Marginal Emissions Projection in Ireland's Electricity Sector

EEEN30220 Final Report

Jesse Onolememen

20344056

Supervisor: Andrew Keane

April 26, 2024

Abstract

Addressing climate change through emission reduction is imperative for environmental sustainability. The Irish Government, in alignment with the United Nations Sustainable Development Goals (SGD) 7 and 13 [1], is committed to net-zero gas emissions by 2050 [2]. Achieving this target necessitates a dramatic transformation in Ireland's electricity sector, notably through the increased penetration of Variable Renewable Energy (VRE) sources into the national grid, which advances SGD 7.3, but adds complexity to the assessment of climate policy. In this context, Marginal Emissions Factors (MEF) are proposed as an evaluation factor for policies such as load shifting or demand curtailment [3].

This project seeks to apply machine learning techniques for forecasting marginal CO_2 emissions in Ireland's electricity sector. Using a novel approach outlined in Rabiee et al. [4], historical marginal emissions are calculated and used to train an Artificial Neural Network (ANN) to predict MEFs profiles. The erratic nature of MEFs within the highly variable, VRE-penetrated power grid in Ireland necessitates the sophisticated pattern recognition capabilities of an ANN.

However, ANN models are not without ethical considerations. The model's training and validation processes must address concerns about data privacy and consent. In designing the network, it must be ensured that rights of individuals, and entities, that produced the data are well respected. The risk of overfitting raises ethical questions about the reliability and interpretation of predictions. Within the context of this project, overfitting could cause misguided policy decisions, if not carefully managed. Therefore, transparency in the limitations of the model, its intended usage, and interpretation is imperative in ensuring the integrity of the model's predictions.

The best model produced was a 10-input Long Short-Term Memory (LSTM) neural network. Performance was evaluated using Root Mean Squared Error (RMSE) and achieved a best result of 1074 $\frac{\mathrm{g\,CO_2}}{\mathrm{kWH}}$. This suggestes that the network developed an ability to track general marginal emissions trends, but faced challenges predicting the spikes. Traditional regression models discard these spikes as 'noise', and as such, are ignored in typical neural network problems. However, with MEFs, this data is invaluable.

These limitations highlight the necessity for additional research, particularly in expanding the range of variables analysed to enhance the accuracy of predicting these spikes. Avenues for further research could explore incorporating a broader array of variables, including extraneous factors that may influence on marginal emissions, such as weather data, to enhance the model's predictive power. Further refinements on the developed LSTM network can be made, particularly in adjusting the input feature set, network layer configuration, or training parameters. As well, or alternative network designs can be explored, two of which are outlined in this report: a Multi Layer Perceptron (MLP), a hybrid linear-neural model, as done by Mayes et al. [5].

Despite these limitations, the project has laid a foundation for the projection of MEFs in Ireland, of which no prior work has been established in literature. Improvements on the model will offer a new perspective to policymakers in their quest to achieve Ireland's climate goals. The application of this model holds the potential for enriching the assessment and integration of new climate policies, furthering our SGD targets—supporting Ireland's commitment to a sustainable future.

Contents

List of Figures

List of Tables

Listings

Glossary

AEF Average Emissions Factors. 3, 4

ANN Artificial Neural Network. iv, 1, 2, 5, 9, 21, 22, 25, 30, 45

CAISO Californias Independent System Operator. 5

CAP Climate Action Plan. 1

CLI Command Line Interface. viii, ix, 16–18

CNN Convolutional Neural Network. 41

CSO Central Statistics Office. v, ix, 2, 17–20, 30

EV Electric Vehicle. 3, 4

GRA Grey Relational Analysis. vii, 2, 13, 14, 22, 31–33, 37, 47

GRC Grey Relational Coefficent. 13, 14

GRG Grey Relational Grade. vii, 14, 23, 31–33

GUI Graphical User Interface. 48

LSTM Long Short-Term Memory. iv, vi–ix, 1, 2, 5, 7, 12, 14, 24, 27–29, 33–45, 47, 49

MEF Marginal Emissions Factors. iv, vii, viii, 1–4, 12, 13, 18, 20–22, 24, 26, 30–32, 35, 37–42, 44–49

MLP Multi Layer Perceptron. iv, vii–ix, 2, 5–7, 10, 11, 24, 40, 42–45, 49

MSE Mean Squared Error. 43, 44

PCA Principal Component Analysis. 40

RMSE Root Mean Squared Error. iv, ix, 26, 28, 29, 34, 35, 37, 38, 43, 47

RNN Recurrent Neural Network. 5

SEMO Single Electricity Market Operator. 18, 41

SGD Sustainable Development Goals. iv, 2, 49

SVM Support Vector Machine. 5

UC Unit Committment. 4

UI User Interface. vii, 48

VRE Variable Renewable Energy. iv, 1, 3, 4, 7, 32, 39, 41, 47, 48

1 Introduction

Meeting the net zero emissions targets outlined in the Climate Action Plan (CAP) [2] necessitates a transition from conventional, CO_2-intensive fossil fuel generation to renewable energy sources. In Ireland, wind and hydro power, which require specific conditions for production, are the primary sources of renewable energy [8]. This reliance results in a highly dynamic power system characterised by a fluctuating share of renewable generation.

Carbon emissions in the electricity sector are contingent upon the carbon intensities of generation sources used to balance the load [9]. In a traditional fossil-fuel-based system, assessing the environmental impact of demand changes is straightforward given the static nature of conventional generators. In contrast, the variable nature of renewable sources retards this assessment. Renewable sources are inherently carbon deficient, but their variability often necessitates supplemental conventional generation to meet demand gaps and ensure grid stability.

Theoretically, marginal emissions could be determined by identifying the generator responsible for the last megawatt of power (i.e., the marginal operator) and its carbon intensity. However, the integration of renewable generation makes it difficult to identify this operator. While conventional generators operate within predictable system constraints (.i.e. ramp-rates, min/max output, min up-times etc.), renewable sources do not. This leads to emissions intensities that vary significantly throughout the day [5]. As a result, the timing of a load can significantly affect carbon emissions: at one time period the marginal operator could be a clean wind generator, while at another, it may be a highly polluted fossil fuel plant. Integrating new technologies (.e.g electric vehicles) intended to mitigate carbon emissions, can inadvertently exacerbate them if the increased demand is incommensurate with renewable energy sources [9].

