learning methods, including the basic theory, notations, linear regression and learning algorithms to build up to the Deep Neural Networks with the use of "The Hundred Page Machine Learning Book" by Andriy Burkov. In addition, external educational sources such as Andrew Ng's public ML CS229 lectures notes and educational YouTube videos were used. Hence, making it possible for student to reach a level capable of understanding various research papers, books and ML framework documentations, to turn the theoretical knowledge into practical skills in forming MFNNs.

Additionally, this process included understanding the theory and concepts while also putting them into practice, by working with synthetic and real-world datasets, such as examples taken from literature, and NACA aerofoil example provided in the report's poster presentation.

In terms of coding practice, the learning process was conducted sequentially. Initially, by learning basics of ML, then forming a Gaussian Process Regression model (GPR) and MF-GPR model using GPyTorch package. Followed by forming a singular NN using

PyTorch framework, which led to the development of a MFNN using the same framework. The entire process took approximately 4.5 months starting from October 2022, and all the code had been written from scratch with the aid of official documentation provided by PyTorch's website.

4.3 Required software and packages

In this report, PyTorch framework had been used to form the 2-step MFNN model, along with various software and packages such as, NumPy to initialise the data sets as the first step and matplotlib to visualise the final MFNN's predictions as the last step.

4.3.1 PyTorch Machine Learning framework

Various ML models such as DNNs and MFNNs can be formed using various techniques and ML frameworks such as, TensorFlow, Sci-Kit Learn, PyTorch and more. Due to its great documentation, clarity and flexibility, PyTorch has been picked to build the MFNN model in this report.

PyTorch framework offers a rich library consisting of built-in functions needed to form ML models. Hence, allowing the user to focus on designing models rather than implementing basic ML functions from scratch, such as gradient descent and chain rule for backpropagation. As a result, simplifying the process of building and training ML models.

4.3.2 Supporting packages and software

Before setting up the PyTorch framework, Python language and its various mandatory packages must be downloaded to the device or to the cloud-environment, where the coding is going to be taking place. In this report's case, Windows 10 Home (Version 22H2) Operating System (OS), containing Python 3.9.10 have been used. Additionally, to install all mandatory packages, Python Package Manager 'PIP' have been used with a special virtual environment (venv) created for the report to prevent any un-needed packages existing in the OS from clashing. Moreover, Project Jupyter Notebook extension included in Visual Studio Code (VsCode) IDE had been used to write the codes throughout the project's lifetime, as it offers better de-bugging and code editing and testing capabilities.

Furthermore, crucial Python packages such as, Numpy, Torch and Pandas have been used to create the proper tensors[5] needed to pre and post process the data that ran through

[5]A generalisation of vectors or matrices, that can easily be processed by computers as multi-dimensional arrays.

the PyTorch framework. Whereas, data visualisation packages such as Matplotlib and Seaborn have been used to visualise the results produced by the NN models.

Lastly, online hosting services such as GitHub workspaces and Google Drive, in addition to external hard-drives have been used to backup the code and sources that have been used in this report.

The PIP list containing the Python packages used is included in Appendix B and PIP version 23.1.2 was being used at the time of writing.

4.4 Hardware Specifications

4.4.1 Device setup

The models were run on an MSI GS66 10SE-041 laptop, capable of running a typical DNNs around 10-1000 neuron per 1-10 hidden layers with no issues. Although, it was noted that the run times were significantly increased when more than 2000 neurons per hidden layer in a 8 hidden layer NN was introduced. The hardware specifications of the laptop is outlined in Table 2.

Table 2: MSI GS66 Stealth 10SE-041 Hardware Specifications

Device	**MSI GS66 Stealth 10SE-041**
Processor	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
RAM	16.0 GB (2933 MHZ)
GPU	Nvidia GeForce RTX 2060 Mobile - 90 Watt

4.4.2 GPU utilisation using CUDA

NVIDIA CUDA Toolkit[6] is a parallel computing platform for NVIDIA graphical processing units (GPUs), which enables PyTorch models to run through the GPU instead of the CPU of the device. Hence, significantly reducing the model's run time by making it possible to perform parallel calculations using GPUs.

Listing 1: Setting up CUDA in PyTorch

```
1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') # For
  ↪  activating of CUDA
2 example_data = example_data.to(device) # To transfer data from CPU to GPU
```

After installing CUDA Toolkit using its installation package, CUDA can be implemented into the model, enabling the transfer of data to the GPU to utilise parallel processing, as demonstrated in the example code above. Additionally, it was observed that in the report that, use of CUDA through the RTX2060 GPU instead of the processor shortened the model run times by a significant margin.

[6]https://developer.nvidia.com/cuda-toolkit

4.5 A Neural Network model's structure

It is strongly advised for the reader to refer to Section 2 to refresh their knowledge about the basics of Neural Networks, as this sub-section provides the same theoretical knowledge, but in a practical way using Python and PyTorch.

A NN using PyTorch or any framework can be formed through following these six major steps:

1. **Importing Essential Packages:** Python packages such as, Torch, NumPy and matplotlib must be imported to be able to perform data pre-processing, run the NN model and demonstrate the prediction using data visualisation.

2. **Data pre-processing:** The data going to be ran through the NN must undergo proper processing and labelling before it can be used effectively.

3. **Setting Hyperparameters:** Defining parameters such as number of neurons, learning rate, and number of epochs, must be performed before the model can be run.

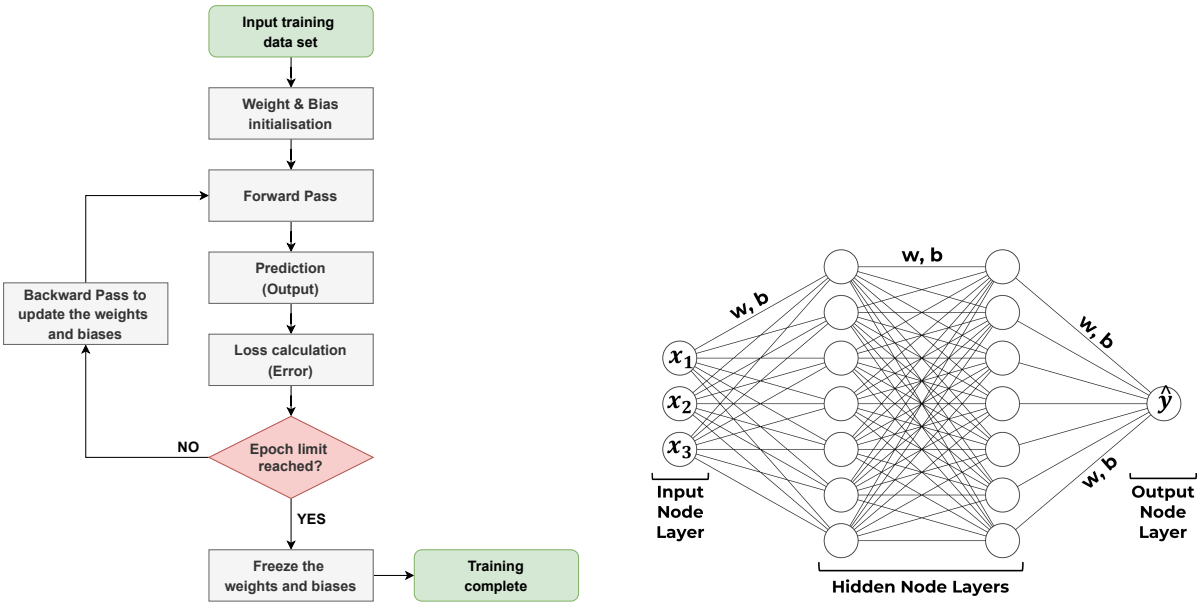
4. **Initialising NN model's structure:** Defining the dimension of input and output layers, number of hidden layers, and each neuron's respective activation functions.

5. **Initialising the training loop:** Defining the model's loss function and optimiser algorithm, and configuring the training loop's structure for the model to be properly trained.

6. **Model evaluation and forming predictions:** Use of methods such as, recording the change in loss of the model to determine the trained model's prediction accuracy. Hence, enabling it to form predictions on unseen data sets.

Important: In this section, the NN will be illustrated using multiple code snippets of the major components of the NN as enumerated above. It is recommended for the reader to refer back to Training Overview Section 2.3 in Background Theory, before they read through the sub-sections.

Please refer to Appendix B for the entirety of the complete code provided for the reader to grasp the full understanding of the code as a whole.



(a) NN training flowchart (b) Basic DNN schematic

Figure 25: Training process and structure of a DNN

Additionally, for the reader's better understanding, the training overview and an example DNN's schematic are provided again in Figure 25.

4.5.1 Importing essential packages

Listing 2: Importing Essential Packages

```
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') # CUDA
↪ device configuration
```

Before, any data processing can be made and NN model can be run, packages essential to the code's operation must be imported. Including, NumPy and Pandas to pre-process the data, Torch to run the MFNN model and matplotlib to visualise the results.

4.5.2 Data pre-processing

Listing 3: Setting up and pre-processing synthetic data set

```
def F(y):
    return np.sin(8 * np.pi * y) # Target function (synthetic data)

X_train = torch.linspace(0, 1, training_data)[:,None] # Model's training and
↪  testing data sets
Y_train = F(X_train)
```

In this example's case, synthetic data was used to test the capabilities of the NN, with adjustable training data points and target to adjust the complexity of the NN to further test it. In practice, data often comes from large databases stored in formats such as .csv or SQL. This step ensures that the data being fed into the NN undergoes desired processing and labelling to guarantee effective and accurate training.

4.5.3 Setting Hyperparameters

Listing 4: Model Hyperparameters Initialisation

```
num_epochs = 1000 # No. of epochs
input_dim = 1; output_dim = 1 # Input & Output dimensions
n_per_layer = ([500, 500, 500]) # Neurons per layer
learning_rate = 0.0001 # Learning rate of optimiser
training_data = 13 # Number of training points

```

Before the training can start, the hyperparameters such as the number of epochs, neurons per layer, input and output dimensions of the model and the learning rate of the optimiser are set. These hyperparameters can differ depending on the complexity of the problem that is aimed to be solved.

4.5.4 Initialising NN model's structure

Listing 5: NN Structure with 2 Hidden Layers

```
# -----Model Structure Initialisation with 2 Hidden Layers-----
class Network(torch.nn.Module):
  def __init__(self, n_per_layer, input_dim, output_dim): # Layer setup
    super().__init__()
    self.fc1 = nn.Linear(input_dim, n_per_layer[0])  # Input Layer
```

```
6       self.fc2 = nn.Linear(n_per_layer[0], n_per_layer[1]) # Hidden Layer 1
7       self.fc3 = nn.Linear(n_per_layer[1], n_per_layer[2]) # Hidden Layer 2
8       self.fc4 = nn.Linear(n_per_layer[2], output_dim) # Output Layer
9
10    def forward(self, x): # Activation function after each layer (ReLu)
11      x = torch.relu(self.fc1(x))
12      x = torch.relu(self.fc2(x))
13      x = torch.relu(self.fc3(x))
14      x = self.fc4(x)
15      return x
16
17  model = Network(n_per_layer, input_dim, output_dim) # Calling Python class
    ↪  as to use the model when needed
```

Beginning from line 1, the NN model is formed using the PyTorch framework's built in **torch.nn.Module** function in a Python Class and Function. The model consists of two hidden layers, enclosed by the input and output layer. Moreover, referring to line 10, a second function (**def forward**) in the model is set to assign activation functions to the end of every neuron in the input and hidden layers. There is no activation function on the output layer, as the final prediction shall not be altered by the activation function.

Lastly, on line 17, the Python Class can be called to use the model for training, or if the training is done, for predicting purposes.

4.5.5 Initialising the training loop

Listing 6: Training Loop

```
1  # -----Setting Loss Function and Optimiser-----
2  criterion = torch.nn.MSELoss() # Loss Criterion (Mean Squared Error)
3  optimiser = torch.optim.Adam(model.parameters(), lr=learning_rate)
4
5  # -----Training Loop (A single Forward and Backward pass per epoch)-----
6  for epoch in range(num_epochs): # Sets a loop for each epoch
7      y_pred = model(X_train) # Initiates forward pass
8      loss = criterion(y_pred, Y_train) # Computes loss from the prediction
9      optimiser.zero_grad() # Rests gradiets per epoch
10     loss.backward() # Calculates gradients of loss with backpropagation
11     optimiser.step() # Updates model parameters using gradients
```

Firstly, starting from line 1, the criterion and the optimiser get set using the in-built

PyTorch functions. In this case, the **criterion** (loss function) is set to be MSELoss (Mean Squared Error) and **optimiser** is set to Adam with the pre-set learning rate of 0.001 from the hyperparameters.

Secondly, moving down to line 5 of the code, the training loop initiates to be ran until the maximum **number of epochs** are reached. Within this for loop, the model initiates the forward pass by running the input (X_{train}) through the **model** to form the prediction (y_{pred}) at line 6. Then, the **loss** between Y_{pred} and Y_{train} is computed using the criterion at line 7, following that, the gradient of the optimiser is set to **zero** to prevent from the previous gradient from mixing with the current one at line 8. Next, the gradients of the loss is calculated with the **backward** pass at line 9. Finally, the **optimiser.step()** updates the model's parameters using the gradients and the for loop continues to run.

To conclude this section, this section goes over the core of a NN model and its structure in PyTorch, aiming for the reader to be able to apply this knowledge to other languages and frameworks such as TensorFlow or Sci-Kit Learn, as they all function in a similar manner. Additionally, it is highly recommended for the reader to read over the complete code provided at Appendix B to form a great foundation of this process and carry this to the next section about the structure of MFNNs.

4.5.6 Model evaluation and forming predictions

Listing 7: Model evaluation and forming predictions

```
1 new_X_input = torch.linspace(0, 1, 1000)[:,None] # Random ranged data array
  ↪ for new X_input data
2 with torch.no_grad():
3     y_pred_new = model(new_X_input) # Define the NEW input data
4 # Plotting the results and change in loss using matplotlib (Provided in
  ↪ Appendix B)
```

After the training is complete, the model gets evaluated by plotting its change in loss during the training, on training and validation sets. If the model performs well, and the losses converge at values close to zero, this indicates that the model has been trained effectively.

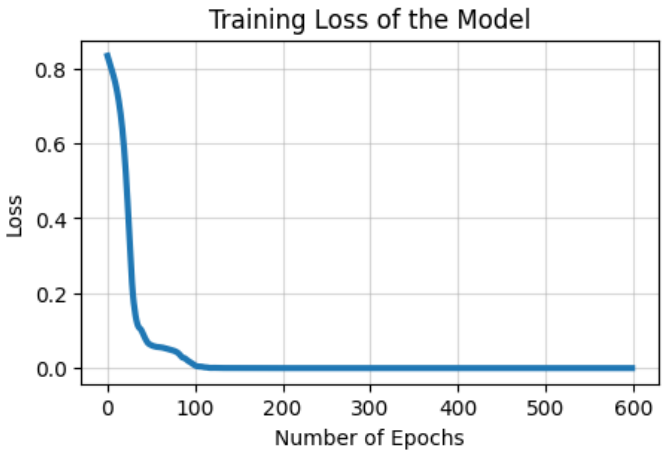


Figure 26: NN's change in loss per epoch

Referring to Figure 26, the change in loss and its convergence to zero can be illustrated by logging the loss at each epoch and then plotting it over the number of epochs.

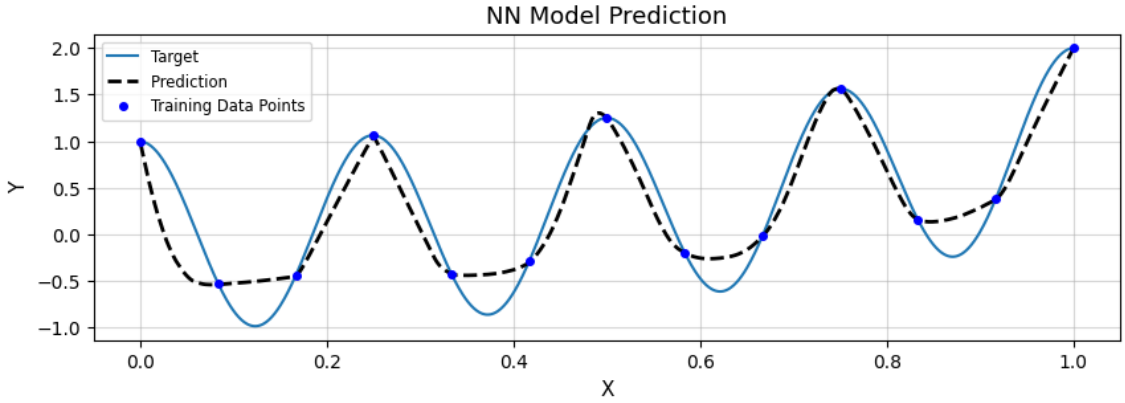


Figure 27: NN model's prediction of a sine wave

Additionally, as illustrated in Figure 27, the trained NN's predictions can be visually represented to demonstrate the model's performance in capturing the underlying pattern of the sine wave function it was trained on.

4.6 A Multi Fidelity Neural Network structure

MFNNs ultimately function in a similar manner to singular NNs, with only difference being the sharing of LF and HF data between the two models, to form the MFNN.

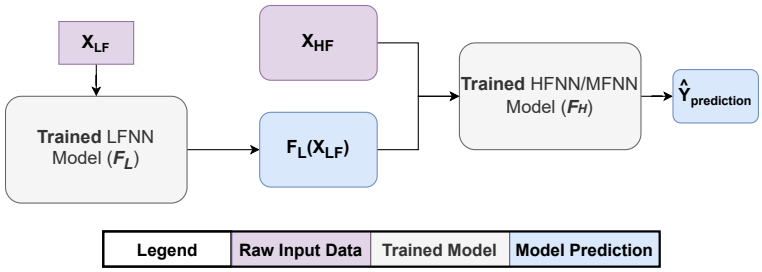


Figure 28: MFNN structure flowchart

For the reader's better understanding of the MFNN formation process, Figure 28 which was previously mentioned in Literature Review section, is shown again. Demonstrating the combination process of the HF input X_{HF}, with the LF output from the LFNN model ($F_{L(X_{LF})}$), being fed into the MFNN model.

Hence, making it possible for the MF model to share the correlation between the two fidelities of data and NN models inside it. Furthermore, it is strongly advised for the reader to refer to Section 3.5.2 to refresh their understanding of MFNN structures and their formation before continuing this section.

A 2-step MFNN can be formed by following these major steps:

1. **Importing and pre-processing LF and HF data:** Import or create the LF and HF training, testing and validation data. Ensuring that the LF and HF data and their respective models are compatible with each other after pre-processing, to ensure an effective MFNN model can be formed.

2. **Pre-processing both LF and HF data sets:** The imported or created data is then standardised or split into training, testing and validation sets. More processing techniques can be utilised depending on the type of the data.

3. **Training individual LFNN and HFNN models:** Form the LF and HF models respectively, ensuring that they are properly trained and ready to form predictions on their own.

4. **Forming the MFNN training loop by combining the fidelities:** Combine the fidelities and their models to form the MFNN model, following the theory mentioned in Literature Review.

5. **Model evaluation and forming predictions:** Once the MFNN model is trained, the MFNN predictions are obtained, and are assessed against the LFNN and HFNN models, to evaluate its performance.

Similar to the previous section, only crucial snippets of the code had been mentioned in this section, enabling the reader to gain a solid understanding of MFNNs and attempt to replicate the process themselves. Similarly, please refer to Appendix B for the complete source code.