1.1 Motivations

In Ireland, environmental impact assessments currently rely on average emissions data, which lack the granularity provided by marginal emissions assessments [9]. To address this, a method to evaluate marginal emissions factors in Ireland was developed in Rabiee et al. [4]. This project applies that method to assess historical MEFs at a half-hourly interval. This historical data is then used to train an ANN for forecasting MEFs in response to demand profiles and fuel mixes.

The final design for the neural network is a LSTM network. LSTMs have been effectively used in electrical and energy engineering for load forecasting, VRE generation forecasting, and emissions predictions [10, 11, 12]. LSTMs are particularly suited for this project due to their ability to process time series data and 'remember' information from prior inputs. [10]

The application of GRA, established in previous research [7, 13], is employed in this project to discern the factors influencing Marginal Emissions. Alternative network designs are proposed as MLP model, and a LSTM linear composite network design, similar to the one used in Mayes et al. [5].

1.2 Aims

This project aims to develop a method for predicting the marginal emissions factor's in Ireland's electricity sector, leveraging machine learning techiques, from the historical datasets of EirGrid and CSO.

1. Neural networks and machine learning techniques are integrated for forecasting MEFs given their advantage in dealing with complex non-linear multivariate time series. [14]
2. The model will be test against the historical MEFs calculated from 2020 onwards through the publicly available data from EirGrid the MEF calculation from [4]
3. Identifying the underlying and mitigating factors that influence marginal emissions through the application of feature analysis techniques such as Grey Relational Analysis (GRA) [13]
4. The potential application of the model by stakeholders and policy makers in evaluating the effectiveness of climate policies and for climate consequence analysis in alignment with SGD 7.1, 7.3, 7.a and 13.2 [1]

1.3 Outline

The report is laid out as follows

- Section 2 presents a literature review which examines
  - the latest developments surrounding marginal emissions and their applications in climate policy
  - common methods used for forecasting power system parameters and marginal emissions
  - a breakdown into machine learning techniques and applications in electrical engineering
  - a breakdown of MLP and LSTM ANN designs and their applications in literature
  - an overview of Grey's theory and the application of GRA in ANN feature selection
- Section 3 details the methology and a complete overview of the work completed
  - outlines the steps in designing ANN (data collection, processing etc.) with relevant code listings
  - outlines the extra considerations taken in feature selection and network selection
  - outlines the data windowing technique used for LSTM sequence formation
  - outlines the procedures for training and validating the network
- Section 4 presents the results from the project
  - the historical MEFs in Ireland's electricity sector
  - the identified influencing factors on MEF based on the GRA and statstical analysis
  - the results and performance of the LSTM model
- Section 5 discusses the results, challenges and outlines alternative network designs
- Section 6 concludes, offering a basis for future work, addresses ethical and sustainability concerns

2 Literature Review

2.1 Average vs Marginal Emissions

2.1.1 Average Emissions

Average Emissions Factors (AEF), also known as carbon intensity, are a conventional measure for assessing the carbon-climate impacts of electricity generation. Typically AEFs are calculated by dividing the total carbon emissions of a power system by the total electricity generated, yielding the average CO_2 emissions per unit of generation [4]. This implicitly assumes equal responsibility for emissions across all generators in the grid, irrespective of their individual emission levels. There is no distinction given between a highly polluted coal generator and 'cleaner' natural gas generator. While offering a simplistic overview, AEFs mask the significant discrepancies in emissions between different types of generators (VRE, conventional), thereby providing an incomplete picture of the true environmental impact of the power grid [9].

2.1.2 Marginal Emissions

Marginal Emissions Factors consider the change in carbon emissions as a consequence of a specific action within the system (.i.e. a change in demand) [4]. It is the carbon impact associated with the marginal operator—which may vary between renewable or non renewable— used to commensurate this action. MEFs vary significantly in VRE-penetrated grids (such as Ireland's) where the marginal operator can frequently shift between low-emission renewable sources and higher-emission conventional sources [5]. This variability means that the environmental impact of additional electricity consumption (or savings) can vary dramatically, depending on the specific generators meeting that marginal demand.

2.1.3 Policy Implications

Average emissions are based on the assumption that any change in electricity demand is uniformly distributed across all generators which does not hold in practise [3]. Only the marginal operator can react to short term demand fluctuations given typical system constraints (e.g. ramp-rates, minimum up or down times, etc.). The carbon intensity of this marginal generator can vary significantly from the system wide average assumed by AEFs. Relying solely on AEFs for carbon consequence analysis can lead to inaccuracies [9, 4], as they do not account for the dynamic nature of power generation and the specific contributions of individual generators.

2.1.3.1 Climate Policy In Holland et al. [9] researchers contend that effective climate policies should concurrently aim to reduce both average and marginal emissions to achieve environmental objectives. A notable example cited in the paper is US-President Biden's initiative to have Electric Vehicles (EVs)

constitute 50% of new vehicle sales by 2030. An analysis based solely on AEFs might suggest that the incremental CO_2 emissions from this increased EV usage would be offset by the reduction in CO_2 emissions from traditional vehicles, especially considering the declining trend in average emissions. However, when this policy is evaluated using MEFs an adverse effect emerges. The surge in electricity demand, primarily met by the marginal operator– often a coal plant— leads to a rise in output and an increase in emissions. This results in an overall detrimental impact on the climate, contrary to the policy's intended goal.

2.1.3.2 Demand Response Programs MEFs are most suitable for demand-side policies and initiatives that necessitate changes in both the timing and magnitude of electricity loads. Unlike AEFs, which lack the granularity needed for precise analysis, MEFs can accurately reflect the environmental impact of shifting electricity demand patterns. For instance Tran et al. [15] developed a demand response program aimed at minimising the environmental cost of operating refrigerators. This cost is quantified using forecasted marginal emissions. Similarly, MEFs have been utilised in another demand-response application Pepiciello et al. [3], which encourages consumers to adjust their energy usage to periods with lower marginal emissions. This approach is facilitated by a predictive day-ahead model that identifies 'cleaner' periods with reduced emission intensity from electricity generation.