The code given below is taken from the MFNN model formed to solve the synthetic data set examples in the results section, which will be further discussed later in the report.

4.6.1 Importing Data

Listing 8: Setting up the synthetic data set

```
#-----Setting up the synthetic data set-----
# Setting the complxity of the synthetic functions
def LF(y): # LF output function
    return 0.5 * (6 * y - 2) ** 2 * np.sin(12 * y - 4) + 10 * (y - 0.5) - 5

def HF(y): # HF output function
    return (6 * y - 2)**2 * np.sin(12 * y - 4)
```

After the essential packages have been imported, the LF and HF datasets can either be imported using Pandas or SQL, or be synthetically formed using mathematical functions. In this example snippet, synthetic LF and HF data sets are formed with the use of two Python def functions.

4.6.2 Data Pre-processing

Listing 9: Pre-processing LF and HF data

```
X_LF_train, X_LF_test, Y_LF_train, Y_LF_test = train_test_split(LF_TS,
↪  LF(LF_TS), test_size=0.8, shuffle=156)
X_HF_train, X_HF_test, Y_HF_train, Y_HF_test = train_test_split(HF_TS,
↪  HF(HF_TS), test_size=0.8, shuffle=156) # Splitting the data sets into
↪  training and testing sets per fidelity

# -----Additional pre-processing methods-----
Scaler1 = StandardScaler()
```

```
6  X_train = pandas.DataFrame(Scaler1.fit_transform(X_train)) #  removes the
   ↪  mean and scaling to unit variance
7
8  X_train = X_train.fillna(X_train.mean(),inplace=True) # replaces NAN values
   ↪  with mean of the data set
```

After forming the data sets, they are then divided into respective training and testing sets on line 1 using the **train_test_split** function provided by Sci-Kit Learn framework. This process prepares both LF and HF data to be run through the MFNN, and also provides a method to adjust the number of training and testing points.

Additionally, starting from line 8, other techniques such as, **StandardScaler** can be used to standardise data's features by removing its mean and scale it to unit variance. Furthermore, **fillna** function can also be used to fill the empty cells in the training data, which can improve the model's training capabilities (Pedregosa et al., 2011).

4.6.3 Training LFNN and HFNN models

LF and HF models can be trained and formed by going over the major steps discussed in sub-section 4.5 above that are used to form a singular NN model.

4.6.4 MFNN training loop

Listing 10: Forming the MFNN training loop by combining the fidelities

```
1  LF_pred = LF_model(X_HF_train) # LFNN prediction of LF data
2  mixed_fidelity_array = torch.hstack((X_HF_train, L1mean)) # stacking arrays
   ↪  side to side
3
4  # MFNN NN Model Structure (Identical to a typical NN - Appendix B)
5  MF_model = MultiFidelityNetwork(hidden_dims, MF_input_dim, MF_output_dim)
6
7  # ------MFNN Training Loop-----
8  MF_model.train()
9  criterion = torch.nn.MSELoss()
10 optimizer = torch.optim.Adam(MF_model.parameters(), lr=0.00005)
11
12 for epoch in range(MF_epochs):
13     MF_y_pred = MF_model(mixed_fidelity_array)
14     MF_loss = criterion(MF_y_pred, Y_HF_train) # Trains the MFNN model using
       ↪  the mixed array and HF training set
```

```
15      optimizer.zero_grad()
16      MF_loss.backward(retain_graph=True)
17      optimizer.step()
```

As seen on line 1, to form the MFNN, the LFNN prediction and HF input are combined to form a single tensor (array) by stacking the data sets with each other. This forms an array consisting of various fidelities, show on line 2. Then, this MF array is fed into the MFNN model. Hence, forming a relationship between the mixed fidelities and the HF target during the training loop, pointed out on line 14. As a result, forming the MFNN model capable of forming connections between fidelities.

4.6.5 Model evaluation and forming predictions

Listing 11: Forming predictions

```
1  with torch.no_grad(): # prevents the gradients of the models from changing
   ↪ during prediction
2      y_LF_pred = LF_model(X_new_input) # predtion obtained from the singular
       ↪ NN model
3
4      prediction_array = torch.hstack((X_new_input, y_LF_pred)) # combining
       ↪ new X_input with LF prediction
5      y_MF_pred = MF_model(prediction_array) # Running the combined array
       ↪ through MFNN
```

After the training is complete, to obtain the MFNN prediction. Firstly, the new input (**X_new_input**) is ran through the LF model first. Then, the prediction obtained from that model is combined with the input data again, in the same fashion as sub-section 4.6.4 above. Finally, the combined **prediction_array** is ran through the network to obtain the final prediction of the model. The reader can refer to Figure 28 again for a better understanding of this process. Similar to singular NN models, after the predictions are obtained all three models model gets evaluated by plotting their respective changes in loss during their training, and validation stages. In addition to singular NNs, all model's losses must converge to zero for the model to be trained effectively.

4.6.6 MFNN model optimisation

NN models offer a large range of flexibility in terms of accepting possible improvements. Various techniques either manual, or pre-built PyTorch functions can be implemented into the model. Such as, batch normalisation, skip connections and embedding theory.

Lastly, embedding theory had been given in further detail, as it is a significant part of the MFNN.

Embedding Theory: As mentioned before in the report, implementing embedding theory makes it possible for the MFNN to learn more complex non-linear correlations between the fidelities. In result, increasing the MFNN model's prediction accuracy and capability.

Listing 12: Implementing Embedding Theory

```
1 embedding = 0.43 # embedding hyperparameter (tau)
2 LF_pred = LF_model(X_HF_train)
3 LF_pred_up= LF_model(X_HF_train + embedding)
4 LF_pred_dn = LF_model(X_HF_train - embedding)
5
6 combined_array = torch.hstack((X_HF_train, LF_pred, LF_pred_up, LF_pred_dn))
  ↪ # Combine the embedded predictions in addition to the LFNN prediction
7 # the 4 dimensional array is then fed into the MFNN model with an input
  ↪ dimension of 4 instead of the typical 2.
```

Referring to the snippet above and Figure 22 illustrating embedding theory. The embedding theory introduces another hyperparameter τ making it possible to shift the LFNN prediction by a small margin as can be seen from lines 1 to 3. Then, the LFNN model's predictions and HF X_input is combined on line 6, just like in the previous subsection. Consequently, embedding results in better prediction capabilities of the MFNN, by increasing its generalisability and its effect against phase-shift between the LF and HF data.

4.7 Testing MFNN Models using numerical examples

To evaluate the performance of the MFNN to ensure that it can handle complex real-word problems is to test it with progressively harder numerical problems problems. Aiming to increase its prediction and generalisation capabilities. Consequently, numerical examples including linear and non linear or continuous and discontinuous functions, become a great choice to benchmark the MFNN model.

In addition to this, the functions adjustable range, and adjustable phase-shift between the fidelities also play a significant role in testing the network. Moreover, the number of training LF and HF can also be adjusted to push the model into its generalisation limits. Additionally, the numerical functions sourced from (Chen, Gao, and Liu, 2022) to test the predicting capabilities of the MFNN model is provided in Table 3.

In addition to the provided table, the synthetic numerical problems formatted in Python are included in Appendix B for the reader.

Table 3: LF and HF synthetic numerical examples (Chen, Gao, and Liu, 2022)

No.		LF model	HF model
(a)	Continuous functions with linear relationship	$Q_L(y) = 0.5(6y-2)^2\sin(12y-4) + 10(y-0.5) - 5, \quad 0 \le y \le 1$	$Q_H(y) = (6y-2)^2\sin(12y-4), 0 \le y \le 1$
(b)	Discontinuous functions with linear relationship	$Q_L(y) = \begin{cases} 0.5(6y-2)^2\sin(12y-4) + 10(y-0.5), & 0 \le y \le 0.5 \\ 3 + 0.5(6y-2)^2\sin(12y-4) + 10(y-0.5), & 0.5 < y \le 1 \end{cases}$	$Q_H(y) = \begin{cases} 2Q_L(y) - 20y + 20, & 0 \le y \le 0.5 \\ 4 + 2Q_L(y) - 20y + 20, & 0.5 < y \le 1 \end{cases}$
(c)	Continuous functions with nonlinear relationship	$Q_L(y) = 0.5(6y-2)^2\sin(12y-4) + 10(y-0.5) - 5$	$Q_H(y) = (6y-2)^2\sin(12y-4) - 10(y-1)^2$
(d)	Continuous oscillation functions with nonlinear relationship	$Q_L(y) = \sin(8\pi y), 0 \le y \le 1$	$Q_H(y) = (y-\sqrt{2})Q_L^2(y), 0 \le y \le 1$
(e)	Phase-shifted oscillations	$Q_L(y) = \sin(8\pi y)$	$Q_H(y) = y^2 + Q_L^2(y + \pi/10)$
(f)	Different periodicities	$Q_L(y) = \sin(6\sqrt{2}\pi y)$	$Q_H(y) = \sin(8\pi y + \pi/10)$
(g)	50-dimensional functions	$Q_L(y) = 0.8Q_H(y) - \sum_{i=1}^{49} 0.4y_i y_{i+1} - 50, -3 \le y_i \le 3$	$Q_H(y) = (y_1 - 1)^2 + \sum_{i=2}^{50}(2y_i^2 - y_{i-1})^2, -3 \le y_i \le 3$

Lastly, these synthetic numerical functions are then implemented into the MFNN model as shown in Section 4.6.1.

4.8 An alternative use of MFNN models

Due to Multi Fidelity NN models excellence in combining data using multiple NN models. A similar structure like a **Multi Model Neural Network**[7] (MMNN) can be utilised to develop networks that can form predictions based on pre-existing historical data, in contrasts to using different fidelities in a MFNN. Subsequently, forming models that can generate predictions utilising pre-existing (historical) data sets, similar to the singular NN method discussed in Section 3.4.1.

Although, while performing the same 'forming predictions using historical data' task as the singular NN model, this new model offers more flexibility in its training by allowing multiple inputs of data to be run through each NN model, compared to the single input data used in a singular NN model. In addition, due to its superior capability of forming relationship between the data, it can produce way more accurate results compared to a singular NN model.

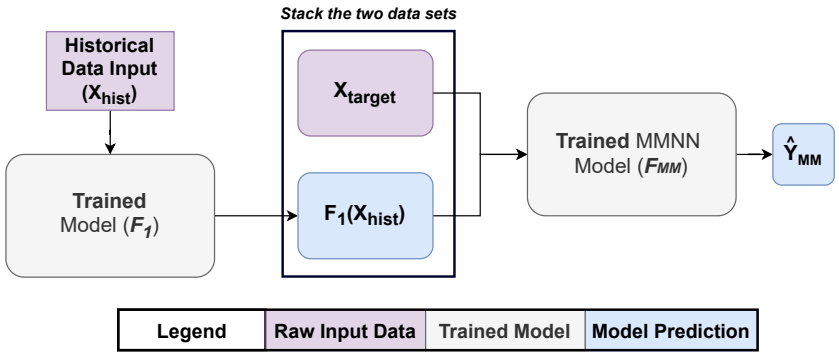


Figure 29: Flowchart illustration of a MMNN model

A MMNN is structured exactly identical to a MFNN, with only difference being the type of data running through the models. As illustrated in Figure 29, the pre-existing data is fed into the first network. Then, the prediction of that network and the target that is aiming to be predicted is combined, and fed into the second network that produces the final prediction.

As an example, an array containing pre-existing set of NACA aerofoil aerodynamic parameters can be fed into the first network (replacing LFNN), whereas the new NACA aerofoil that is aiming to be predicted with its few existing data points (replacing HFNN), can be fed into the network. Hence, forming a MMNN network that can be used to accurately predict the aerodynamic parameters of the newly introduced aerofoil by utilising the pre-existing training data.

[7]Not to be confused with Multimodal Deep Learning.

4.9 Additional techniques to improve MFNNs

nn.BatchNorm1d: As observed from this report's model, implementing batch normalisation into the model can result in significant increase in the model's training speed, as it can help the model converge faster and learn better. Additionally, as it is a function included in PyTorch and many other frameworks, it can be easily implemented into the code's structure as shown in the snippet below.

Listing 13: Implementing batch normalisation

```
class NNmodel(torch.nn.Module):
  def __init__(self, hidden_dims, LF_input_dim, LF_output_dim):
    super().__init__()
    self.fc1 = nn.Linear(LF_input_dim, hidden_dims[0])
    self.bn1 = nn.BatchNorm1d(hidden_dims[0]) # Implemented into the
    ↪ structure
    self.fc2 = nn.Linear(hidden_dims[0], hidden_dims[1])
    self.fc3 = nn.Linear(hidden_dims[1], hidden_dims[2])
    self.fc4 = nn.Linear(hidden_dims[2], hidden_dims[3])
    self.fcEND = nn.Linear(hidden_dims[3], LF_output_dim)

  def forward(self, x):
    x = torch.relu(self.fc1(x))
    x = self.bn1(x)  # Implemented into the forward pass
    x = torch.relu(self.fc2(x))
    x = torch.relu(self.fc3(x))
    x = torch.relu(self.fc4(x))
    x = self.fcEND(x)
    return x
```

However, it must be noted that implementing batch normalisation makes the hyperparameter optimisation process more fragile due to its nature, meaning that proper adjustments must be made to properly configure the model. Hence, it is important to properly configure the model to take the advantage of the improvements introduced by batch normalisation without sacrificing the prediction quality.

Embedding theory: Once again, right after the first LFNN is trained, the embedding theory can be implemented by adding an -/+ offset to the model's input. After combining the LFNN predictions and the HF X input, the 4 dimensional new array is fed into the MFNN which must be adjusted to intake 4 dimensions as an input.

4.10 Summary

Methodology section can be summarised by, how the theory lying beneath the MFNN model can be learned in a progressive manner with the aid of correct sources and guidance by a professional in this report's area. Followed by understanding how ML models can practically be formed using open-sourced software such as Python and PyTorch.

Additionally, with the help of the provided code snippets the NN's working principles have been explained, to build a solid background to form MFNN models, optimise them and then validate them using numerical examples taken from literature. Finally, building up to how they can be further modified to be used in real-world engineering applications such as aerofoil parameter optimisation. Lastly, with suitable additional techniques presented to improve the MFNNs performances. Hence, providing the reader enough knowledge and aid to build their own NN and MFNN model's structure.

5 Results and Discussion

5.1 Chapter Overview

Results and discussion chapter covers the main work done on the report to aid in proving and accomplishing its aim and objectives. The section starts with the introduction of the core of the MFNN structure that is going to be used to solve the validation numerical examples and the NACA aerofoil prediction. Then, it proceeds onto proving why MFNNs are superior to singular NN models and a potential better alternative to running standalone computer simulations. With the goal of proving the report's aim and objectives throughout the section.

Additionally, the section will go over the effect of GPU utilisation on the processing speed of the MFNN and provide an overall discussion about the results produced by the model to summarise the section.

5.2 MFNN architecture used for testing

A 2-step MFNN model structure consisting of with 6 hidden layers with ReLu activation function, MSELoss criterion, and Adam optimiser have been used per for the two NNs making up the MFNN model. Additionally, the LFNN model is equipped with adjustable batch sizes in its training loop. Lastly, embedding theory was implemented into the model and used to predict all the results in this section.

A summary of the configuration of MFNN's hyperparameters and number of LF and HF data points used to process the numerical examples are provided in Table 4.

Table 4: Hyperparameters and number of LF and HF data used while training the MFNN per problem

Problem	**No. of epochs**	**Neurons per hidden layer**	**Learning rate**	**Batch size**	**No. of LF points**	**No. of HF points**
Sine waves	600	500	0.005	100	20	7
(a)	1200	1000	0.001	12	20	6
(b)	1200	2000	0.005	100	40	12
(c)	1200	1000	0.001	100	20	6
(d)	600	1000	0.005	100	40	25
(e)	600	1000	0.005	100	40	20
(f)	1200	1500	0.0005	100	40	20
MMNN Aerofoil	2000	2000	0.00001	150	56	14

5.3 Effect of GPU utilisation on processing speed

Hyperparameters such as number of neurons per layer and learning rate greatly varies as the complexity of the problem increases as can be seen from Table 4 above. This is due to the model needing to form deeper connections between the more complex data to form proper relationships, hence increasing the process time of the models.

As a solution, previously mentioned NVIDIA CUDA Toolkit can be utilised to transfer this process from the process to the GPU of the device to enable parallel processing of the network calculations. Hence, decreasing the processing time of the models.

The processing times were measured using 'time' package in Python.

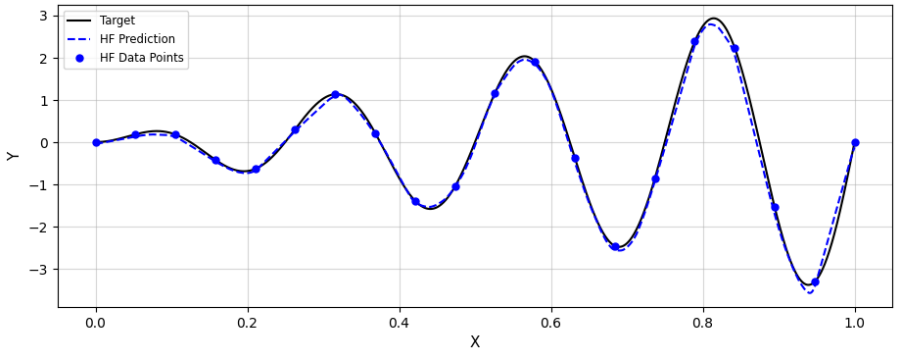
Table 5: CPU vs GPU comparison of processing speeds per problem

Problem	**CPU speed (seconds)**	**GPU speed (seconds)**
Sine wave	33	262 (4.3 minutes)
(a)	13	25
(b)	59	678 (11.3 minutes)
(c)	17	127 (2.1 minutes)
(d)	23	76 (1.3 minutes)
(e)	27	81 (1.4 minutes)
(f)	34	108 (1.8 minutes)
MMNN Aerofoil	51	854 (14.2 minutes)
Average	32.1	276.4 (4.6 minutes)

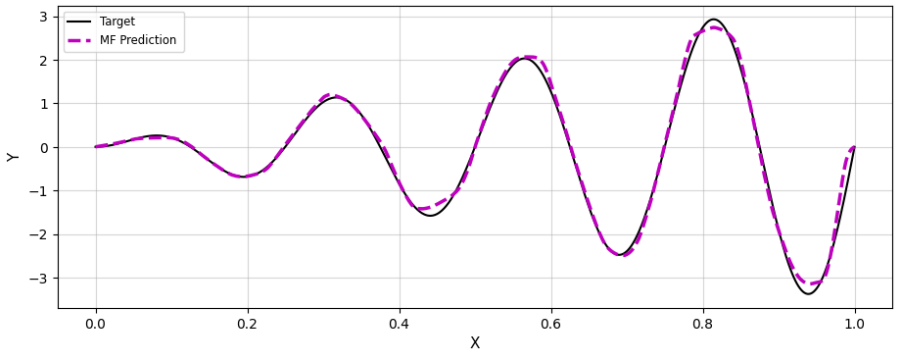
Table 5 shows the significant advantage of GPU utilisation on MFNN models running the synthetic data and NACA example that are going to be discussed in the next section. As can be seen, the utilisation of GPUs provides an average 244 seconds (4 minute) difference in processing time, indicating that the use of a device that can run parallel calculations is a must to form more effective and time efficient models. Hence, the GPU utilisation with the help of CUDA Toolkit was used to run all the models produced throughout this report.

5.4 Reducing the reliance on HF data

Referring to the aim and Objective 3 of this report, the MFNN's one of the main aims to be developed is to decrease the reliance of NN models on HF training data that are expensive and time consuming to gather in large amounts.



(a) Singular HFNN model prediction with 20 HF training data points



(b) MFNN prediction with only 7 HF and 20 LF training points

Figure 30: Comparison of number of HF points required per model

Figure 30 illustrates the HF data reduction capabilities of a MFNN model by comparing it to a singular HFNN model. As can be seen, the HFNN model requires at least 20 HF training data points to form an accurate prediction, whereas the MFNN only needs 5 HF and 20 cheaper LF data points in comparison. Hence, proving the MFNN models effectiveness in decreasing costs of obtaining expensive HF data by utilising it less to form accurate predictions. Additionally, Figure 30b will be explained and shown in more detail in the next section.

5.5 Performance testing and model validation using numerical examples

Analytical functions offer a good range of test to benchmark and validate MFNN models, they do contain complicated and abruptly changing trends in them that are hard for Neural Networks to catch. The aim of running this test is to prove that MFNNs can overcome these challenges and check if the model actually perform better compared to singular NN models (Objective 1,2 and 3).

The complexity of the given numerical examples of various LF and HF functions progressively increases as the list below goes on:

- Basic sine waves
- (a) Continuous functions with linear relationships
- (b) Discontinuous functions with linear relationships
- (c) Continuous functions with nonlinear relationship**
- (d) Continuous oscillation functions with nonlinear relationship**
- (e) Phase shifted oscillations
- (f) Different periodicities**

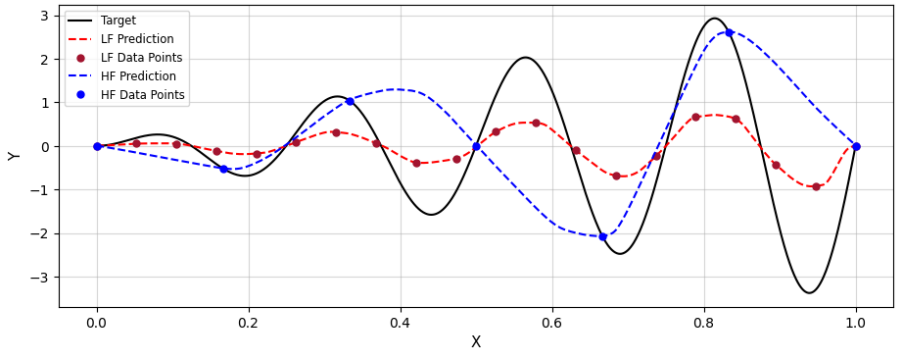
*The excess numerical examples marked by ** were found to be too similar to existing examples listed above and have been excluded from this section and moved to Appendix B.*

During performance testing, the model's prediction performances will be compared by using the numerical examples listed in Table 3 (or Appendix B) taken from (Chen, Gao, and Liu, 2022). The reader can refer to the mentioned table or appendix to examine or copy the equations of the provided LF and HF mathematical functions.

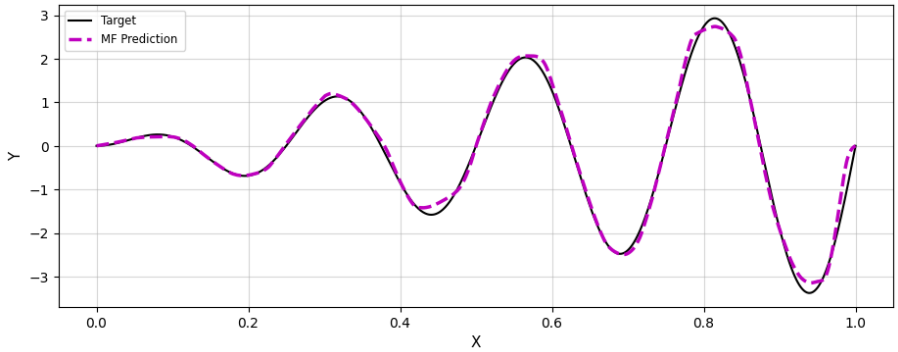
Additionally, referring to Table 4, the reader can check the hyperparameters and number of LF and HF data points used to form the models respective to their assigned numerical example to solve.

5.5.1 Basic sine waves

As previously mentioned, the comparison of the HFNN and LFNN models with their respective 7 and 20 training points are shown below.



(a) LFNN and HFNN model predictions



(b) MFNN prediction with 7 HF and 20 LF training points

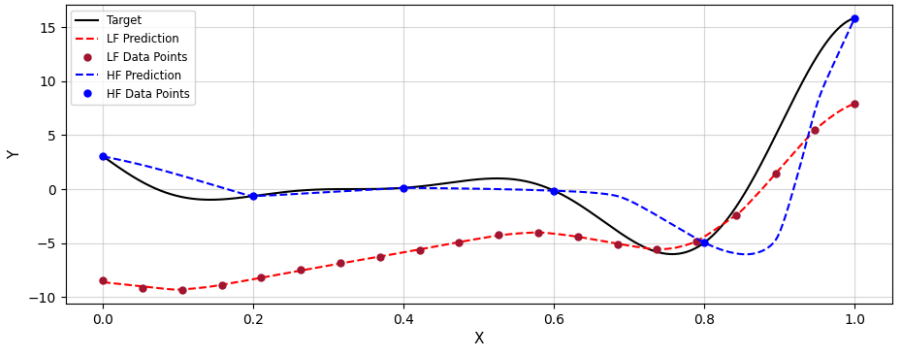
Figure 31: Sine wave predictions

$$HF(y) = 1.8 \cdot \sin(y \cdot (8\pi)) \cdot 2y \tag{5}$$

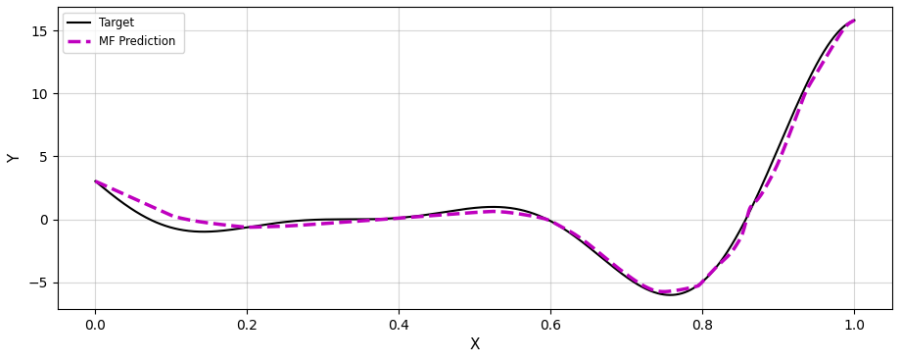
$$LF(y) = \sin(y \cdot (8\pi)) \cdot y \tag{6}$$

Figure 31a illustrates LFNN and HFNN model's predictions on the target represented by Equation (6). As can be seen, the LF model consisting of 20 training data points forms an LFNN accurate respective its own points, while completely missing the target line. On the other hand the HF model consisting of 7 training data points manages to approximately predict the target, although fails due to the lack of data points. Furthermore, Figure 31b presents the MFNN's slightly inaccurate prediction on the target line. Hence, demonstrating MFNN's prediction capabilities at a simple level and proving its potential to use cases in computer simulations by negating the reliance on expensive HF data with the utilisation of easily accessible LF data (Objective 2 and 3).

5.5.2 Continuous functions with linear relationship



(a) LFNN and HFNN model predictions

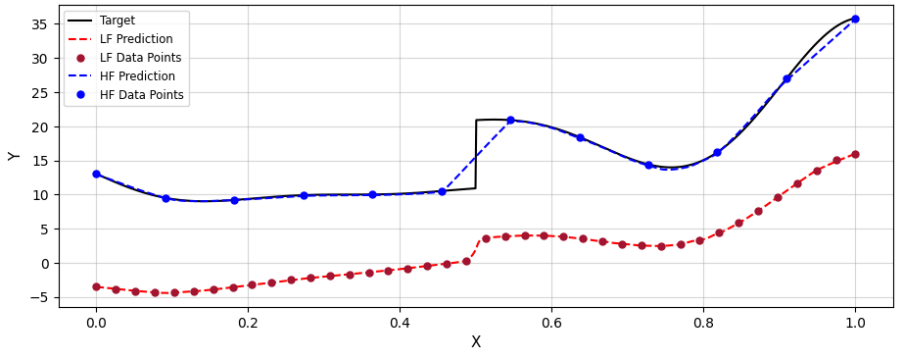


(b) MFNN prediction with 6 HF and 20 LF training data points

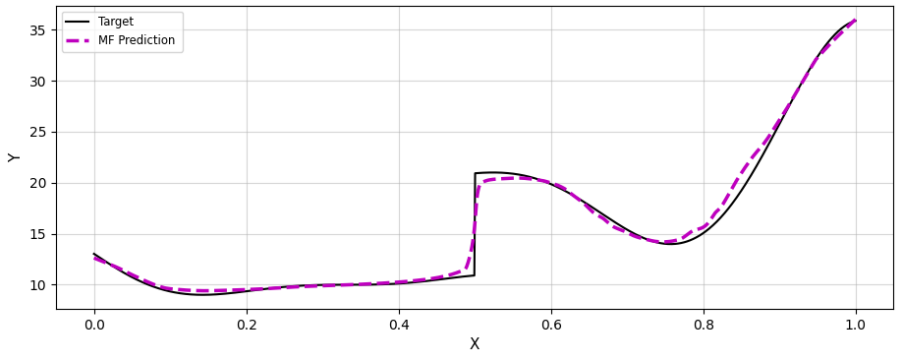
Figure 32: (a) Continuous functions with linear relationship

A similar result to 'basic sine waves' can be observed in Figure 32b, where the MFNN model successfully predicts the target compared to singular models on numerical functions getting progressively harder.

5.5.3 Discontinuous functions with linear relationship



(a) LFNN and HFNN model predictions



(b) MFNN prediction with 12 HF and 40 LF training points

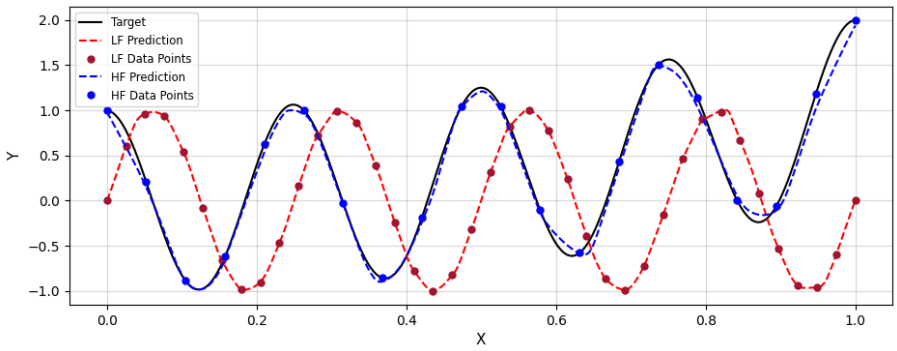
Figure 33: (b) Discontinuous functions with linear relationship

Figure 33, illustrates the comparison of LFNN, HFNN an MFNN model's prediction on a discontinuous function (No. B) taken from Table 3. The MFNN model agrees with another MFNN models result mentioned from literature at Section 3.5.4. Furthermore, the MFNN proves that it can perform significantly better in predicting the target compared to singular NN models, proving its capability of dealing with discontinuous data commonly present in majority of CFD simulations.

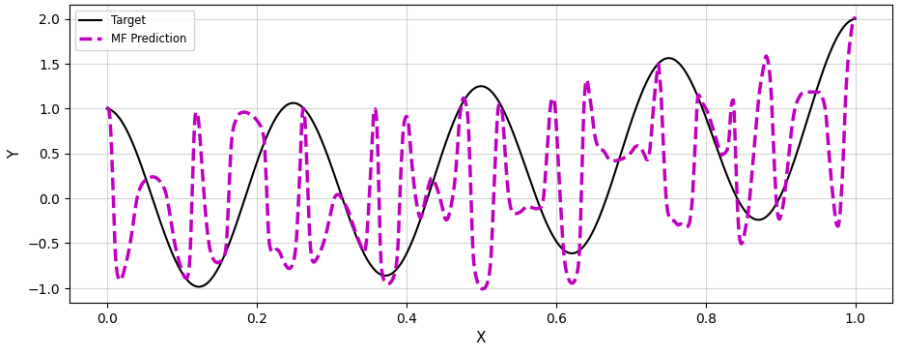
5.5.4 Embedding theory and phase shifted oscillations

Phase-shifting, which can be a common occurrence in real-world engineering applications existing between different simulations or simulation techniques. It was observed from the results that, these shifts can form model-breaking significant issues in MFNNs. Moreover, this problem and its solution was also identified by (Chen, Gao, and Liu, 2022). With the solution being the utilisation of embedding theory as previously discussed in Section 3.5.5.

Without embedding theory:



(a) LFNN and HFNN model predictions

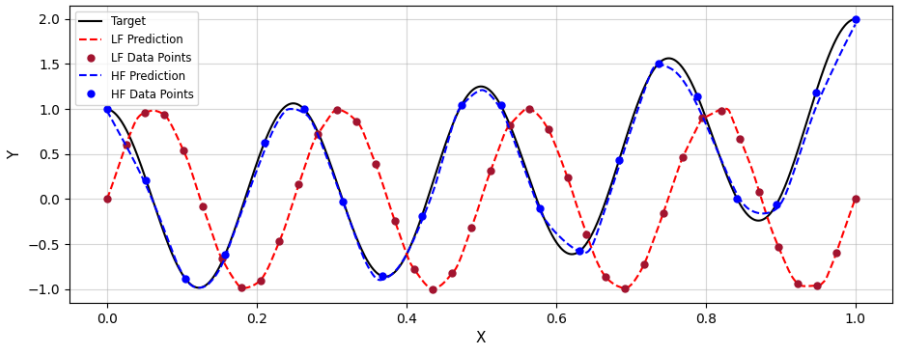


(b) MFNN prediction without no embedding

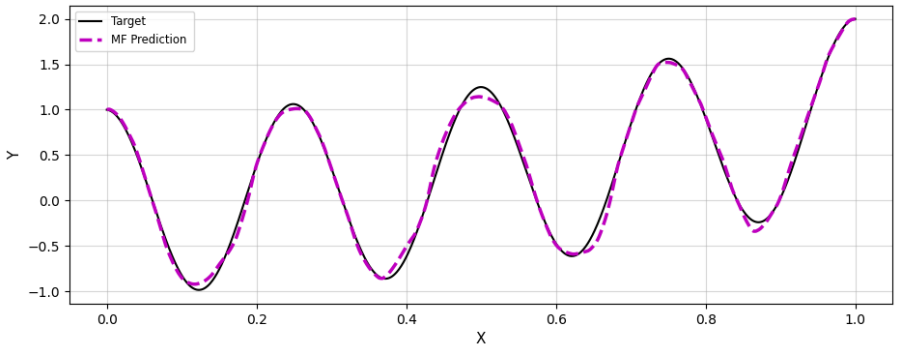
Figure 34: (e) Phase shifted oscillations without embedding theory

Referring to Figure 34b, it can be seen that the MFNN had problems at combining different fidelities of models when a Phase-Shift between the models was present in the system. This weakness can decrease the flexibility and the area of use of the MFNN model in engineering applications. Therefore, the use of techniques such as, embedding theory can significantly increase the reach of MFNNs in a practical sense, while offering a significant advantage over singular NN models.

With embedding theory:



(a) LFNN and HFNN model predictions



(b) MFNN MFNN prediction with embedding

Figure 35: (e) Phase shifted oscillations with embeddings

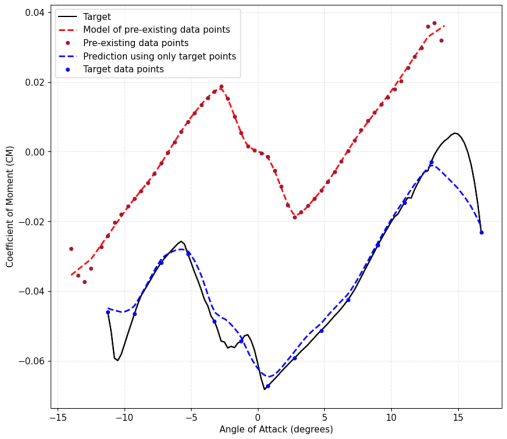
As a result, as it can be seen from Figure 35b with the implementation of embedding theory, the model can form better relationships between the 40 LF and 25 HF training data points, hence forming a significantly improved prediction for the MFNN model (Objective 4).

5.6 NACA Aerofoil predictions utilising MMNNs

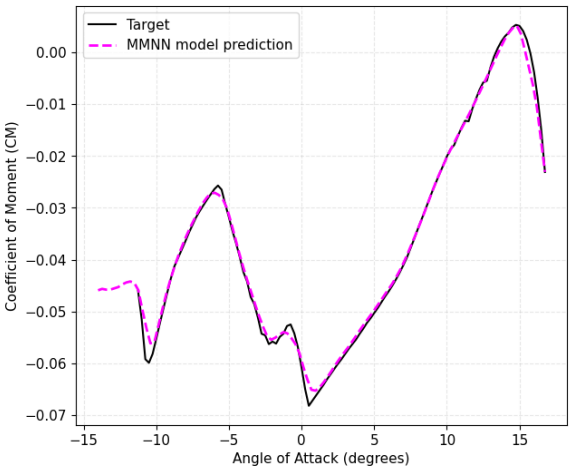
After being **validated** by the numerical examples in the previous sub-section, as previously mentioned, the MFNN model can then be used as a multi model network to process pre-existing data sets to aid in forming new predictions of new design hypotheses with less expensive data required, which offers a significant advantage in the field of engineering.

In the case of a new NACA aerofoil's analysis in an engineering setting, an array containing pre-existing NACA aerofoil aerodynamic parameters can be fed into the first network (replacing LFNN), whereas the new NACA aerofoil that is aiming to be predicted with its few existing data points (replacing HFNN), can be fed into the second network.

Hence, forming a MMNN network that can be used to accurately predict the aerodynamic parameters of the newly introduced aerofoil by utilising the pre-existing training data. Thus, keeping the strengths of both models while lessening their individual flaws.



(a) Predictions of pre-existing and target aerofoils in red and blue respectively



(b) MMNN prediction of target aerofoil (NACA 2412)

Figure 36: Coefficient of Moment over Angle of Attack predictions by the models

Figure 36 above represents a MMNN's predictions of two NACA aerofoil's Coefficients of Moment C_m against AoA. The black line indicates the target results for the reader's better understanding. Moreover, the colour red represents 56 pre-existing data points gathered from a NACA 0012 aerofoil and its model's predictions. Whereas, the colour blue represents the target NACA 2412 aerofoil's limited number of 14 data points and its model's predictions. Furthermore, Figure 36b represents the MMNN's prediction of the target NACA 2412 aerofoil's aerodynamic properties with magenta. The aim of this study is to prove MMNNs ability to form accurate predictions of the newly introduced NACA 2412 aerofoil with limited data points, with the assistance of NACA 0012's existing data points. Hence, proving the model's use in forming new predictions of a novel design with less expensive training data that has to be produced by computationally demanding simulations in engineering applications (Objective 5).