2.2 Forecasting Methods

2.2.1 Unit Commitment

Unit Committment (UC) models are designed to determine the optimal economic dispatch and operation of power plants on the grid, catering to a specified demand level within a framework of network constraints. A UC based model to predict MEFs does so by assessing the change in carbon emissions over time, from t to $t + \Delta t$, based on the system's updated dispatch.

UC models operate on a set time step (e.g., hourly), which might not be sufficient to capture the finer fluctuations in renewable energy generation and corresponding emissions. The granularity of data used in UC models may lead to inaccuracies in identifying the true marginal unit, especially during periods of rapid VRE ramp-up or ramp-down [16].

UC models often simplify the intricate dynamics of the electrical grid. Simplifications can understimate the real-time variability and unpredictability introduced by VREs [5]. VRE sources, such as wind and solar power, do not conform to the traditional operational constraints integral to UC models, like fixed ramp rates, minimum output levels, or predictable maintenance downtimes. The inherently intermittent nature of renewable energy sources results in swift fluctuations in generation capacity, retarding the ability for UC models to accurately forecast emissions at the margin.

2.2.2 Machine Learning

The application of machine learning techniques is an established precedent when empirical and statistical models fall short in capturing complex, multivariate, non-linear relationships. Various machine learning methods have been explored in the literature, demonstrating their effectiveness in different contexts:

Singhal and Swarup [17] employed a ANN for electricity price forecasting. The ANN, specifically a two-hidden-layer MLP with 9 input features, was chosen for its proficiency in mapping the intricate interdependencies between electricity price, historical load, and other factors.

Maarif et al. [10] employ a LSTM neural network to forecast energy usage. The authors highlighted the inability of traditional statistical methods to account for the non-linearity in energy usage patterns. The LSTM design was selected to address the limitations of traditional Recurrent Neural Networks (RNNs), particularly their issues with sustaining long-term dependencies effectively.

Singh and Dubey [18] developed a LSTM-based ANN to predict CO_2 emissions using vehicle sensor data, while Bouziane et al. [11] used an LSTM for modelling renewable energy production and CO_2 emissions in Algeria Amarpuri et al. [12] created a hybrid approach combining two network topologies to predict CO_2 emissions in India.

Mayes et al. [5] developed a MLP model to forecast Marginal Emission Factors from historical data provided by Californias Independent System Operator (CAISO). They emphasised the MLP's superior capability in handling the temporal aspects of data compared to traditional grid or statistical models.

Wang et al. [19] utilised a Support Vector Machine (SVM) approach to estimate and predict hourly marginal emissions in the PJM market area in the United States.

2.3 Artificial Neural Networks (ANNs)

Neural networks can be understood as sophisticated function estimators. Their primary objective is to contrive a function that captures the underlying relationship between a given set of inputs and their corresponding outputs. This function is then used to predict outputs for new, unseen inputs, embodying the essence of machine learning where the model 'learns' the relationship inherent in the data.

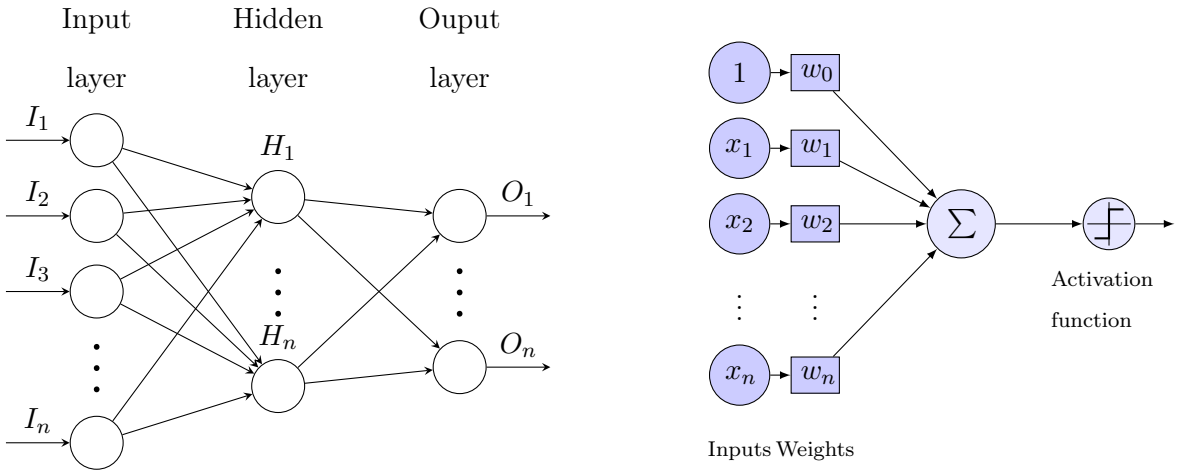
The architecture of ANNs is inspired by biological neural systems. Just like the human brain processes data and information through a network of interconnected neurons, ANNs simulate this process using artificial neurons or perceptrons. These nodes are linked by 'synapses', analogous to connections in biological neural networks, which facilitate the transfer and processing of information [20].

Each of these artificial synapses are assigned a weight, which is adjusted during the learning process. When new input data is fed into the network, these synaptic weights change, allowing the network to

adapt and improve its accuracy in generating responses. This process of weight adjustment and adaptation is at the heart of how ANNs learn and make predictions, enabling them to solve complex problems and recognise patterns in data.

2.3.1 Multi-layer Perceptron (MLP)

Mohammadazadeh et al. [20] introduces MLP as one of the simplest neural network designs consisting of:



(a) Simplified MLP Network Topology **(b)** Perceptron model

Figure 1: Multi layer network topologies

- **Input Layer** which receives the data fed into the network in its raw form
- **Hidden Layer** which does the actual processing through a system of weighting neurons. A network may contain one, or more hidden layers
- **Output Layer** which produces the output of the network

The hidden layer is the 'brain' of the neural network and is composed of interconnected neurons. Each neuron is modelled as a 'perceptron' which contains a series of external inputs, and internal input called a bias, a threshold and an output [20].