It can be seen from Figure 36a that, the singular blue NN model trained just on the target aerofoils limited data points fails to provide an accurate prediction, making significant mistakes at angles of -10, 0 and +15 degrees specifically. On the other hand, the model trained using the red pre-existing data on the existing aerofoil performs well and forms an accurate prediction passing through its own training points. As a result, by implementing the knowledge from the existing NACA 0012 aerofoil model, the MMNN can enhance the new aerofoil model's predictions to significantly increase its prediction accuracy.

As can be seen from Figure 36b, the MMNN model manages to form a way more accurate model compared to the singular NN model. Hence, proving that MMNNs can be a useful tool for looking at new aerofoil configurations based on pre-existing data. Decreasing the dependency on running new computer simulations by utilising these models instead, for the rapid gaining of approximate insight for new design hypotheses, where data is expensive to generate.

5.7 Overall discussion and critical review

Secondly, referring to the presented results and Research Questions at Section 1.3, it can be concluded that the report has successfully addressed its aim by satisfying its respective objectives and research questions. Consequently, the final MFNN/MMNN model's results has decreased the reliance of NN models on HF data by forming more accurate and flexible predictions, both on synthetic data sets and the NACA aerofoil analysis. It must be noted that model being successful on these tests does not represent its generalisability to all engineering related applications. Hence, a better practice would be to perform more tests on other various engineering related problems in the future..

Additionally, implementation of performance enhancing techniques such as batch normalisation and embedding theory shows to increase the models range of prediction especially in cases where phase-shifting was encountered such as the case in sub-section 5.5.4. Although, while these techniques show an improvement in the accuracy of the results, they might not be effective enough at solving other engineering based problems and there could be better alternatives to these discussed methods that are more suitable for MFNNs.

6 Conclusion and Limitations

6.1 Conclusion overview

As an overall and concise conclusion, the results from this report revealed the potential of MFNNs in handling varying range of problems differing in complexity in comparison to singular NN models. Hence, proving their capability of enhancing models by decreasing their dependency on HF data, and improving their precision and scalability, with all statements being backed-up by Section 5. As a result, demonstrating their potential in increasing sustainability, and driving down costs in the engineering industry and academia.

Furthermore, critical analysis made on the model have demonstrated model's adaptability to handle data sets that are discontinuous and phase shifted, which might be encountered in engineering simulations. Hence, suggesting that further improvements can be made on the model, to further enhance its potential for use in real-world engineering applications, to address challenges such as the industries high reliance on computationally demanding computer simulations.

6.2 Changes on initial aim & objectives

The key issue set out in the report and its initial aim had seen few minor changes since the project's proposal, since the aim was chosen with proper consideration of the future of the report, the changes were quite expected and were prepared for. Therefore, significant but expected changes were made, such as, removing the implementation of Bayesian Neural Networks (BNNs) and Physics-Informed Neural Networks (PINNs) from the structure of the MFNN, as they were discovered to be too complex and time demanding to implement into the model.

Additionally, the report's initial aim was to apply the MFNN model into a 2D aerofoil CFD simulation. Although, this aim was later changed to implementing the model into CFD simulations in general, allowing the MFNNs to become the main focus of the dissertation project in contrast to just aerofoil simulations. Moreover, research questions and objectives were naturally linked to the aim and minimal changes were also present on them.

6.3 Objective 1: Form and analyse NNs using PyTorch

6.3.1 Learning steps and initial optimisation

A basic Neural Network was created in Python using PyTorch ML framework with the assistance of PyTorch's documentation and guides. During the initial stages, significant problems such as the lack of understanding of the NN's structure and problem in producing results formed road blocks in the report's progress by a month long frame. In the end, a successful network was formed and the hyperparameters and the network's structure were adjusted and explored with random educational guesses. Throughout this process, a deeper understanding of theoretical and practical knowledge about the model was obtained and the first NN model was formed and slowly optimised.

6.4 Objective 2: Form and compare MFNNs to singular NNs

Building up on the knowledge gained from developing NN networks, with the guidance of the student's project supervisor and academic sources, a MFNN capable of forming accurate predictions was successfully created using PyTorch. The formation process was quite straight forward due to the background knowledge from NNs and this objective had not seen any drastic changes since the initial proposal.

6.4.1 MFNN structure and comparison to NNs

By using numerical examples resourced from the literature, the MFNN model's structure was improved and various enhancements such as implementing batch sizes into the training loop, adding batch normalisation into the layers and better data visualisation techniques such as better use of matplotlib to indicate the change in loss was implemented. Therefore, enabling performance tests to be made on the MFNN and NN models to identify their strengths and weaknesses. During these tests between the two models, the MFNN model's limit on handling phase-shifts was discovered.

6.5 Objective 3: Optimise MFNNs to decrease their HF data dependency

Similar to Objective 2, this objective had not seen any drastic changes as it was one of the main objectives of the project to decrease MFNNs reliance on HF data. Therefore, the objective was successfully completed and as illustrated before in Figure 30, the implementation of MFNNs demonstrated that they can cause a significantly decrease in the use of HF training data points during a novel design's initial testing stages (from 20 to 7 points in the case of the referred figure). As a result, highlighting the strengths of

MFNNs in reducing the dependency on HF data, to further drive downs costs compared to single fidelity NN models and standalone computer simulations.

6.5.1 MFNN performance testing

In addition to the tests made between MFNN and NN models, a specific numerical example test was also found from literature to test the MFNNs prediction capabilities and push it to its limits while comparing it to singular NN models. As expected, this sub-section had not seen any changes as it was introduced later on in the project, the testing method can be read again at Section 5.5.

6.6 Objective 4: Improve MFNNs phase-shift handling capabilities

Phase-shift handling capability of the MFNN formed a minor roadblock in the progress of the report, this objective had not significantly differed from the initial objective of increasing prediction capabilities of the MFNN and it has only gotten more specific as the project progressed.

As a result, the phase-shift handling capability was successfully implemented into the model with the use of embedding theory, which played a significant role in the production of the final optimised model. This improvement significantly improved the results obtained from the MFNN as the model formed a stronger relationship between the LF and HF data. Thus, resulting in increased generalisability and accuracy of the model. Additionally, similar to the sub-section above, this section was introduced later on into the project.

6.7 Objective 5: Illustrate MFNNs use in real-world engineering applications

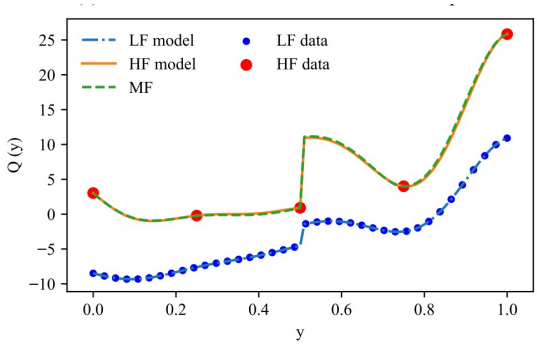
During the dissertation project, major issues were encountered in completing this objective and multiple changes were made on it. The main key issue being the broad nature of the dissertation topic, as it offered multiple pathways to follow through. These paths included, focusing on optimising 2D aerofoil CFD simulations, testing different techniques such as BNNs or PINNs into the MFNN model. As a result, due to these option's unknown difficulty and challenges to the student, a time frame was existent where the objectives were not fully determined by the student.

As a result, later on in the project, a more realistic path of optimising CFD simulations results with the use of MFNNs was chosen and was successfully achieved with the NACA

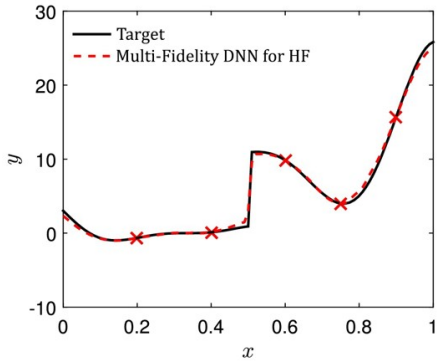
aerofoil study in Section 5.6 to prove its potential of implementation into real-world engineering applications.

6.8 Comparison of literature against findings

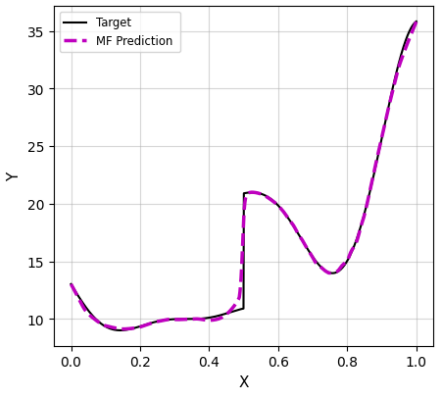
Overall, the literature mentioned about NNs and MFNNs have agreed with the project's results. A significant indication being the report MFNN developed by the student, agreeing with the numerical examples tested by both (Chen, Gao, and Liu, 2022) and (Meng and Karniadakis, 2020) on the same (b) Discontinuous functions with linear relationship example, in Section 5.5.



(a) Prediction of (b) by (Chen, Gao, and Liu, 2022)



(b) Prediction of (b) by (Meng and Karniadakis, 2020)



(c) Prediction of (b) by this report

Figure 37: Comparison of literature and report's MFNN models

As illustrated in Figure 37, this report's predictions in general strongly correlates with the two literature cases' predictions of the exact same function (b) and the rest of the numerical examples. Proving the accuracy of this report's with two separate literature review cases. Additionally, the comparison of this report's all numerical example results mentioned in Section 5.5 and Appendix B can be compared to the literature cases by checking the referenced literature cases.

Although, it is crucial to point out that the quantity and position of the HF data points used vary between the two literature cases. As previously mentioned, this report is trained in a similar fashion and configuration to (Chen, Gao, and Liu, 2022).

6.9 Relevance and impact

6.9.1 Possible influences of MFNNs on industry and research

As mention multiple times through the report, MFNN models play a significant role in leading the way in making Research and Development more affordable and common in the industry and academia, due to its unique characteristic of utilising cheap and easily attainable LF data to aid the expensive and power demanding HF data. Hence, decreasing the reliance on these expensive data sets, making it possible to run simulations for a shorter period of time, without sacrificing in the model's accuracy and detail. Hence, introducing a new way of, decreasing the energy and time spent during the testing of novel design hypotheses and increasing efficiency.

Moreover, enhancing and potentially implementing this model into the industry can make a significant difference to the conventional methods of utilising computer simulations. As this model can shorten the require simulation time and potentially offer cost and time savings in an example engineering field, where time is of utmost of importance.

6.9.2 Environmental implications of MFNNs

Utilising this MFNN model in academia or especially industry, the Research and Development process of novel design hypotheses can significantly be affected in a positive manner. As these models can improve the sustainability in the field of engineering by providing quicker, cheaper, hence more efficient solutions. In comparison to running computationally demanding simulations which negatively impact the environment and sustainability with excess power and energy use.

6.10 Limitations

Since, MFNN is branch in ML that has just gained popularity in the recent years, the main limitation encountered during the dissertation project was the research being mostly limited to synthetic data sets and pre-processed data taken from literature not suitable for the report.

As a result, literature such as "Multi-Fidelity Surrogate Modelling of Wall Mounted Cube" (Mole, Skillen, and Revell, 2020) providing an amazing knowledge about the MFNN model was the only available case that the offered potential data and results, that could

be tried to be replicated by the student's own MFNN model.

Although, the (Mole, Skillen, and Revell, 2020)'s study proved to be too complicated for the student, due to there being not enough time to learn pre-process raw data obtained from CFD simulation models, due to the student's lack of background on modelling and simulation. Additionally, the lack of student's knowledge on modelling & simulation had also prevented the student from running their own simulations using XFoil and Star-CMM+ on a 2D aerofoil to form two fidelities of data sets. The student was capable of extracting data from the STAR-CMM+ and Xfoil simulation however, idea was scratched later on due to the lack of simulation's validation that had to be done to be presented in the report.

In addition to this, another further objective was to implement PINNs into the model, which proved to be too hard to implement on top of working with MFNNs in the report. Later on it was discovered by the student that PINNs were another separate dissertation topic provided by the university, proving how unrealistic that goal was.

6.11 Recommendations and future work

The main future work that is planned to be done is continuing working on implementing the CFD simulation data from (Mole, Skillen, and Revell, 2020)'s paper into the MFNN, to compare the model's performance to the literature's. As the results obtained by the MFNN model developed for this report has high potential considering its prediction capabilities, and room for improvements such as implementing a CNN layer through the network to increase its relationship forming capabilities as discussed in Section 3.5.5, on the literature published by (Chen, Gao, and Liu, 2022).

As discussed in the limitations section, MFNN is a branch in ML that has gained popularity in the recent years. From the research that was done on this topic, MFNNs have a high potential to be further improved with more sophisticated and interesting research to deal with the current limitations of the model. These enhancements also include, implementing PINNs, which was one of the original objectives of this project.

Since it was one of the original objectives, the student had went through the literature of the PINNs through a period of time during the project and did their own research (illustrated in finalised Gantt chart project plan), and managed to replicate a simple "2D blood flow in a stenosis" PINN model from (Arzani, Wang, and D'Souza, 2021). Hence, making PINNs an appropriate target for the student's future work thorough their academic or work life, as its a skill that has the potential to have its benefits in multiple branches.

Lastly, the final MFNN model took around approximately 20 Machine Learning models to build up to, including analysing and writing Sci-Kit, GPyTorch models and various PyTorch functions, to build a rich background of ML modelling and most importantly theory. A strong recommendation would be to, go in a progressive, step by step fashion to build up to the goal. In this report's case by the time of its submission, the next step is to work to implement the MFNN model into CFD models, and most likely to be followed by PINNs.

The student can be contacted any time to provide the GitHub workspace for the dissertation project, more information is provided in Appendix B.

7 References

Altun, H, A Bilgil, and B C Fidan (2007). "Treatment of multi-dimensional data to enhance neural network estimators in regression problems". In: *Expert Systems with Applications* 32 (2), pp. 599–605. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2006.01.054. URL: https://www.sciencedirect.com/science/article/pii/S095741740600056X.

Arzani, Amirhossein, Jian-Xun Wang, and Roshan M. D'Souza (July 2021). "Uncovering near-wall blood flow from sparse data with physics-informed neural networks". In: *Physics of Fluids* 33 (7), p. 071905. ISSN: 1070-6631. DOI: 10.1063/5.0055600.

Bakar, Abu et al. (2022). "Multi-Objective Optimization of Low Reynolds Number Airfoil Using Convolutional Neural Network and Non-Dominated Sorting Genetic Algorithm". In: *Aerospace* 9 (1). ISSN: 2226-4310. DOI: 10.3390/aerospace9010035. URL: https://www.mdpi.com/2226-4310/9/1/35.

Brownlee, Jason (2018). "What is the Difference Between a Batch and an Epoch in a Neural Network". In: *Machine Learning Mastery* 20.

Burkov, Book Andriy (2019). *The Hundred-Page Machine Learning*.

Chen, Jie, Yi Gao, and Yongming Liu (Mar. 2022). "Multi-fidelity Data Aggregation using Convolutional Neural Networks". In: *Computer Methods in Applied Mechanics and Engineering* 391. ISSN: 00457825. DOI: 10.1016/j.cma.2021.114490.

Cilimkovic, Mirza (2010). "Neural Networks And Back Propagation Algorithm". In.

Ding, Shifei, Chunyang Su, and Junzhao Yu (Aug. 2011). "An optimizing BP neural network algorithm based on genetic algorithm". In: *Artificial Intelligence Review* 36 (2), pp. 153–162. ISSN: 0269-2821. DOI: 10.1007/s10462-011-9208-z.

Feng, Jianli and Shengnan Lu (June 2019). "Performance Analysis of Various Activation Functions in Artificial Neural Networks". In: *Journal of Physics: Conference Series* 1237 (2), p. 22030. DOI: 10.1088/1742-6596/1237/2/022030. URL: https://dx.doi.org/10.1088/1742-6596/1237/2/022030.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2017). *Deep learning*. ISBN: 9780262035613 0262035618. URL: https://www.worldcat.org/title/deep-learning/oclc/985397543&referer=brief_results.

Grigoryan, Davit (2021). *Activation Functions in Neural Networks*. Accessed: 18 April 2023. URL: https://www.superannotate.com/blog/activation-functions-in-neural-networks.

Haykin, Simon (1998). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

Hecht-Nielsen, Robert (1992). *III.3 - Theory of the Backpropagation Neural Network*. Ed. by Harry Wechsler. DOI: https://doi.org/10.1016/B978-0-12-741252-8.50010-8. URL: https://www.sciencedirect.com/science/article/pii/B9780127412528500108.

Kalder, Gaudenz (2023). *drawio*. Accessed: 16 April 2023. URL: diagrams.net.

Karpathy, Andrej, Justin Johnson, and Li Fei-Fei (2016). *CS231n: Convolutional Neural Networks for Visual Recognition - Optimization*. Accessed: 14 April 2023. URL: https://cs231n.github.io/optimization-1/.

Lawton, J. (2022). *Where are you on the dunning-kruger wiggle?* Accessed: 2 May 2023. URL: https://www.trainingpeaks.com/coach-blog/where-are-you-on-the-dunning-kruger-wiggle/.

Lecun, Y et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86 (11), pp. 2278–2324. DOI: 10.1109/5.726791.

LeCun, Yann, Y Bengio, and Geoffrey Hinton (Apr. 2015). "Deep Learning". In: *Nature* 521, pp. 436–444. DOI: 10.1038/nature14539.

Lenail, Alex (2018). *NN-SVG: Publication-Ready Neural Network Architecture Schematics*. Accessed: 13 April 2023. URL: https://github.com/alexlenail/NN-SVG.

Meng, Xuhui and George Em Karniadakis (Jan. 2020). "A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems". In: *Journal of Computational Physics* 401. ISSN: 10902716. DOI: 10.1016/j.jcp.2019.109020.

Mole, Andrew, Alex Skillen, and Alistair Revell (2020). *Multi-Fidelity Surrogate Modelling of Wall Mounted Cubes*.

Pedregosa, Fabian et al. (2011). "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct, pp. 2825–2830.

Peter, Jacques and Meryem Marcelet (Apr. 2008). "Comparison of surrogate models for turbomachinery design". In: *WSEAS Transactions on Fluid Mechanics* 3.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747. URL: http://arxiv.org/abs/1609.04747.

Singh, Anand Pratap, Shivaji Medida, and Karthik Duraisamy (2017). "Machine-Learning-Augmented Predictive Modeling of Turbulent Separated Flows over Airfoils".

In: *AIAA Journal* 55 (7), pp. 2215–2227. DOI: 10.2514/1.J055595. URL: https://doi.org/10.2514/1.J055595.

Vinuesa, Ricardo and Steven L. Brunton (Oct. 2021). "Enhancing Computational Fluid Dynamics with Machine Learning". In: DOI: 10.1038/s43588-022-00264-7. URL: http://arxiv.org/abs/2110.02085%20http://dx.doi.org/10.1038/s43588-022-00264-7.

Waldrop, M. Mitchell (Feb. 2016). "The chips are down for Moore's law". In: *Nature* 530 (7589), pp. 144–147. ISSN: 0028-0836. DOI: 10.1038/530144a.

8 Appendix A - Project Management

8.1 Initial project plan

No Gantt charts were provided by the student during the formative initial project plan stage. Although, the initial project plan submitted with the title of "Deep Learning Multi-Fidelity Neural Network's Use in Simulation", had a proper motivation that covered the expense of HF data points. With the goal of implementing two different fidelities to form a MFNN to optimise aerofoil CFD simulation results, consisting of Xfoil and detail CFD simulation that was going to be determined later.