A perceptron output modelled as $y(i)$ where $g(i)$ is the activation function

$$y_i = g(i) \times (w_0 + x_1 w_1 + ... + x_n w_n) = \sum_i w_i x_i \times g(i) \tag{2.1}$$

Training of a neural network is done through back propagation, iteratively adjusting the perceptron weights to minimise the error in output. Initially the network starts with random assigned weights. During training an input is fed forward through the network layers producing an output. The output is compared with the desired output and the difference between them constitutes the error [20, 21]

The gradient of error with respect to each weight is calculated determining how much each weight con-

tributes to the overall error. The weights are then adjusted in the direction that most reduces the error. This adjustment is achieved using gradient descent where the weights are updated in proportion to their error contribution. [21].

The training continues until the network reaches an acceptable level of accuracy or a predefined number of iterations. Through this process of continuous learning and adjustment, an MLP network fine-tunes its weights to produce an output that closely aligns with the desired outcome, effectively learning the underlying pattern in the data.

2.3.2 Long-term Short Term Neural Network

LSTMs are a specialised form of neural networks renowned for their proficiency in addressing time series forecasting challenges. Their distinct advantage lies in their ability to both retain and utilise historical data effectively [10]. LSTMs have been extensively in literature with implementations in energy usage prediction [10], VRE production forecasting [11], price forecasting [17], and carbon emissions forecasting [12, 11, 18].

2.3.2.1 LSTM Cell An LSTM network is characterised by its unique structure, which includes an additional hidden layer consisting of LSTM cells. Each LSTM cell is composed of four distinct gates, contributing to the network's capacity to process and analyse temporal data. A schematic representation of an LSTM cell is provided in the figure below.

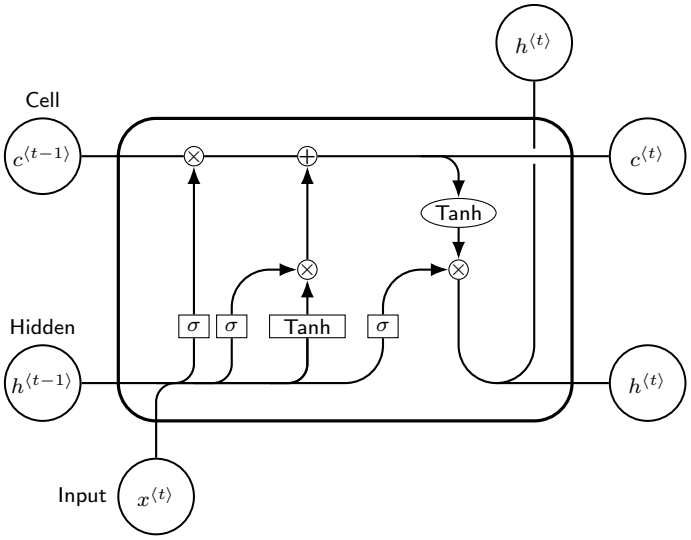


Figure 2: Diagram of an LSTM memory cell

Forget Gate The forget gate is used to determine which messages pass through the cell and enter the input gate. It is responsible for determining which information from the memory state unit c_{t-1} should

be forgotten or maintained in the current memory unit state c_t.

$$f_t = \sigma(W_f x_t + U_f h_t - 1 + b_f) \tag{2.2}$$

eq. (2.2) represents he output of the forget gate at time t ranging from $(0, 1)$. When the output is closer to zero, it is more necessary to forget past information and vice versa. [10, 12].

- W, U indicate the weight matrix concerning the input and hidden units (respectively)
- b is the bias matrix

Input Gate From the forget gate information is carried to the input gate. The input gate determines the number of new messages to add to the cell's internal state. It is responsible for determining which information from an input x_t must be updated in the current state of the memory unit c_t.

$$i_t = \sigma(W_i x_t + U_i h_t - 1 + b_i) \tag{2.3}$$

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_t - 1 + b_c) \tag{2.4}$$

During the process a portion of the information in the memory gate is discarded. A portion of the vital information in x_t is transferred to the memory unit c_t. The process of updating information is performed using eq. (2.5) [10] where $\otimes$ represents the element-by-element wise multiplication of two vectors.

$$c_t = f_t \otimes c_t + i_t \otimes \tilde{c}_t \tag{2.5}$$

Output Gate The output gate performs the final calculation. It determines which information from the current memory unit c_t must be transmitted to the current hidden unit h_t. The value of the current hidden unit is calculated using [10]

$$h_t = \sigma(W_o x_t + U_o h_t - 1 + b_0) \otimes \tanh(c_t) \tag{2.6}$$

where h_t is the output of the hidden unit, c_t the memory unit, i_t the input gate at time t.

2.3.3 Backpropagation

Backpropagation is the algorithm used by neural networks to learn by adjusting the weights of connections [6]. The goal is to reduce the error between the predicted and the target values. Marsland [6] offers a

simplified example, where the error for each output neuron is calculated using a sum-of-squares error:

$$E(t, y) = \frac{1}{2}\sum_{k=1}^{n}(t_k - y_k)^2 \tag{2.7}$$

here t_k and y_k are the target and the predicted values. Training involves repeatedly adjusting the weights to minimise the error which improves the performance of the network.

2.3.3.1 Gradient Descent Gradient descent is an optimisation technique for finding the local minimum or maximum of a differentiable function by using the steepest gradient. The optimal solution is got when the slope of the gradient is zero [22]. Gradient descent is used in neural networks to minimise the error function given by eq. (2.7). Marsland [6] states the gradient of the error regarding the weights is computed:

$$\nabla E(w) = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_n}\right] \tag{2.8}$$

where each component of the gradient is calculated by applying the chain rule:

$$\frac{\partial E}{\partial w_{ik}} = (t_k - y_k)(-x_i) \tag{2.9}$$

Here, x_i represents the input to the neuron, and $t_k - y_k$ is the error term from the output. The weight update rule is then applied:

$$w_{ik} \rightarrow w_{ik} + \eta(t_k - y_k)x_i \tag{2.10}$$

where η is the learning rate, which determines the step size during the gradient descent update. This process is repeated until the algorithm converges to a minimum. Backpropagation calculates the gradient needed for the gradient descent method, adjusting each weight based on its contribution to the overall error to optimise the network towards more accurate predictions.

2.3.4 Hyperparameters

Hyperparameters are settings that influence the architecture and training behaviour of ANN. They are set before the training process begins and remain constant throughout [14, 6]

2.3.4.1 Learning Rate Denoted by η, it controls the change in magnitude of the network's weights, dictating the step size at each iteration toward a loss function's minimum, as described by Marsland [6].

A high learning rate can lead to instability, as the network might overshoot the minimum, failing to converge. Conversely, a low learning rate, while offering stability during the training process, increases the likelihood of the network becoming trapped in a sub-optimal local minima and prolongs training [6].

Adaptive learning rate strategies address these challenges by initially employing a higher learning rate to circumvent shallow local minima and gradually reducing it to refine the search for the global minimum, enhancing stability and accuracy.

MATLAB's implementation, as outlined by Mark Hudson Beale et al. [23], allows for specifying an `InitialLearnRate` and adjusting it at predefined intervals by a decay factor.

2.3.4.2 Number of Epochs An epoch represents a complete training cycle where all training data has propagated through the network. Specifying the number of epochs determines how many times the entire dataset is processed during training.

While theoretically, a network could train indefinitely, it will eventually stop generalising and begin to overfitting the training data. To prevent this, Marsland [6] recommends an early-stopping mechanism that defines the optimal number of training epochs.

Early stopping halts training when there is no improvement in validation performance beyond a predefined threshold. For instance, in the design of a MLP, one might initially set a target of 1000 epochs. However, if the validation error rises beyond 5% after 340 epochs, it suggests the network has stopped learning effectively at this point. Therefore, the optimal number of epochs would be around 340.

This approach to early stopping not only prevents overfitting but also optimises the training duration. The MATLAB implementation is detailed in Mark Hudson Beale et al. [23], with the `ValidationFrequency`, `ValidationPatience` and `trainParam.max_fail` properties.

2.3.4.3 Batch Size The batch size, as defined by Marsland [6], is the number of training samples processed before the model updates its internal parameters. Choosing a smaller batch size can extend the training duration but might improve the network's learning capabilities, as it allows for more frequent updates with a higher sensitivity to data variability. A larger batch size speeds up the training process but may reduce the model's overall performance because of fewer updates and decreased sensitivity to individual data points.

2.3.4.4 Number of Hidden Layers Hidden layers can be thought of as the 'brain' of the network (see section 2.3.1) Marsland [6] suggests that two hidden layers are sufficent for most MLP networks

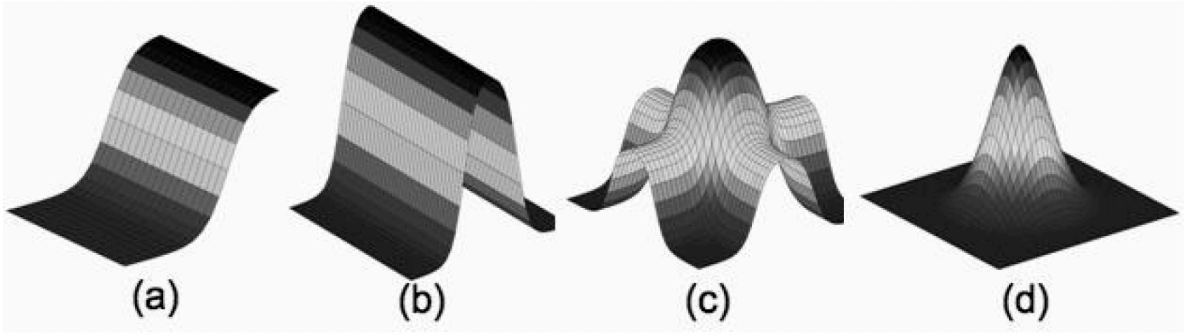


Figure 3: Photo extract from Marsland [6], illustrating the function approximation process in a MLP

In fig. 3 above,

(a) represents the output of a single sigmoidal neuron

(b) shows the combination of two sigmodial neurons, with one reversed

(c) shows the cumulative effect of combining multiple sigmoidal neurons to form complex shapes

(d) demonstrates the refinement of these shapes into sharply defined peaks

A single hidden layer can use sigmoid functions to approximate simple ridges or bumps in data. By combining outputs from multiple sigmoid functions, you can create more complex, ridge-like shapes. With two layers, the network can further transform these shapes into even more intricate patterns.

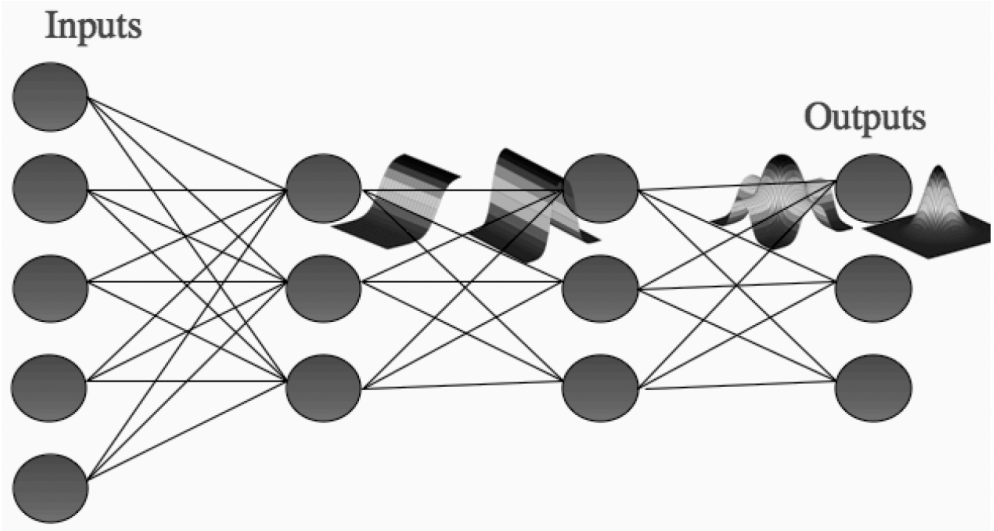


Figure 4: Photo extract from Marsland [6] showing the topology of a MLP with the learning shape at each stage

Essentially, this means that with two layers of neurons, the network can approximate virtually any function to high accuracy by combining these bumps in various ways. This capability makes two layers quite effective for a wide range of tasks [6]

2.3.4.5 Number of Neurons

The architecture of a neural network necessitates a specified count of neurons in each layer. The input layer is determined by the number of features introduced to the network, while the output layer corresponds to the number of predicted variables.