Moreover, the given generalised motivation was logical and technical enough for an experienced reader from the field to understand. Although, the motivation had lacked the clarity for a new reader to fully grasp what they would read.

Additionally, the initial project plan included broad objectives such as:

1. Utilizing Python and PyTorch to form a multi-fidelity neural network to analyse and improve the prediction of flow characteristics of 2D aerofoils with changing parameters.

2. Improve accuracy of surrogate models that can be acquired using multi-fidelity modelling compared to single fidelity modelling.

3. Prove that a trained model can predict results with high accuracy at locations in the parameter space away from the trained data set.

4. Produce high and low fidelity models utilizing CFD (preferably ANSYS) and XFoil for the initial training procedure of the data-sets.

5. Produce a 3D aerofoil model building up on the model and data collected from the 2D aerofoil.

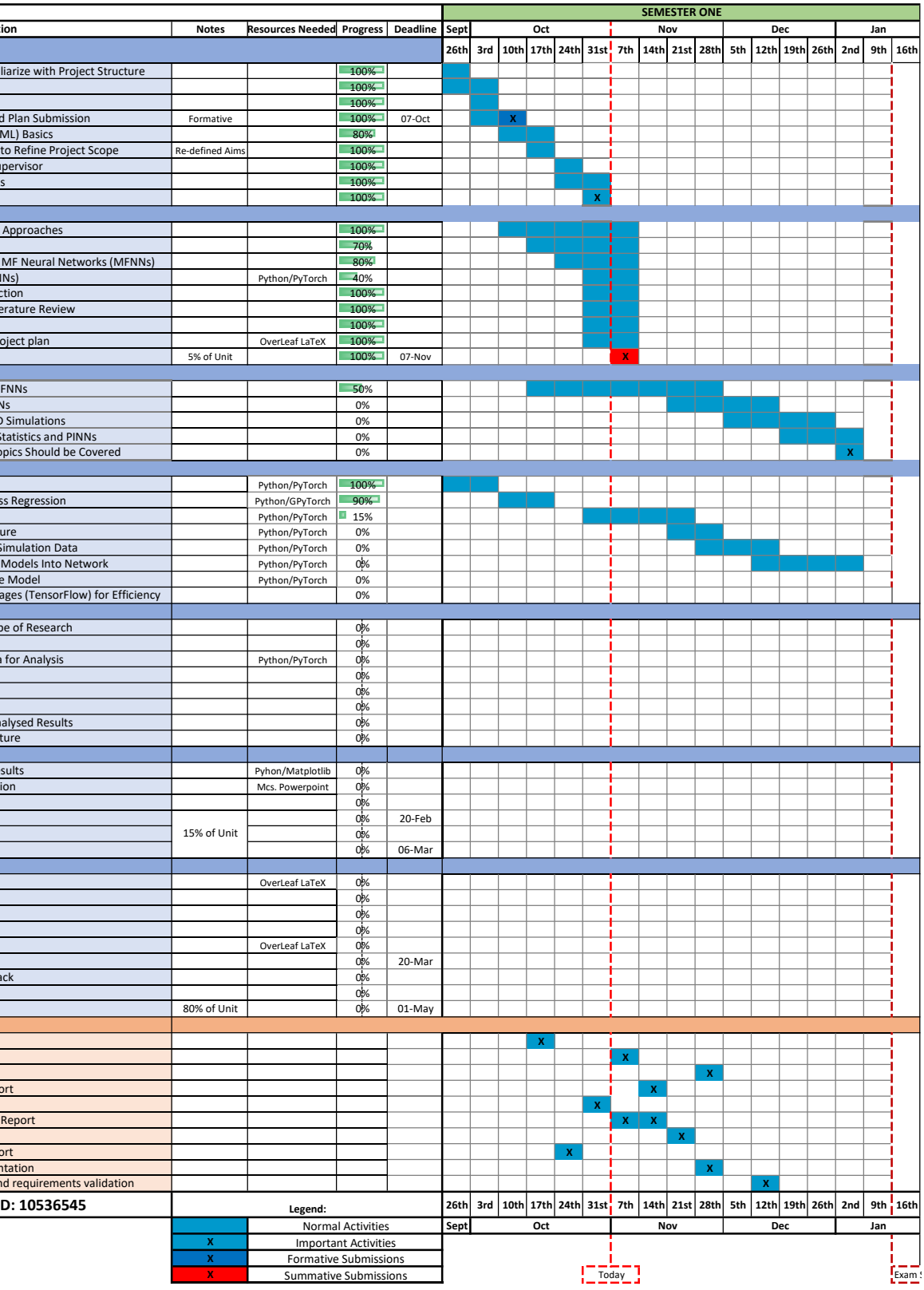
As the dissertation project progressed, it can be observed the last objective 5 was clearly pushing the limits of the project. Although, the broad but logical initial objectives 1 to 4 were still properly introduced.

8.2 Project plan during project proposal

A well project plan greatly appreciated by the student's supervisor was prepared and submitted. In the plan, stage had shown a significant progress, where a proper Gantt chart indicating the student's path was formed. As discussed, this project plan formed

a great and properly formed foundation to follow throughout the dissertation project, helping the student prioritise tasks and be aware of upcoming project related and non-related deadlines.

ID	Description	Notes	Resources Needed	Progress	Deadline
1	**Initial Project Scope Phase**				
1.1	Initial Meeting with Supervisor to Familiarize with Project Structure			100%	
1.2	Research Initial Aims and Objectives			100%	
1.3	Construct Initial Plan			100%	
1.4	Initial Project Objectives/Questions and Plan Submission	Formative		100%	07-Oct
1.5	Read Literature on Machine Learning (ML) Basics			80%	
1.6	Unscheduled Meeting with Supervisor to Refine Project Scope	Re-defined Aims		100%	
1.7	Discuss Possible Topics of Aims with Supervisor			100%	
1.8	Review Initial Plan, Aims and Objectives			100%	
1.9	Finalise Initial Project Scope Phase			100%	
2	**Project Proposal and Plan Phase**				
2.1	Explore Applicable Methodologies and Approaches			100%	
2.2	Read Literature on Statistics and ML			70%	
2.3	Read Literature on Neural Networks & MF Neural Networks (MFNNs)			80%	
2.4	Practice Coding on Neural Networks (NNs)		Python/PyTorch	40%	
2.5	Plan Structure of Abstract and Introduction			100%	
2.6	Plan Structure of Methodology and Literature Review			100%	
2.7	Plan Structure of Project Plan			100%	
2.8	Write Project Proposal and Detailed Project plan		OverLeaf LaTeX	100%	
2.9	Project Proposal and Plan Submission	5% of Unit		100%	07-Nov
3	**Research and Literature Review**				
3.1	In-depth Research on Application of MFNNs			50%	
3.2	Read In-depth Literature on ML and NNs			0%	
3.3	Research on Application of NNs on CFD Simulations			0%	
3.4	Read In-depth Literature on Bayesian Statistics and PINNs			0%	
3.5	Discuss with Supervisor if Any Other Topics Should be Covered			0%	
4	**Model Implementation and Coding**				
4.1	Set Python and Required Packages Up		Python/PyTorch	100%	
4.2	Practice MF Coding on Gaussian Process Regression		Python/GPyTorch	90%	
4.3	Form a Preliminary Neural Network		Python/PyTorch	15%	
4.4	Reproduce Results From MFNN Literature		Python/PyTorch	0%	
4.5	Produce Own Models From Gathered Simulation Data		Python/PyTorch	0%	
4.6	Implementation of Bayesian and PINN Models Into Network		Python/PyTorch	0%	
4.7	Work on Refining and Improving on the Model		Python/PyTorch	0%	
4.8	Consider Switching to Alternative Packages (TensorFlow) for Efficiency			0%	
5	**Analysis**				
5.1	Refine Approaches to Confirm the Scope of Research			0%	
5.2	Form Criterias for Hypothesis Testing			0%	
5.3	Document and Pre-Process Result Data for Analysis		Python/PyTorch	0%	
5.4	Perform Analysis			0%	
5.5	Compare Results of Various Models			0%	
5.6	Test Hypothesis with Model Results			0%	
5.7	Search for Areas of Improvement in Analysed Results			0%	
5.8	Compared With Already Existing Literature			0%	
6	**Poster**				
6.1	Plot Illustrative Graphs From Model Results		Pyhon/Matplotlib	0%	
6.2	Integrate Text and Additional Information		Mcs. Powerpoint	0%	
6.3	Review Layout of Poster			0%	
6.4	Poster Submission			0%	20-Feb
6.5	Prepare for Poster Presentation	15% of Unit		0%	
6.6	Poster Presentation			0%	06-Mar
7	**Final Report Writing**				
7.1	Form Final Report Layout		OverLeaf LaTeX	0%	
7.2	Update Methodology and Findings			0%	
7.3	Discuss Results and Findings			0%	
7.4	Update Abstract			0%	
7.5	Proof Read and Check References		OverLeaf LaTeX	0%	
7.6	Sample Writing Submission			0%	20-Mar
7.7	Fix Mistakes Pointed Out In the Feedback			0%	
7.8	Perform Final Updates on the Report			0%	
7.9	Report Submission	80% of Unit		0%	01-May
8	**Activities In Other Units**				
-	Aircraft Aerodynamics Quiz 1				
-	Aircraft Aerodynamics Quiz 2				
-	Aircraft Aerodynamics Quiz 3				
-	Aircraft Aerodynamics Laboratory Report				
-	Control Engineering Coursework				
-	Vibrations & Aeroelasticity Laboratory Report				
-	Operations Management Courswork				
-	CASD Initial System Requirements Report				
-	CASD Design Review Tradeshow Presentation				
-	CASD Final concept summary report and requirements validation				



Emrecan Serin, ID: 10536545

Legend:

	Normal Activities
X	Important Activities
X	Formative Submissions
X	Summative Submissions

Figure 38: Gantt chart - project plan during project proposal stage (1 of 2)

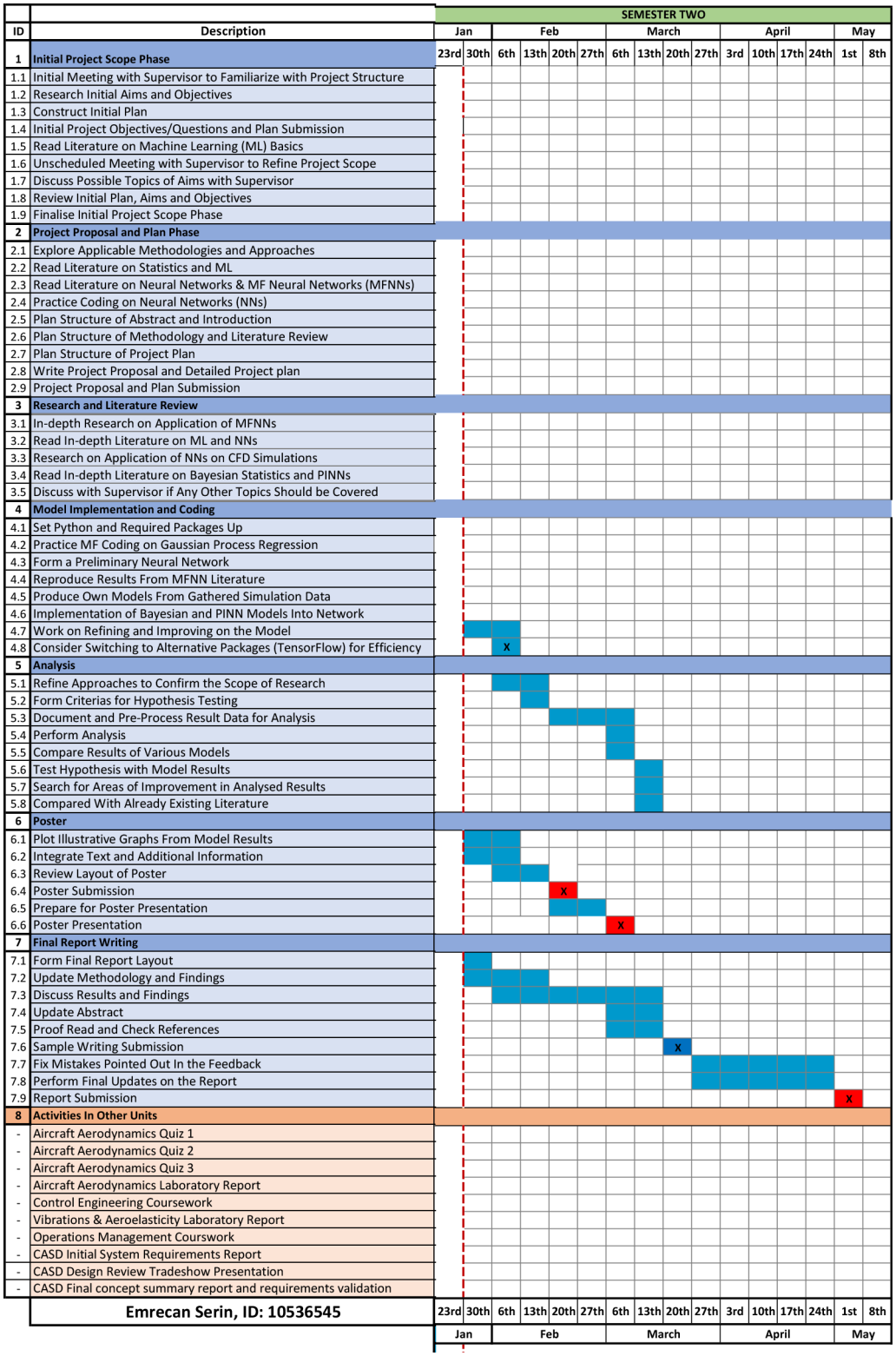


Figure 39: Gantt chart - project plan during project proposal stage (2 of 2)

8.3 Updated project plan at the start of semester 2

Referring to the Gantt chart for semester 2 in Figure 40, the major changes made to the previous plan was the exclusion of implementing BNNs and PINNs as discussed in conclusion Section 6 and decision to keep on working with PyTorch instead of TensorFlow framework, the rest of the plan followed accordingly. To add furhter comments about these decisions, the decision to remove the extra implementations such as BNNs and PINNs was appropriate choice to make, as by semester 2, the student had realised how difficult the core of the project which was MFNNs. Hence, it was decided to focus the attention onto MFNNs. Additionally, the switch to TensorFlow from PyTorch had not been made, as the student was already familiar to the framework and also listened to the helpful advice of sticking to PyTorch from their supervisor.

Moreover, referring to the Gantt chart, significant progress had been made on the project during the time and the student was well on track with the project, where the figures and code was already prepared for the poster presentation, with the poster itself on preparation, a healthy time before its deadline.

Lastly, the poster presentation is illustrated in Figure 42 to provide further context to the reader on how the project has progressed, specifically the future work section.

ID	Description	Notes	Resources Needed	Progress	Deadline	Sept	Oct				
1	**Initial Project Scope Phase**					26th	3rd	10th	17th	24th	31st
1.1	Initial Meeting with Supervisor to Familiarize with Project Structure			100%							
1.2	Research Initial Aims and Objectives			100%							
1.3	Construct Initial Plan			100%							
1.4	Initial Project Objectives/Questions and Plan Submission	Formative		100%	07-Oct			X			
1.5	Read Literature on Machine Learning (ML) Basics			100%							
1.6	Unscheduled Meeting with Supervisor to Refine Project Scope	Re-defined Aims		100%							
1.7	Discuss Possible Topics of Aims with Supervisor			100%							
1.8	Review Initial Plan, Aims and Objectives			100%							
1.9	Finalise Initial Project Scope Phase			100%							X
2	**Project Proposal and Plan Phase**										
2.1	Explore Applicable Methodologies and Approaches			100%							
2.2	Read Literature on Statistics and ML			100%							
2.3	Read Literature on Neural Networks & MF Neural Networks (MFNNs)			100%							
2.4	Practice Coding on Neural Networks (NNs)		Python/PyTorch	100%							
2.5	Plan Structure of Abstract and Introduction			100%							
2.6	Plan Structure of Methodology and Literature Review			100%							
2.7	Plan Structure of Project Plan			100%							
2.8	Write Project Proposal and Detailed Project plan		OverLeaf LaTeX	100%							
2.9	Project Proposal and Plan Submission	5% of Unit		100%	07-Nov						
3	**Methodology Research and Literature Review**										
3.1	In-depth Research on Application of MFNNs			100%							
3.2	Read In-depth Literature on ML and NNs			100%							
3.3	Research on Application of NNs on CFD Simulations			100%							
3.4	Read In-depth Literature on Bayesian Statistics and PINNs			100%							
3.5	Discuss with Supervisor if Any Other Topics Should be Covered			100%							
4	**Model Implementation and Coding**										
4.1	Set Python and Required Packages Up		Python/PyTorch	100%							
4.2	Practice MF Coding on Gaussian Process Regression		Python/GPyTorch	100%							
4.3	Form a Preliminary Neural Network		Python/PyTorch	100%							
4.4	Reproduce Results From MFNN Literature		Python/PyTorch	100%							
4.5	Produce Own Models From Gathered Simulation Data		Python/PyTorch	100%							
4.6	Implementation of Bayesian and PINN Models Into Network		Python/PyTorch	0%							
4.7	Work on Refining and Improving on the Model		Python/PyTorch	100%							
4.8	Consider Switching to Alternative Packages (TensorFlow) for Efficiency			0%							
5	**Analysis**										
5.1	Refine Approaches to Confirm the Scope of Research			0%							
5.2	Form Criterias for Hypothesis Testing			0%							
5.3	Document and Pre-Process Result Data for Analysis		Python/PyTorch	0%							
5.4	Perform Analysis			0%							
5.5	Compare Results of Various Models			0%							
5.6	Test Hypothesis with Model Results			0%							
5.7	Search for Areas of Improvement in Analysed Results			0%							
5.8	Compared With Already Existing Literature			0%							
6	**Poster**										
6.1	Plot Illustrative Graphs From Model Results		Pyhon/Matplotlib	100%							
6.2	Integrate Text and Additional Information		Mcs. Powerpoint	50%							
6.3	Review Layout of Poster			25%							
6.4	Poster Submission			0%	20-Feb						
6.5	Prepare for Poster Presentation	15% of Unit		5%							
6.6	Poster Presentation			0%	06-Mar						
7	**Final Report Writing**										
7.1	Form Final Report Layout		OverLeaf LaTeX	0%							
7.2	Update Methodology and Findings			0%							
7.3	Discuss Results and Findings			0%							
7.4	Update Abstract			0%							
7.5	Proof Read and Check References		OverLeaf LaTeX	0%							
7.6	Sample Writing Submission			0%	20-Mar						
7.7	Fix Mistakes Pointed Out In the Feedback			0%							
7.8	Perform Final Updates on the Report			0%							
7.9	Report Submission	80% of Unit		0%	04-May						
8	**Activities In Other Units**										
-	Aircraft Aerodynamics Quiz 1								X		
-	Aircraft Aerodynamics Quiz 2										
-	Aircraft Aerodynamics Quiz 3										
-	Aircraft Aerodynamics Laboratory Report										
-	Control Engineering Coursework										X
-	Vibrations & Aeroelasticity Laboratory Report										
-	Operations Management Courswork										
-	CASD Initial System Requirements Report									X	
-	Heat Transfer Coursework 1										
-	Flight Dynamics Coursework 1										
-	Aerospace Propulsion CW										
-	Structures 3 CW										
-	Flight Dynamics Coursework 2										