Determining the optimal quantity of neurons in the hidden layers, however, is not guided by a definitive theory, as stated by Marsland [6]. The complexity and uniqueness of each problem require experimentation to identify the most effective number of hidden neurons.

More hidden neurons increase the model's complexity, given a larger number of weight connections that require optimisation. While this can enhance the network's capacity to model intricate relationships, it can also lead to performance degradation.

2.4 Grey Systems Theory

Yi Lin and Sifeng Liu [24] describes how grey systems theory was developed in China in the 1980s to process grey information by developing models and methods that can handle the uncertainty and incompleteness inherent in many real-world systems.

2.4.1 Grey Information

Grey information refers to information that is partially known and partially unknown. This partial knowledge can be because of incomplete, uncertain, or imprecise data [25, 13, 7].

The term 'grey' is used to distinguish this type of information from 'black' and 'white' information, where black represents completely unknown information and white represents completely known information.

2.4.2 Grey Systems

Grey systems are characterised by partial knowledge—specifically, they are systems where some information is known and some information is unknown. This concept stands in contrast to black systems (completely unknown systems) and white systems (completely known systems) [24, 25]

Liu et al. [25] delineates that any system (e.g. economic, power, or educational system) incorporates various factors. These factors are categorised into primary and secondary groups, wherein the primary factors predominantly influence the system's behaviour, whereas the secondary factors have a lesser impact.

When developing regression models (such as LSTM networks) it is crucial to amplify the role of primary factors while mitigating the influence of secondary factors. Referring to MEFs, though multiple factors contribute to determining the marginal operator, only the primary factors should be significant. Including secondary, less pertinent factors might introduce noise and dampen performance.

2.4.3 Grey Relational Analysis

Grey Relational Analysis (GRA) is used to analyse grey systems where statistical methods fall short [25].

- Statistical models typically require complete datasets for accurate analysis. Grey systems that contain partial knowledge and missing data do not satisfy this requirement [25, 24]. Given the multitude of primary and secondary influencing factors associated with MEFs, it is challenging to uncover the completeness of any dataset.
- Many statistical models assume data randomness and follow specific probability distributions. Grey systems often involve non-random and uncertain data that do not conform to predefined distributions [25, 7]. MEFs do not adhere to any typical probabiltiy distributions given their variable nature.
- Statistical methods often require large datasets to ensure the reliability of the results. Grey systems often deal with limited data samples [25]
- Outliers or extreme values can significantly influence the results in Grey systems, where such extremes might carry important information or show the underlying system dynamics. Statistical models are sensitive and do not perform well to outliers [25]. For MEFs these outliers are crucial, like the extreme levels of change in emissions between time steps.

Liu et al. [25] explains that GRA employs a method based on comparing the geometric shapes of data sequences. GRA evaluates the closeness of their relationships by comparing the similarity of the geometric curves. A high degree of similarity between the curves shows a strong relational connection between the sequences, whereas large differences in the curves suggest a weaker relationship.

Sallehuddin et al. [7] outlines three main steps of performing GRA

1. All features must be encoded to a machine-readable format (such as date times)
2. The data is processed to ensure all sequences are on the same scale. This process is described by Sallehuddin et al. [7] as generation the grey relation, or 'standard-processing'.
3. The Grey Relational Coefficent (GRC) is located using eq. (2.11)

$$\zeta_i(k) = \frac{\Delta\min + \zeta\Delta\max}{\Delta_{0,j}(k) + \zeta\Delta\max} \tag{2.11}$$

$$\Delta_{0,j} = ||x_0^*(k) - x_i^*(k)|| \tag{2.12}$$

where,

- $\Delta_{0,j}$ is the deviation sequences between the target and predictors,
- $x_0^*(k)$ is the target sequence (.i.e. MEFs)

- $x_i^*(k)$ is the predictor sequence (.i.e. input features)
- ζ is the identification co-efficent between 0-1. Typically $\zeta = 0.5$. Adjusting the value will only change the magnitude of the GRC but not change the rank of the GRA.

4. The Grey Relational Grade (GRG) is calculated by averaging the value of the grey relational coefficient $\zeta_i(k)$

$$\gamma_i = \frac{1}{n} \sum^{n} k = 1\zeta_i(k) \tag{2.13}$$

it is a numerical measure of the relevance between two systems or sequences and is between $(0, 1)$.

According to Sallehuddin et al. [7], if a particular predictor variable is more relevant to a given target, then the GRG γ_i will be higher. If $\gamma_i(x_0, x_i) > \gamma_i(x_0, x_j)$ then the element x_i is closer to the reference element x_0 than x_j. Given a typical $\zeta = 0.5$ value, the GRG can be interepreted as follows,

- $\gamma_i > 0.9$ indicates a marked influence
- $\gamma_i > 0.8$ indicates a notable infleunce
- $\gamma_i > 0.7$ indicates a subtle influence
- $\gamma_i < 0.6$ indicates a neglible influence

2.4.4 Neural Network Feature Selection

Huang et al. [13] uses GRA as a feature selection mechanism in building a LSTM neural network to predict CO_2 emissions in China. An empirical analysis and literature review identified 16 input features. The dimensionality of these input features was reduced by asessing the GRGs and degree of relational significance between the input sequences and carbon emissions.

Several features with a $\gamma_i > 0.9$ were kept in the final feature set, given their strong correlation. Conversely, the integrated circuit feature, with a $\gamma_i \approx 0.5$, was excluded from the final input set, as it was weakly correlated with CO_2 emissions. The authors highlight that the accuracy of the final model underscores the efficacy of GRA as a feature selection tool.

3 Methodology

3.1 Marginal Emissions Factor

The method developed in Rabiee et al. [4] provides an estimate of the marginal emissions factors in Ireland's electricity sector based on the historical data available from EirGrid. The calculation is based on the assumption that within a 15-minute time interval t any change in the system can be represented as a deviation from an existing operating point, such that the data at the next interval $t + \Delta t$ reflects the updated system dispatch.