Emrecan Serin, ID: 10536545

Legend:	
	Normal Activities
X	Important Activities
X	Formative Submissions
X	Summative Submissions

26th	3rd	10th	17th	24th	31st
Sept	Oct				

Toc

Figure 40: Project plan at the start of semester 2 (1 of 2)

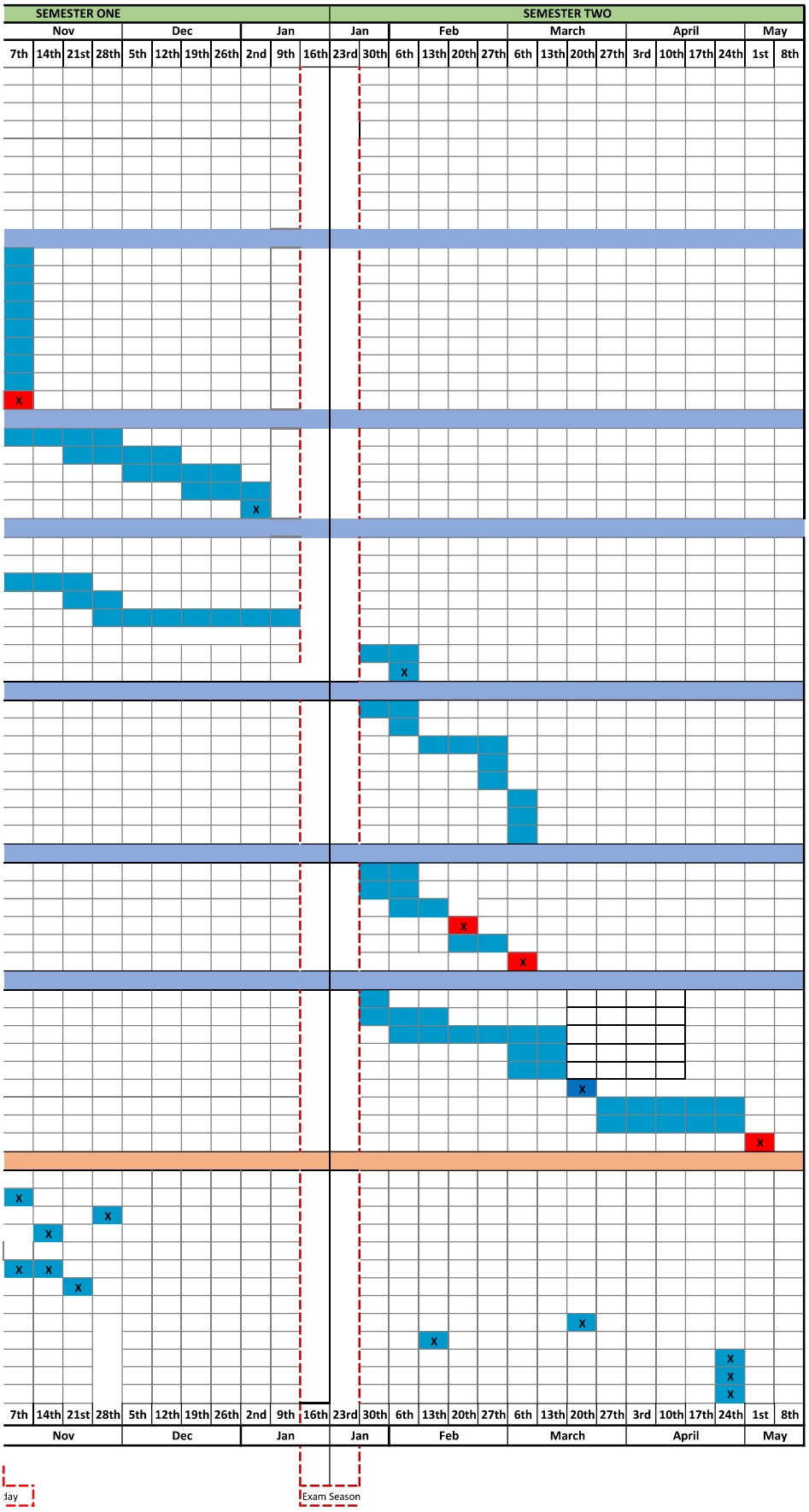


Figure 41: Project plan at the start of semester 2 (2 of 2)

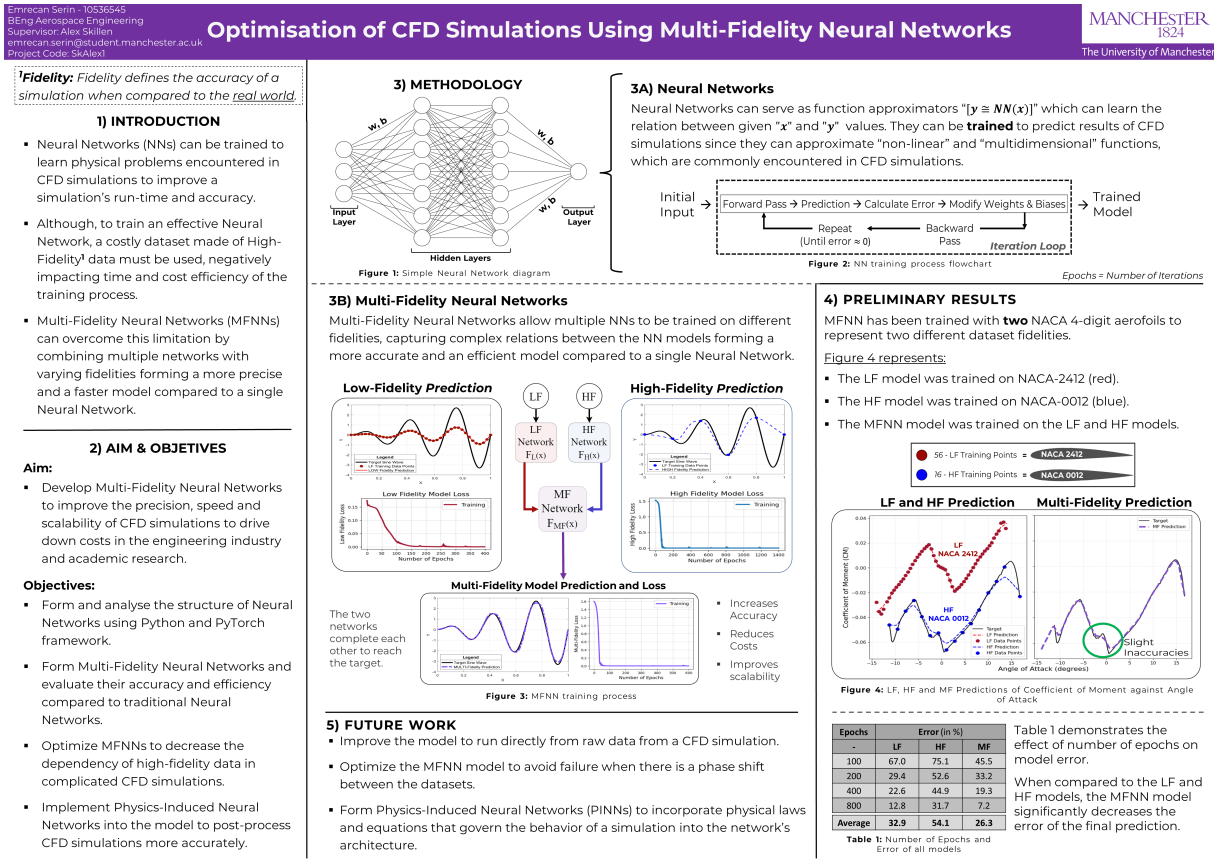
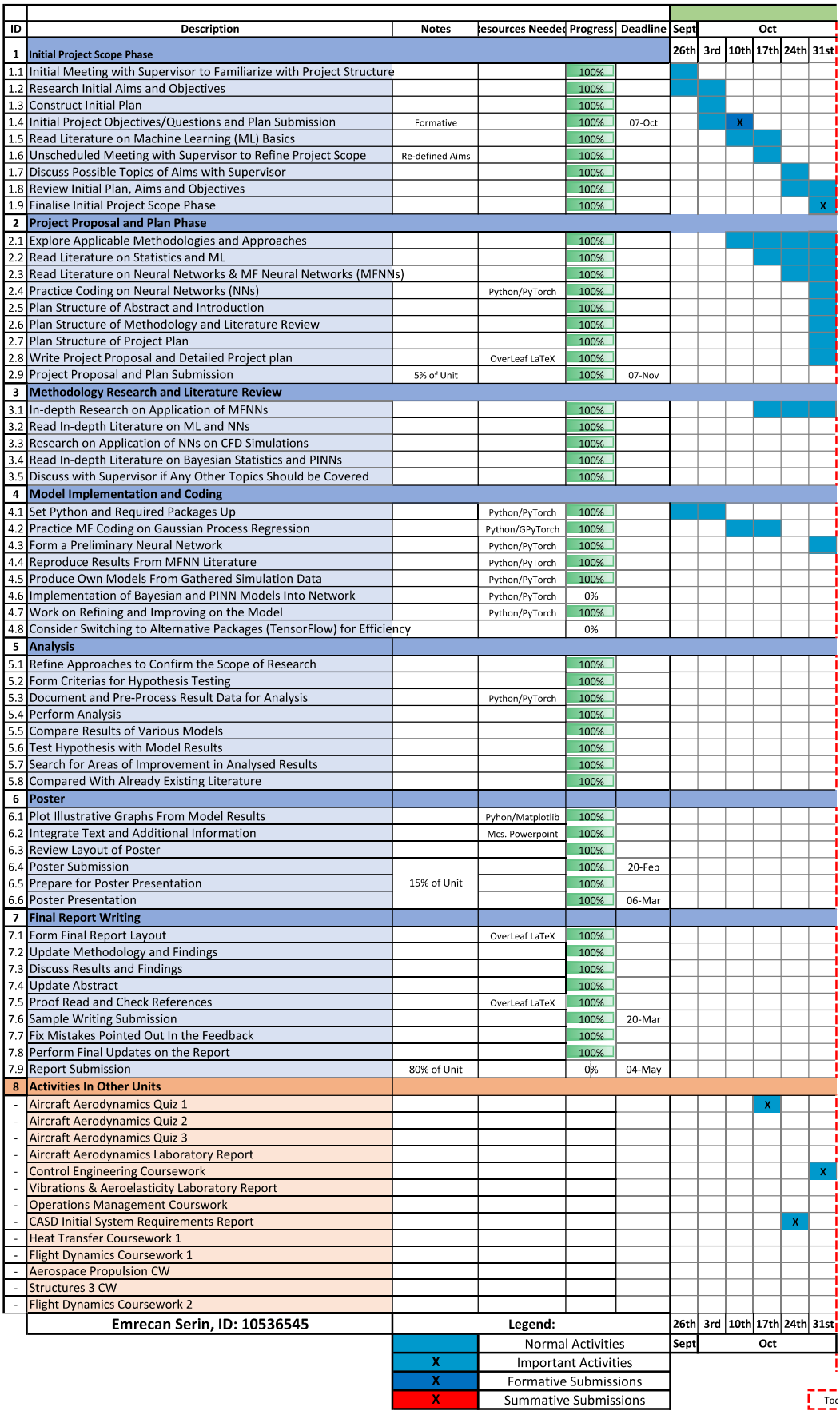


Figure 42: Project's poster from the beginning of semester 2

8.4 Finalised project plan

As can be seen from the Gantt chart, majority of the tasks were fully completed, excluding the already mentioned BNNs, PINNs and the switch to TensorFlow. As a result, it can be determined from the finalised project plan that the targeted aim have successfully been reached and all slightly reconfigured objectives have been reached by the student. No major changes were made on the plan as it was satisfactory enough in addition to the student's progress.



ID	Description	Notes	Resources Needed	Progress	Deadline
1	**Initial Project Scope Phase**				
1.1	Initial Meeting with Supervisor to Familiarize with Project Structure			100%	
1.2	Research Initial Aims and Objectives			100%	
1.3	Construct Initial Plan			100%	
1.4	Initial Project Objectives/Questions and Plan Submission	Formative		100%	07-Oct
1.5	Read Literature on Machine Learning (ML) Basics			100%	
1.6	Unscheduled Meeting with Supervisor to Refine Project Scope	Re-defined Aims		100%	
1.7	Discuss Possible Topics of Aims with Supervisor			100%	
1.8	Review Initial Plan, Aims and Objectives			100%	
1.9	Finalise Initial Project Scope Phase			100%	
2	**Project Proposal and Plan Phase**				
2.1	Explore Applicable Methodologies and Approaches			100%	
2.2	Read Literature on Statistics and ML			100%	
2.3	Read Literature on Neural Networks & MF Neural Networks (MFNNs)			100%	
2.4	Practice Coding on Neural Networks (NNs)		Python/PyTorch	100%	
2.5	Plan Structure of Abstract and Introduction			100%	
2.6	Plan Structure of Methodology and Literature Review			100%	
2.7	Plan Structure of Project Plan			100%	
2.8	Write Project Proposal and Detailed Project plan		OverLeaf LaTeX	100%	
2.9	Project Proposal and Plan Submission	5% of Unit		100%	07-Nov
3	**Methodology Research and Literature Review**				
3.1	In-depth Research on Application of MFNNs			100%	
3.2	Read In-depth Literature on ML and NNs			100%	
3.3	Research on Application of NNs on CFD Simulations			100%	
3.4	Read In-depth Literature on Bayesian Statistics and PINNs			100%	
3.5	Discuss with Supervisor if Any Other Topics Should be Covered			100%	
4	**Model Implementation and Coding**				
4.1	Set Python and Required Packages Up		Python/PyTorch	100%	
4.2	Practice MF Coding on Gaussian Process Regression		Python/GPyTorch	100%	
4.3	Form a Preliminary Neural Network		Python/PyTorch	100%	
4.4	Reproduce Results From MFNN Literature		Python/PyTorch	100%	
4.5	Produce Own Models From Gathered Simulation Data		Python/PyTorch	100%	
4.6	Implementation of Bayesian and PINN Models Into Network		Python/PyTorch	0%	
4.7	Work on Refining and Improving on the Model		Python/PyTorch	100%	
4.8	Consider Switching to Alternative Packages (TensorFlow) for Efficiency			0%	
5	**Analysis**				
5.1	Refine Approaches to Confirm the Scope of Research			100%	
5.2	Form Criterias for Hypothesis Testing			100%	
5.3	Document and Pre-Process Result Data for Analysis		Python/PyTorch	100%	
5.4	Perform Analysis			100%	
5.5	Compare Results of Various Models			100%	
5.6	Test Hypothesis with Model Results			100%	
5.7	Search for Areas of Improvement in Analysed Results			100%	
5.8	Compared With Already Existing Literature			100%	
6	**Poster**				
6.1	Plot Illustrative Graphs From Model Results		Pyhon/Matplotlib	100%	
6.2	Integrate Text and Additional Information		Mcs. Powerpoint	100%	
6.3	Review Layout of Poster			100%	
6.4	Poster Submission			100%	20-Feb
6.5	Prepare for Poster Presentation	15% of Unit		100%	
6.6	Poster Presentation			100%	06-Mar
7	**Final Report Writing**				
7.1	Form Final Report Layout		OverLeaf LaTeX	100%	
7.2	Update Methodology and Findings			100%	
7.3	Discuss Results and Findings			100%	
7.4	Update Abstract			100%	
7.5	Proof Read and Check References		OverLeaf LaTeX	100%	
7.6	Sample Writing Submission			100%	20-Mar
7.7	Fix Mistakes Pointed Out In the Feedback			100%	
7.8	Perform Final Updates on the Report			100%	
7.9	Report Submission	80% of Unit		0%	04-May
8	**Activities In Other Units**				
-	Aircraft Aerodynamics Quiz 1				
-	Aircraft Aerodynamics Quiz 2				
-	Aircraft Aerodynamics Quiz 3				
-	Aircraft Aerodynamics Laboratory Report				
-	Control Engineering Coursework				
-	Vibrations & Aeroelasticity Laboratory Report				
-	Operations Management Courswork				
-	CASD Initial System Requirements Report				
-	Heat Transfer Coursework 1				
-	Flight Dynamics Coursework 1				
-	Aerospace Propulsion CW				
-	Structures 3 CW				
-	Flight Dynamics Coursework 2				

Figure 43: Final Gantt chart - project plan (1 of 2)

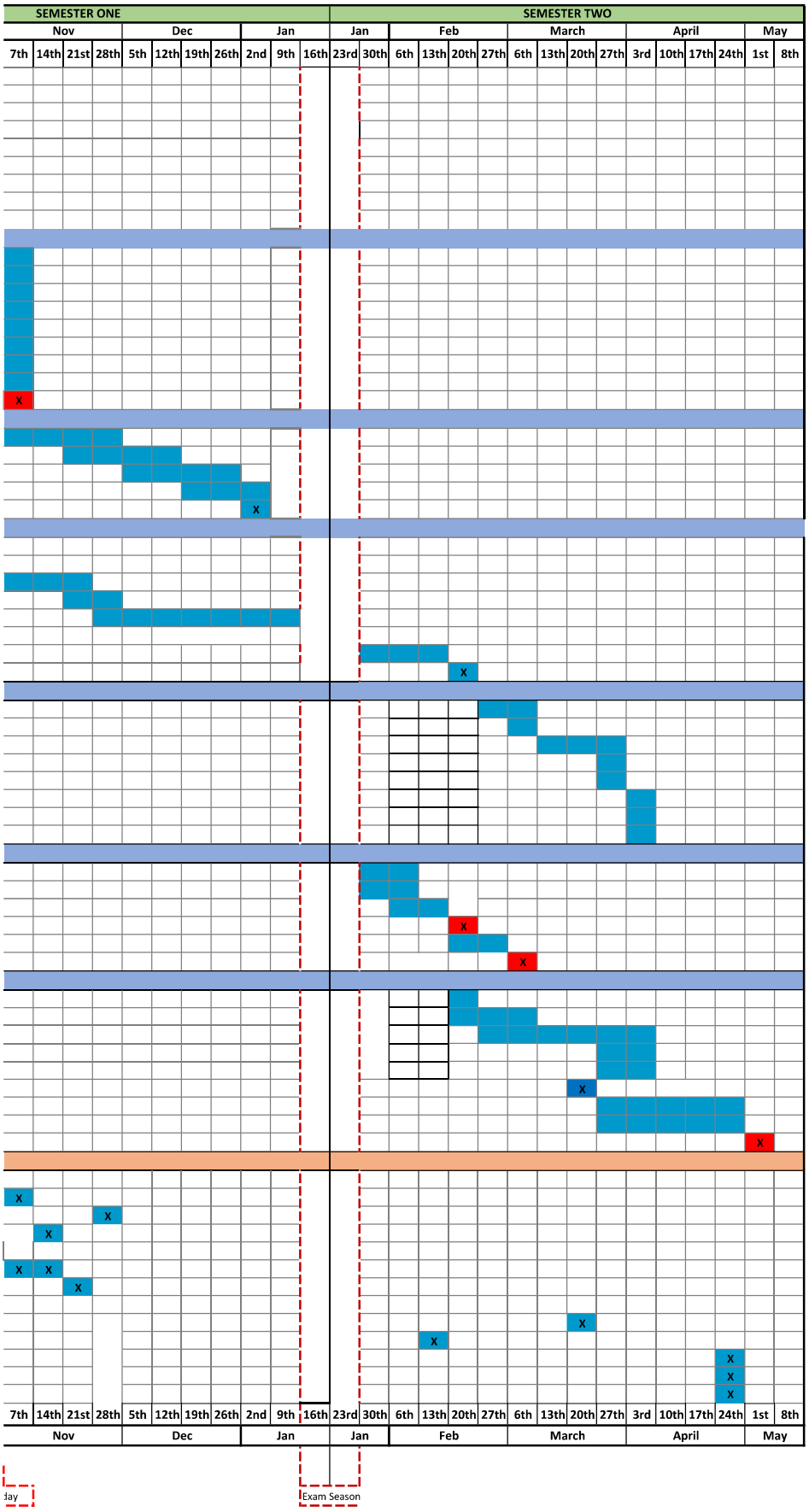


Figure 44: Final Gantt chart - project plan (2 of 2)

8.5 Reflections and changes on the plan

First of all, the Gantt chart produced right after the project proposal was considered to be strongly effective, at an unbelievable level, as it helped the student put their focus onto starting learning the theory on top of coding. This decision had prevented the student from performing "Infinite Monkey Theorem"[8] to write the code, instead, the Gantt chart helped the student understand the logic lying beneath the code as it was being written. This resulted in student's great understanding of the theory and have significantly helped in forming and enhancing the structure of the models developed throughout the project.

Most importantly, this constant progression was a good insight for the student not to become overconfident about their tasks in hand and not to take them lightly, specially a task as serious as the dissertation project. Although, during the dissertation project, it was clear that the student fell into "Dunning-Kruger Effect", by being overconfident about his initial progress. Hence slowing his own progress down due to overconfidence, this can clearly be seen at Figure 46a on the next sub-section, where a significant drop in time spent on the project during November and December periods can be seen after a successful month of October.

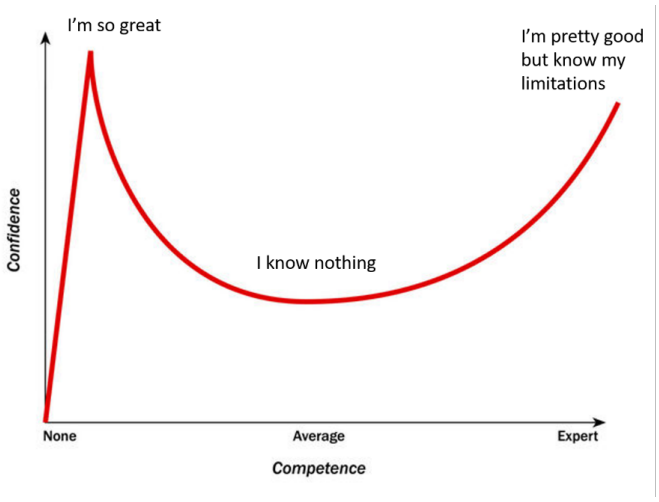


Figure 45: Dunning–Kruger Effect (Lawton, 2022)

The student stayed at, 'I'm so great' phase for longer than they should have, hence further delayed their main aims that were due on the Gantt chart, losing a significant pace in the project. As a result, partially causing significant objectives to be dropped from the project, such as running the MFNN model on complete RANS & LES simulation data sets provided by (Mole, Skillen, and Revell, 2020) and implementing PINNs had to be

[8]Infinite monkey theorem states that if a monkey hits keys at a random on a keyboard with no thoughts for infinity, the monkey would finally type every possible combination of text, including every poet of William Shakespeare, or the MFNN model in this report's case.

scratched from the project.

Additionally, another major mistake was student's optimism about being able to implement BNNs or PINNs into the MFNN with ease, before even getting its basics and details down. Resulting in the student postponing their MFNN related tasks to work on the PINN model, hence resulting in slight delays on the project's schedule, as can be seen from the comparison of the final and project plan Gantt charts.

Despite all the mistakes done, the project was mostly on track due to the drops in further objectives previously mentioned. Subsequently, the original targeted aim and slightly changed objectives were reached by the end of the project.

8.5.1 Time management

The importance of time management has been one of the main focuses for the student throughout the project as it is a 30-Unit project requiring approximately 300 hours of work. Hence, a time management/tracking software called 'Clockify' was used throughout the whole length of the dissertation project to track the time spent on clearly labeled tasks. Additionally, according to the universities recommendation the time spent during semester 2 should be twice of semester 1.

(a) 51 hours spent during semester 1

(b) 170 hours spent during semester 2

Figure 46: Approximately 221 hours was spent on the project in total

To illustrate the total time spent on the project, Figure 46 represents the time spent on each task labeled, Writing, Coding, Poster, Theory and Simulation. As can be seen from the figure a total approximate time of 221 have been spent on the project, with 51 hours during semester 1 and 170 during semester 2. As evident, the time spent was below the recommended 300 hours, where the coding made up 74 hours of this time. Although, it is important to note that these results are more of an estimate, as the student has done significant amount of work during non-tracked times.

Moreover, the use of a time tracking software had positive effects, such as motivating the student to work more and be up to date with his Gantt chart and feel an accomplishment with the time that they have spent on a project that they have enjoyed doing.

8.6 Critical Reflection

8.6.1 Risk assessment and encountered delays

First of all, referring to theoretical and coding aspect of the risk assessment, the problem of the lack of reliable resources and reproduce-ability of results occurred from the literature review, this was a point included in project proposal's risk assessment and it was quite expected. Hence, to oppose this problem, the steps previously mentioned in project proposal was followed, which was utilising the available high quality literature up to a great level, to keep up to date with the project plan. This approach was quite successfully as evident from the report's literature review and results sections. Moreover, keeping in consistent touch with the project's supervisor via weekly meetings and emails have been extremely useful to keep the progress on track.

Secondly, as estimated in the project proposal, there were multiple road blocks that have caused multiple delays in the report's progression. In addition to the delays mentioned in the previous section, delays due to courseworks clashing with dissertation's Gantt chart plans, tougher final year exam season than expected, and un-planned mandatory travel plans have played a significant role in critical moments of the dissertation project.

Moreover, the software and hardware limitations mentioned before hand in the project proposal had not been a problem, since the student was able to perform every needed task with their available hardware and open-sourced software. Additionally, the student had no issues in adjusting to the new coding and ML software introduced to them, as the student learned to use the software effectively with a manageable learning curve to address and upcoming problems.

Lastly, as expected, the majority of the time spent on the project, as evident from Figure 46 was writing the final report and the coding process.

9 Appendix B - Source Code

The reader can contact Emrecan Serin through emrecanserinedu@gmail.com to request access to the GitHub Repository of the source code on request, including the PIP list used throughout the report. Additionally, all the codes provided in this report has been written from scratch by the student.

9.1 Section 4.5 NN complete code

Listing 14: Complete NN code for Section 4.5

```
# %%
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch
import math

# %%
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') # CUDA
↪  device configuration

# -----Model Hyperparameters Initialisation-----
num_epochs = 1000 # No. of epochs
input_dim = 1; output_dim = 1 # Input & Output dimensions
n_per_layer = ([500, 500, 500]) # Neurons per layer
learning_rate = 0.0001 # Learning rate of optimiser
training_data = 13

# Setting the synthetic data set
def F(y):
    return np.sin(8 * np.pi * y)

# Model's training and testing data sets
X_train = torch.linspace(0, 1, training_data)[:,None]
Y_train = F(X_train)


# -----Model Structure Initialisation with 2 Hidden Layers-----
class Network(torch.nn.Module):
  def __init__(self, n_per_layer, input_dim, output_dim): # Layer setup
```

```
30      super().__init__()
31      self.fc1 = nn.Linear(input_dim, n_per_layer[0])  # Input Layer
32      self.fc2 = nn.Linear(n_per_layer[0], n_per_layer[1]) # Hidden Layer 1
33      self.fc3 = nn.Linear(n_per_layer[1], n_per_layer[2]) # Hidden Layer 2
34      self.fc4 = nn.Linear(n_per_layer[2], output_dim) # Output Layer
35
36    def forward(self, x): # Activation function after each layer (ReLu)
37      x = torch.relu(self.fc1(x))
38      x = torch.relu(self.fc2(x))
39      x = torch.relu(self.fc3(x))
40      x = self.fc4(x)
41      return x
42
43  model = Network(n_per_layer, input_dim, output_dim) # Calling Python class
    ↪  as to use the model when needed
44
45  # %%
46  # -----Setting Loss Function and Optimiser-----
47  criterion = torch.nn.MSELoss() # Loss Criterion (Mean Squared Error)
48  optimiser = torch.optim.Adam(model.parameters(), lr=0.005)
49
50  # -----Training Loop (A single Forward and Backward pass per epoch)-----
51  losses = []; val_losses = []; prev_loss = []; loss = torch.zeros(1)
52
53  for epoch in range(num_epochs): # Sets a loop for each epoch
54      prev_loss = loss.item()
55      y_pred = model(X_train) # Initiates forward pass
56      loss = criterion(y_pred, Y_train) # Computes loss from the prediction
57      losses.append(loss.item())
58      optimiser.zero_grad() # Rests gradiets per epoch
59      loss.backward() # Calculates gradients of loss with backpropagation
60      optimiser.step() # Updates model parameters using gradients
61
62  # %%
63  print(f" Number of Epochs needed (out of {num_epochs}): {len(losses)}")
64  print(f" Training Loss of Model: {loss}")
65  plt.plot(losses)
66  plt.xlabel('Epoch')
67  plt.ylabel('Loss')
68  plt.title('Training Loss Graph of Model')
```

```
plt.show()

# %%
# -----Prediction and plotting stage-----
new_X_input = torch.linspace(0, 1, 1000)[:,None] # Random ranged data array for new X_input data

with torch.no_grad():
    y_pred_new = model(new_X_input) # Define the NEW input data

# Define the NEW input datax
fig, ax = plt.subplots(figsize=(10,3))
ax.plot(new_X_input, F(new_X_input), label= "Target ")
ax.plot(new_X_input.cpu().detach().numpy(), y_pred_new.cpu().detach().numpy(),'--k', lw=2, label= "HF Prediction ")
ax.plot(X_train.cpu().detach().numpy(), Y_train.cpu().detach().numpy(), 'bo', markersize = 4, label = 'HF Data Points')
ax.grid(which='both', alpha=0.5); ax.set_xlabel('X', fontsize=11); ax.set_ylabel('Y', fontsize=11); plt.legend(loc='upper right', ncol=1, fontsize='small')
ax.set_title('High Fidelity Network Predictions', fontsize=13); plt.show()
```

9.2 Section 4.6 MFNN complete code

Listing 15: Complete MFNN code for Section 4.5

```
# %%
# A whole new big mess of a step...
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import torch.nn as nn
import torch

# %%
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Model running via: {device}")

# %%
# Globally Shared Model(s) Parameters
```

```
lf_num_epochs = 600
hf_num_epochs = 600
MF_epochs = 600

hidden_dims = ([1000, 1000, 1000, 1000, 1000, 1000, 1000, 1000])
LF_training_size = 25
HF_training_size = 6

def LF(y):
    return 0.5 * (6 * y - 2)**2 * np.sin(12 * y - 4) + 10 * (y - 0.5) - 5

def HF(y):
    return (6 * y - 2)**2 * np.sin(12 * y - 4) - 10 * (y - 1)**2


Z = torch.linspace(0, 1, 1000)[:,None]

# Model's total training/val/testing dataset
LF_TS = torch.linspace(0, 1, LF_training_size)[:,None]
HF_TS = torch.linspace(0, 1, HF_training_size)[:,None]

# Forming LF Test Sets with a neat trick
X_LF_train, X_LF_test, Y_LF_train, Y_LF_test = train_test_split(LF_TS,
↪  LF(LF_TS), test_size=0.8, shuffle=156)
X_HF_train, X_HF_test, Y_HF_train, Y_HF_test = train_test_split(HF_TS,
↪  HF(HF_TS), test_size=0.8, shuffle=156)

# Forming the real equally spaced training sets
X_LF_train = LF_TS
Y_LF_train = LF(LF_TS)

X_HF_train = HF_TS
Y_HF_train = HF(HF_TS)

plt.figure(figsize=(5,3))
plt.plot(Z ,HF(Z))
plt.plot(Z ,LF(Z))
plt.scatter(X_HF_train, Y_HF_train)
plt.scatter(X_LF_train, Y_LF_train)

```

```
# Transfering data to GPU for CUDA
X_LF_train = X_LF_train.to(device); Y_LF_train = Y_LF_train.to(device)
X_HF_train = X_HF_train.to(device); Y_HF_train = Y_HF_train.to(device)

X_LF_test = X_LF_test.to(device); Y_LF_test = Y_LF_test.to(device)
X_HF_test = X_HF_test.to(device); Y_HF_test = Y_HF_test.to(device)

# %% [markdown]
# Low Fidelity Network

# %%
LF_input_dim = 1
LF_output_dim = 1

class LowFidelityNetwork(torch.nn.Module):
  def __init__(self, hidden_dims, LF_input_dim, LF_output_dim):
    super().__init__()
    self.fc1 = nn.Linear(LF_input_dim, hidden_dims[0])
    # self.bn1 = nn.BatchNorm1d(hidden_dims[0])
    self.fc2 = nn.Linear(hidden_dims[0], hidden_dims[1])
    self.fc3 = nn.Linear(hidden_dims[1], hidden_dims[2])
    self.fc4 = nn.Linear(hidden_dims[2], hidden_dims[3])
    self.fc5 = nn.Linear(hidden_dims[3], hidden_dims[4])
    self.fc6 = nn.Linear(hidden_dims[4], hidden_dims[5])
    self.fc7 = nn.Linear(hidden_dims[5], hidden_dims[6])
    self.fcEND = nn.Linear(hidden_dims[6], LF_output_dim)

  def forward(self, x):
    x = torch.relu(self.fc1(x))
    skip_connection = x
    # x = self.bn1(x)
    x = torch.relu(self.fc2(x))
    x = torch.relu(self.fc3(x))
    x = torch.relu(self.fc4(x))
    x = torch.relu(self.fc5(x))
    x = torch.relu(self.fc6(x))
    x = torch.relu(self.fc7(x))
    x = x + skip_connection
    x = self.fcEND(x)
    return x
```

```
93
94  LF_model = LowFidelityNetwork(hidden_dims, LF_input_dim,
    ↪  LF_output_dim).to(device)
95
96  for param in LF_model.parameters():
97      param.requires_grad = True
98
99  # %%
100 LF_model.train()
101
102 # Training
103 LF_losses = []
104 val_losses = []
105 prev_loss = []
106 LF_loss = torch.zeros(1)
107
108 LF_batch_size = 100
109
110 # LF_loss criterion and optimizer
111 criterion = torch.nn.MSELoss().to(device)
112 optimizer = torch.optim.Adam(LF_model.parameters(), lr=0.0005) #
    ↪  weight_decay=1e-5
113
114
115 for epoch in range(lf_num_epochs):
116
117     permutation = torch.randperm(X_LF_train.size()[0])
118
119     for i in range(0,X_LF_train.size()[0], LF_batch_size):
120         optimizer.zero_grad()
121
122         indices = permutation[i:i+LF_batch_size]
123         batch_x, batch_y = X_LF_train[indices], Y_LF_train[indices]
124
125         prev_loss = LF_loss.item()
126         outputs = LF_model.forward(batch_x)
127         LF_loss = criterion(outputs,batch_y)
128         LF_losses.append(LF_loss.item())
129         LF_loss.backward()
130         optimizer.step()
```

```
131
132     if (epoch+1) % 600 == 0:
133
            ↪  print(f'Epoch [{epoch+1}/{hf_num_epochs}], Loss: {LF_loss.item():.4f}')
134
135 # %%
136 plt.figure(figsize=(5,3))
137 plt.plot(LF_losses, label = "Training")
138 plt.xlabel('Epoch')
139 plt.ylabel('LF_Loss')
140 plt.title('Low Fidelity Training Loss Graph of Model')
141
142 plt.plot(val_losses, "--" , label = "Testing")
143 plt.xlabel('Epoch')
144 plt.ylabel('Loss')
145 plt.title('Low Fidelity Validation Loss Graph of Model')
146 plt.grid(which='both', alpha=0.5)
147 plt.legend(loc='upper right')
148 # plt.show()
149
150 print(f"Epochs needed (out of {lf_num_epochs}): {len(LF_losses)}")
151 print(f"LF Training Loss: {LF_loss}")
152
153 # %% [markdown]
154 # High Fidelity Model
155
156 # %%
157 HF_input_dim = 1
158 HF_output_dim = 1
159
160 class HighFidelityNetwork(torch.nn.Module):
161   def __init__(self, hidden_dims, HF_input_dim, HF_output_dim):
162     super().__init__()
163     self.fc1 = nn.Linear(HF_input_dim, hidden_dims[0])
164     # self.bn1 = nn.BatchNorm1d(hidden_dims[0])
165     self.fc2 = nn.Linear(hidden_dims[0], hidden_dims[1])
166     self.fc3 = nn.Linear(hidden_dims[1], hidden_dims[2])
167     self.fc4 = nn.Linear(hidden_dims[2], hidden_dims[3])
168     self.fc5 = nn.Linear(hidden_dims[3], hidden_dims[4])
169     self.fc6 = nn.Linear(hidden_dims[4], hidden_dims[5])
```

```
170     self.fc7 = nn.Linear(hidden_dims[5], hidden_dims[6])
171     self.fcEND = nn.Linear(hidden_dims[7], HF_output_dim)
172 
173   def forward(self, x):
174     x = torch.relu(self.fc1(x))
175     # x = self.bn1(x)
176     x = torch.relu(self.fc2(x))
177     x = torch.relu(self.fc3(x))
178     x = torch.relu(self.fc4(x))
179     x = torch.relu(self.fc5(x))
180     x = torch.relu(self.fc6(x))
181     x = torch.relu(self.fc7(x))
182     x = self.fcEND(x)
183     return x
184 
185 HF_model = HighFidelityNetwork(hidden_dims, HF_input_dim,
    ↪  HF_output_dim).to(device)
186 
187 for param in HF_model.parameters():
188     param.requires_grad = True
189 
190 # %%
191 HF_model.train()
192 
193 # Training
194 HF_losses = []
195 val_losses = []
196 prev_loss = []
197 HF_loss = torch.zeros(1)
198 
199 HF_batch_size = 45
200 
201 # HF_loss criterion and optimizer
202 criterion = torch.nn.MSELoss().to(device)
203 optimizer = torch.optim.Adam(HF_model.parameters(), lr=0.0001,
    ↪  weight_decay=1e-4) # weight_decay=1e-5
204 
205 for epoch in range(hf_num_epochs):
206 
207     permutation = torch.randperm(X_HF_train.size()[0])
```

```
208
209     for i in range(0,X_HF_train.size()[0], HF_batch_size):
210         optimizer.zero_grad()
211
212         indices = permutation[i:i+HF_batch_size]
213         batch_x, batch_y = X_HF_train[indices], Y_HF_train[indices]
214
215         prev_loss = HF_loss.item()
216         outputs = HF_model.forward(batch_x)
217         HF_loss = criterion(outputs,batch_y)
218         HF_losses.append(HF_loss.item())
219         HF_loss.backward()
220         optimizer.step()
221
222     if (epoch+1) % 600 == 0:
223
        ↪  print(f'Epoch [{epoch+1}/{hf_num_epochs}], Loss: {HF_loss.item():.4f}')
224
225
226
227 # %%
228 plt.figure(figsize=(5,3))
229 plt.plot(HF_losses, label = "Training",lw=3)
230
231 plt.xlabel('Number of Epochs')
232 plt.ylabel('High Fidelity Loss')
233 plt.title('High Fidelity Model Loss')
234 plt.grid(which='both', alpha=0.5)
235 plt.show()
236
237 plt.figure(figsize=(5,3))
238 plt.plot(LF_losses, label = "Training",lw=3)
239
240 plt.xlabel('Number of Epochs')
241 plt.ylabel('Low Fidelity Loss')
242 plt.title('Low Fidelity Model Loss')
243 plt.grid(which='both', alpha=0.5)
244 plt.show()
245
246 # print(f"Error: {HF_average_percentage_error}")
```

```
print(f"Epochs needed (out of {hf_num_epochs}): {len(HF_losses)}")
print(f"HF Training Loss: {HF_loss}")

# %% [markdown]
# Multi-Fidelity Model

# %%
embedding = 0.43
L1mean = LF_model(X_HF_train.to(device))
L1mean_up= LF_model(X_HF_train.to(device)+embedding)
L1mean_dn = LF_model(X_HF_train.to(device)-embedding)

L2train = torch.hstack((X_HF_train, L1mean, L1mean_up, L1mean_dn)) # think
↪  of the house price example (sqr feet, rooms, garden, etc.)
print(L2train.shape)

# %%
MF_input_dim = 4
MF_output_dim = 1

class MultiFidelityNetwork(torch.nn.Module):
  def __init__(self, hidden_dims, MF_input_dim, MF_output_dim):
    super().__init__()
    self.fc1 = nn.Linear(MF_input_dim, hidden_dims[0])

    self.fc2 = nn.Linear(hidden_dims[0], hidden_dims[1])
    self.fc3 = nn.Linear(hidden_dims[1], hidden_dims[2])
    self.fc4 = nn.Linear(hidden_dims[2], hidden_dims[3])
    self.fc5 = nn.Linear(hidden_dims[3], hidden_dims[4])
    self.fc6 = nn.Linear(hidden_dims[4], hidden_dims[5])
    self.fc7 = nn.Linear(hidden_dims[5], hidden_dims[6])
    self.fcEND = nn.Linear(hidden_dims[6], MF_output_dim)

  def forward(self, x):
    x = torch.relu(self.fc1(x))
    x = torch.relu(self.fc2(x))
    x = torch.relu(self.fc3(x))
    x = torch.relu(self.fc4(x))
    x = torch.relu(self.fc5(x))
    x = torch.relu(self.fc6(x))
```

```
286     x = torch.relu(self.fc7(x))
287     x = self.fcEND(x)
288     return x
289 
290 MF_model = MultiFidelityNetwork(hidden_dims, MF_input_dim,
    ↪  MF_output_dim).to(device)
291 
292 for param in MF_model.parameters():
293     param.requires_grad = True
294 
295 # %%
296 HF_model.train()
297 
298 # Training
299 MF_losses = []
300 val_losses = []
301 prev_loss = []
302 MF_loss = torch.zeros(1)
303 
304 # MF_loss criterion and optimizer
305 criterion = torch.nn.MSELoss().to(device)
306 optimizer = torch.optim.Adam(MF_model.parameters(), lr=0.00005) #
    ↪  weight_decay=1e-5
307 
308 for epoch in range(MF_epochs):
309 
310     # Training
311     prev_loss = MF_loss.item()
312     MF_y_pred = MF_model(L2train)
313     MF_loss = criterion(MF_y_pred, Y_HF_train)
314     MF_losses.append(MF_loss.item())
315     optimizer.zero_grad()
316     MF_loss.backward(retain_graph=True)
317     optimizer.step()
318 
319     if (epoch+1) % 300 == 0:
320         print(f'Epoch [{epoch+1}/{MF_epochs}], Loss: {MF_loss.item():.4f}')
321 
322 # %%
323 plt.figure(figsize=(5,3))
```

```
plt.plot(MF_losses, lw=3)
plt.xlabel('Epoch')
plt.ylabel('MF Loss')
plt.title('Multi Fidelity Model Loss')
plt.grid(which='both', alpha=0.5)
plt.xlabel('Number of Epochs')
plt.ylabel('Multi Fidelity Loss')

plt.show()

# %%
# Put models
LF_model.eval()
HF_model.eval()
MF_model.eval()

empty = torch.zeros(1000,1).to(device)

# Define the NEW input data
with torch.no_grad():
    y_LF_pred = LF_model(Z.to(device))
    y_LF_pred_up = LF_model(Z.to(device)+embedding)
    y_LF_pred_dn = LF_model(Z.to(device)-embedding)
    y_HF_pred = HF_model(Z.to(device))

    # Step 4: Add X and output of the LF model (similar to Step 2)
    L2test = torch.hstack((Z.to(device), y_LF_pred,  y_LF_pred_up,
    ↪  y_LF_pred_dn))
    y_MF_pred = MF_model(L2test.to(device))


# Data Point Distribution
fig, ax = plt.subplots(figsize=(5,4))
ax.plot(Z, HF(Z), label= "Target ")
ax.plot(Z, LF(Z))
ax.grid(which='both', alpha=0.5)
ax.set_xlabel('X', fontsize=11)
ax.set_ylabel('Y', fontsize=11)
ax.set_title('Low Fidelity Network Predictions', fontsize=12)
```

```
ax.plot(Z.cpu().detach().numpy(), y_LF_pred.cpu().detach().numpy(),'--k', lw=2, label= "LF Prediction ")
ax.plot(X_LF_train.cpu().detach().numpy(), Y_LF_train.cpu().detach().numpy(), 'ro', markersize = 4, label = 'LF Data Points')

ax.legend(loc='upper left', ncol=1, fontsize='small')
plt.show()


fig, ax = plt.subplots(figsize=(5,4))
ax.plot(Z, HF(Z), label= "Target ")
ax.plot(Z.cpu().detach().numpy(), y_HF_pred.cpu().detach().numpy(),'--k', lw=2, label= "HF Prediction ")
ax.plot(X_HF_train.cpu().detach().numpy(), Y_HF_train.cpu().detach().numpy(), 'bo', markersize = 5, label = 'HF Data Points')
ax.grid(which='both', alpha=0.5)
ax.set_xlabel('X', fontsize=11)
ax.set_ylabel('Y', fontsize=11)
ax.legend(loc='upper left', ncol=1, fontsize='small')

ax.set_title('High Fidelity Network Predictions', fontsize=12)
plt.show()



fig, ax = plt.subplots(figsize=(5,4))
ax.plot(Z, HF(Z), label= "Target")
ax.plot(Z.cpu().detach().numpy(), y_MF_pred.cpu().detach().numpy(),'k', lw=2, label= "MF Prediction ")
ax.grid(which='both', alpha=0.5)
ax.set_xlabel('X', fontsize=11)
ax.set_ylabel('Y', fontsize=11)
ax.set_title('Multi Fidelity Network Prediction', fontsize=12)
ax.legend(loc='upper left', ncol=1, fontsize='small')
plt.show()
```