The marginal emissions factor reflects the change in carbon emissions associated with a specific action within the system. Or, rather, the commensurate change in CO_2 emissions as a consequence of this action.

For a network of k generators, each with an associated emission intensity of ϵ_k, the marginal increase in CO_2 emissions for a given change in demand ΔPG from $t \to t + \Delta t$ is

$$ME_t = \frac{\sum_k \epsilon_k \Delta PG_{k,t}}{\sum_k \Delta PG_{k,t}} \tag{3.1}$$

Carbon intensity (CI) as defined by EirGrid, is calculated at each time interval t, for k generators, as [4]

$$CI_t = \frac{\sum_k \epsilon_k PG_{k,t}}{PG_{k,t}} \tag{3.2}$$

.i.e. the average carbon emissions per unit of electricity demand. Consider that the change in demand represents the re-dispatch in generation from t to $t + \Delta t$

$$\Delta PG_k = PG_{k,t+\Delta t} - PG_{k,t} \tag{3.3}$$

Combining eqs. (3.1) to (3.3) arrive at

$$ME_t = \frac{CI_{t+\Delta t} \sum_i PG_{i,t+\Delta t} - (CI_t \sum_i PG_{i,t})}{\sum_i PG_{i,t+\Delta t} - \sum_i PG_{i,t}} \tag{3.4}$$

Listing 1: Python helper function used to calculate the historical marignal emissions from a given dataframe

```
def compute_mef(frame: pd.DataFrame) -> pd.DataFrame:
    df = frame.copy()
    df['deltaP'] =  df['GenExp'] - df['GenExp'].shift(-1)
    df['epsilon'] = df['Co2Intensity']*df['GenExp']
    df['MarginalEmissions'] = (df['epsilon'] - df['epsilon'].shift(-1)) / df['deltaP']
    return df.dropna().replace([np.inf, -np.inf], np.nan).ffill()
end
```

A helper function was written to apply this calculation in Python with Pandas dataframes. According to eq. (3.4), the emissions factor becomes undefined if the difference in generation (i.e., 'deltaP') is zero. In such instances, the code uses `df.ffill()` to backfill from the previous time step, assuming that the marginal emissions remain constant if there is no change in the generation level.

3.2 Data Collection

3.2.1 EirGrid Smart Grid Dashboard

Data was sourced from EirGrid which maintains a live dashboard offering a real time overview on various grid parameters including demand, generation, carbon intensity, emissions, interconnection, and frequency at a quarter-hourly resolution. The dataset is publicly accessible and dates back to 2013. A helper function was designed to fetch and compile the data into a Pandas data frame, enabling efficient and iterative collection of the historical information needed. An extract is shown in table 7

Listing 2: Extract of the Python code used to collect the grid data from EirGrid

```
from lib.eirgrid.smart_grid import query_smartgrid

df_emission = query_smartgrid({
    'area': Area.co2emission,
    'datefrom': '2023-12-25',
    'dateto': '2023-12-30',
    'region': 'ALL',
}).sort_values(by='EffectiveTime')
```

A CLI was developed to facilitate the volume of data collection required (see listing 18)

3.2.1.1 Collect This command leverages the function outlined in Listing 1 to collect and store raw datasets for various system parameters and the specified options.

Listing 3: Available options for the CLI collect command

```
Usage: eirgrid_data_collection.py collect [OPTIONS] AREAS...

Collects the historic data from the eirgrid smart grid dashboard

Options:
  -s, --start TEXT         The start date in DD/MM/YYYY  [required]
  -e, --end TEXT           The end date in DD/MM/YYYY
  -r, --region TEXT
  --help                   Show this message and exit.
```

For instance, to gather the historical demand and frequency profiles for all of 2023, use,

Listing 4: Example usage of the data collect CLI command

```
python eirgrid_data_collection.py demandactual, frequency -s=01/01/2023 -e=12/12/2023 -r=ROI
```

The resulting output is stored in multiple `.csv` files, each corresponding to a different 'Area' (i.e., system parameter). This data is later utilised to create the historical system profile dataset for training.

3.2.1.2 Sync This command updates the previously fetched historical datasets with the latest data available from EirGrid. It efficiently updates the datasets by identifying the most recent historical date and retrieving only the new entries since then.

Listing 5: Available options for the CLI sync command

```
Usage: eirgrid_data_collection.py sync [OPTIONS] AREAS...

Fetches and syncs the most up-to-date data.

Options:
-r, --region TEXT [required]
--help Show this message and exit.
```

Listing 6: Example usage of the data sync CLI command

```
python eirgrid_data_sync demandactual, frequency -r=ROI
```

This command functions similarly to the `collect` script, requiring specification of each 'Area' as a different system parameter. The possible values are,

1. demandactual
2. fuelmix
3. generationactual
4. generationforecast
5. windactual
6. windforecast
7. interconnection
8. frequency
9. co2emission
10. co2intensity
11. demandactual
12. demandforecast
13. SnspAll

The datasets collected for this project are (1, 3, 5, 7, 8, 9, 10, 11).

3.2.2 CSO Reports

The Central Statistics Office provides a comprehensive static database detailing metered electricity generation in Ireland. This database offers data at a half-hourly resolution, extending back to January 2020. It categorises fuel consumption into thirteen distinct categories, each representing a different energy source.

1. Battery Storage
2. Biomass/Peat
3. Coal
4. Distillate
5. Gas
6. Renewable Hydro
7. Oil
8. Solar
9. Pumped Storage Hydro
10. Waste
11. Wind
12. Other Non-Renewable
13. Other Renewable

The `.csv` report contains a breakdown of the metered megawatt generation for each fuel category at each thirty minute interval across the three year period available and is freely available online.

3.2.3 SEM-O Market Reports

Data regarding market and pricing dynamics were sourced from the Single Electricity Market Operator (SEMO). SEMO regularly publishes comprehensive reports detailing market trends and pricing information. A similar approach was used for data collection and fetching as EirGrid for gathering information on imbalance pricing. This data is available at a half hourly resolution.