9.3 Section 5.5 numerical examples

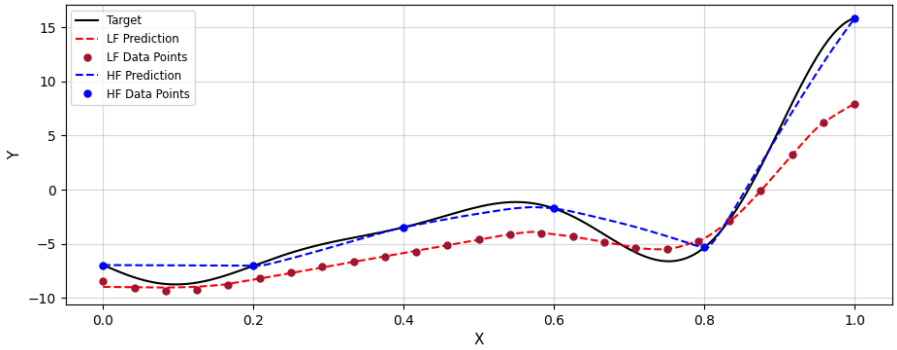
Listing 16: Synthetic Functions written using Python

```python
import numpy as np
import torch

# Synthetic Data - Which gets more complex step by step
# (a) Continuous functions with linear relationship
def QL_a(y):
    return 0.5 * (6 * y - 2) ** 2 * np.sin(12 * y - 4) + 10 * (y - 0.5) - 5

def QH_a(y):
    return (6 * y - 2)**2 * np.sin(12 * y - 4)

# (c) Continuous functions with nonlinear relationship
def QL_c(y):
    return 0.5 * (6 * y - 2)**2 * np.sin(12 * y - 4) + 10 * (y - 0.5) - 5

def QH_c(y):
    return (6 * y - 2)**2 * np.sin(12 * y - 4) - 10 * (y - 1)**2

# (e) Phase-Shifted Osicallations
def QL_e(y):
    return np.sin(8 * np.pi * y)

def QH_e(y):
    return y**2 + QL_e(y + np.pi/10)

# (f) Different Periodicities
def QL_f(y):
    return np.sin(6 * np.sqrt(2) * np.pi * y)

def QH_f(y):
    return np.sin(8 * np.pi * y + np.pi / 10)

# (b) Discontinuous functions with linear relationship
def LF(y):
    zeros = torch.zeros_like(y)
    half = torch.ones_like(y) * 0.5
```

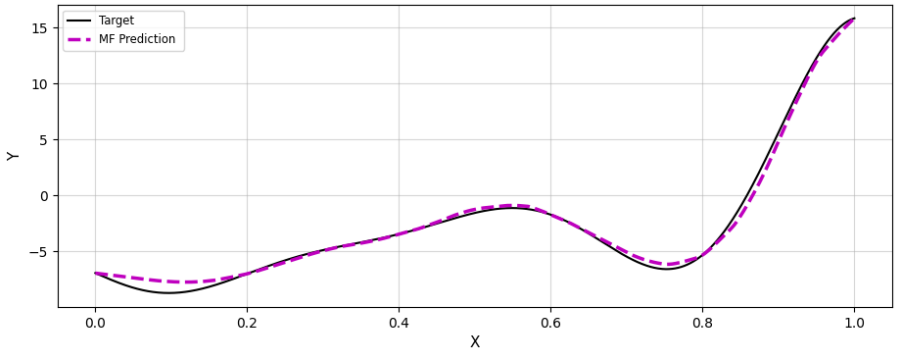
```
    y_0 = torch.where(y <= half, 0.5 * (6 * y - 2)**2 * torch.sin(12 * y - 4) + 10 * (y - 0.5), zeros)
    y_1 = torch.where(half < y, 3 + 0.5 * (6 * y - 2)**2 * torch.sin(12 * y - 4) + 10 * (y - 0.5), y_0)
    return y_1

def HF(y):
    zeros = torch.zeros_like(y)
    half = torch.ones_like(y) * 0.5
    y_0 = torch.where(y <= half, 2*LF(y) - 20*y + 20, zeros)
    y_1 = torch.where(half < y, 4 + 2*LF(y) - 20*y + 20, y_0)
    return y_1

# (d) Continuous oscillation functions with nonlinear relationship
def QL_c(y):
    if 0 <= y <= 1:
        return np.sin(8 * np.pi * y)
    else:
        return ValueError("(d) Range Error")

def QH_c(y):
    if 0 <= y <= 1:
        return (y - np.sqrt(2)) * QL_c(y) ** 2
    else:
        return ValueError("(d) Range Error")
```

9.4 Section 5.5 excluded synthetic results

9.4.1 Continuous functions with nonlinear relationship



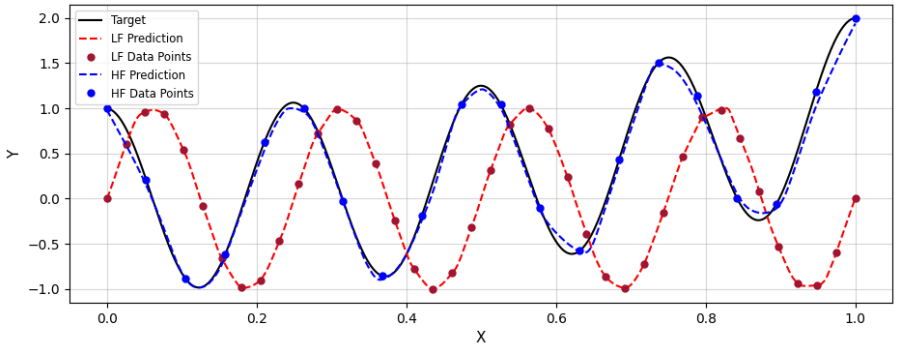
(a) LFNN and HFNN model predictions



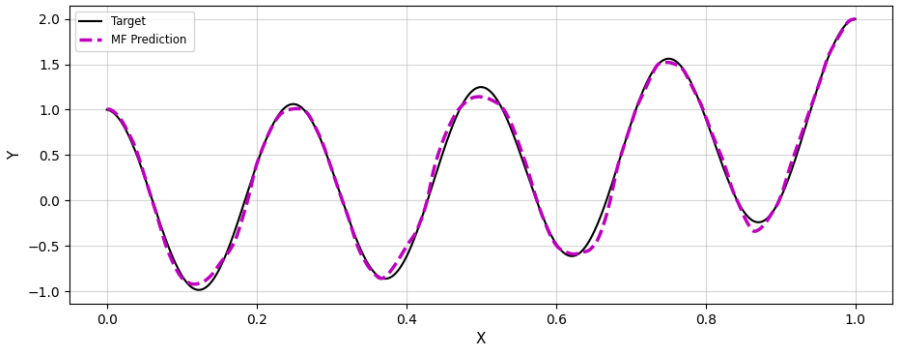
(b) MFNN prediction

Figure 47: (c) Continuous functions with nonlinear relationship

9.4.2 Continuous oscillation functions with nonlinear relationship



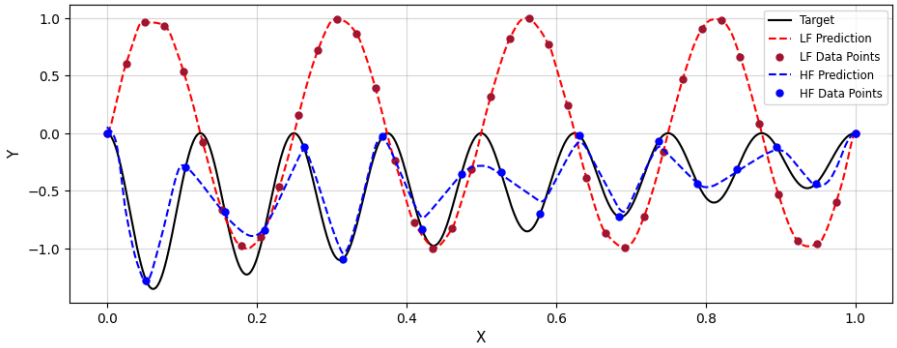
(a) LFNN and HFNN model predictions



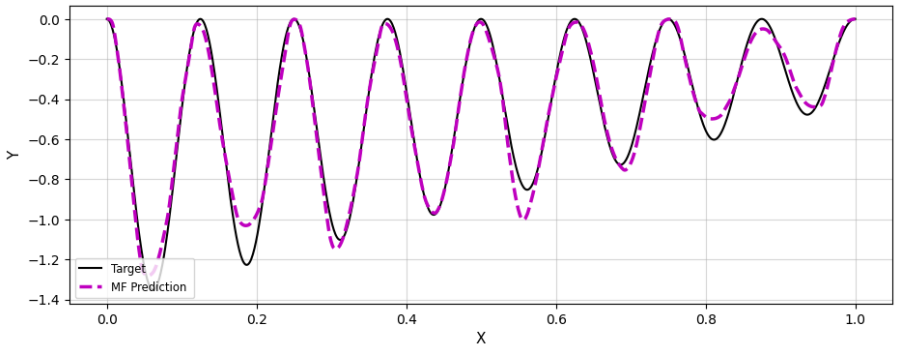
(b) MFNN prediction

Figure 48: (d) Continuous oscillation functions with nonlinear relationship

9.4.3 Different periodicities



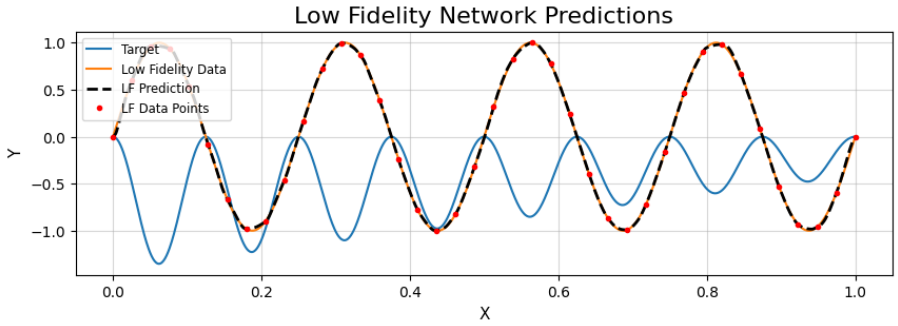
(a) LFNN and HFNN model predictions



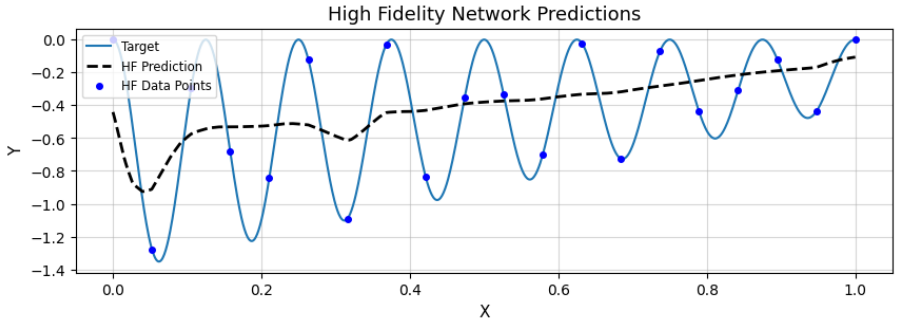
(b) MFNN prediction

Figure 49: (f) Different periodicities

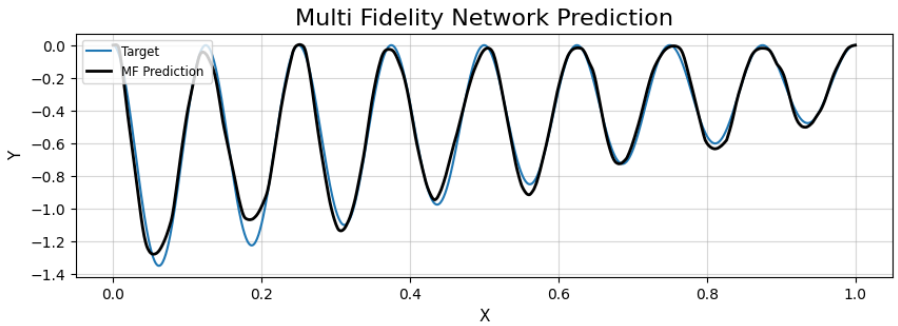
An interesting example encountered during testing:



(a) LFNN prediction



(b) HFNN prediction



(c) MFNN prediction

Figure 50: Example (e) Phase-Shifted Oscillations

9.5 NN schematics and flowcharts

Neural Network Schematics: NN Svg was used to create the Neural Network schematics similar to Figure 2 (Lenail, 2018).

Whereas, the flowcharts using an online diagram software 'draw.io' (Kalder, 2023).