Listing 7: Extract of the Python code used to collect the price data from SEM-O

```
import lib.semo.market_data as market_data
import datetime

df_price = market_data.fetch_market_report(
    report=market_data.SemoMarketReport.Price,
    start_date=datetime.date(2023, 12, 20),
    end_date=datetime.date(2023, 12, 25),
    jurisdiction='ALL',
)
```

A similar CLI was developed to retrieve historical data from SEM-O, including imbalance settlement pricing and other extraneous factors that could influence MEFs. Details of this are provided in the Appendix. However, these factors were not incorporated into the final network design. The available datasets extend back only three months; including these factors would constrain the total number of entries available for training, testing, and validating the network.

3.3 Data Processing

The following steps are taken to prepare the data set for feature selection and training:

1. The historical grid parameters are collected from EirGrid and processed into a single CSV file
2. The historical fuel mix profiles are derived from the raw CSO dataset
3. The EirGrid data set is re-sampled to a 30-minute interval, commensurate to the CSO data set
4. The historical marginal emissions are calculated on the re-sampled dataset using eq. (3.4)
5. The CSO data set is joined with this yielding the historical the grid parameters, marginal emissions and fuel mixes at half-hourly intervals since 2020
6. Feature engineering is applied to this data set to prepare it for feature selection

3.3.1 EirGrid Data Set

A separate script `eirgrid_data_preprocessing.py` was written to merge the individually collected grid parameter dataset `.csv` files into a single file, creating an overview of the historic EirGrid system profile extending back to 2020. An outline of the script is provided, with the full code in listing 19

3.3.1.1 Data Cleaning Functions

- `clean_interconnection()`: Filters the interconnection data for Republic of Ireland, by reformatting the data structure using pivot tables and saving the cleaned data to a `.csv` file.
- `clean_frequency()`: Frequency data recorded at five-minute intervals were re-sampled to fifteen-minute intervals, to align with the other EirGrid system parameters

3.3.1.2 Parallel Data Processing

- `process_file()`: Files were processed in chunks using a thread pool, leveraging multi-threading capabilities. This ensured that large files could be handled without excessive memory consumption.
- `process_chunk()`: Each chunk of data was processed to convert 'EffectiveTime' to a date-time format, and restructured to remove unnecessary attributes and standardise column names.

3.3.1.3 Dataset Merging Each grid parameter file was processed in parallel using a thread pool. The data were merged by EffectiveTime and divided into annual datasets, saved to year-specific files. A complete dataset was created, aggregating all collected system profiles across all years (see table 7)

3.3.1.4 CSO Data Set Historical fuel mix profiles were derived from the raw CSO dataset, which documents half-hourly metered generation, using a dedicated script `cso_fuel_mix_processing.py`.

- The raw file was read (available from data.gov.ie). Missing values were replaced with zero, and data encompassing all time bands were excluded to focus on specific intervals.
- The 'Day' column was converted to a datetime format. An 'Hour' column was extracted from the 'Time Bands' column, which specifies time ranges for metered generation. These elements were combined to form an 'EffectiveTime' column, representing the precise datetime of each entry.
- Redundant columns were removed, and the dataset was restructured to feature a column for each fuel source. Each row corresponds to the metered generation at subsequent half-hour intervals.

The dataset records metered generation in half-megawatt hours. Hence, each value was doubled to align unit measurements with those used in the EirGrid dataset. The script is outlined in listing 20.

3.3.2 Marginal Emissions Calculation

The function outlined in listing 1 was used to calculate the historical marginal emissions on the combined EirGrid system dataset. The dataset was re-sampled from quarter-hourly to half-hourly intervals, to align with the frequency of the fuel mix data set.

Listing 8: Python code used to calculate the historical MEFs at a 30-minute interval

```
df = pd.read_csv(k.CLEANED_DATA_DIR / f'eirgrid.csv', index_col=0, parse_dates=True)
df = df.set_index('EffectiveTime').resample('30T').asfreq()
df = compute_mef(df)
```

3.3.3 Differential Inputs

The fuel mix dataset and the EirGrid dataset were integrated, providing a detailed breakdown of system parameters and fuel-source generation at half-hourly intervals, dating back to 2020. Differential input features were created to measure the changes from one time step to the next (e.g., ΔDemand).

Listing 9: Python code used to create differential input features

```
for col in df.columns:
  df[f'd_{col}'] = df[col] - df[col].shift(-1)
```

In the above code listing, `df` refers to the combined dataset of EirGrid and CSO.

3.3.4 Datetime Encoding

Time data is encoded such that it can be understood by a machine, preserving the cyclical information

$$t_{sin} = \sin\left(\frac{2\pi \times t_{ms}}{86400}\right), \quad t_{cos} = \cos\left(\frac{2\pi \times t_{ms}}{86400}\right) \tag{3.5}$$

A sine/cosine date-time encoding is performed to achieve this and these are stored as two new input features. A helper function was written in python to handle this transformation.

Listing 10: Python code used to perform cyclical date time encoding

```
def dt_encode(df: pd.DataFrame, column: str) -> pd.DataFrame:
  timestamp_s = df[column].map(datetime.datetime.timestamp)
  day_s = 24 * 60 * 60
  features = df.drop(columns=[column])
  features[f'{column}_sin'] = (np.sin(timestamp_s * (2*np.pi/day_s))).values
  features[f'{column}_cos'] = (np.cos(timestamp_s * (2*np.pi/day_s))).values
  return features.dropna()
```

The date time encoding was applied to the combined data set creating two new input features: (1) `EffectiveTime_sin` and (2) `EffectiveTime_cos`. table 8 outlines all the 40 input features post processing, their units and sources.

3.4 Feature Selection

Designing an ANN to forecast MEFs requires finding out the input layers that will be used in the network design. The chosen input features should comprehensively represent the factors influencing MEFs, ensuring that the network can effectively learn and establish the relationship between these inputs and the predicted output.

3.4.1 Empirical Analysis

The empirical analysis in this report involves leveraging domain knowledge to identify and evaluate factors that might influence marginal emissions. The primary factors considered are

- **Fuel Mixes**— the composition of energy sources used for electricity generation at each time step, either renewable (e.g., wind, solar) or non-renewable (e.g., coal, natural gas) sources.
- **Grid Variables**— Grid frequency, demand levels, carbon intensities, emissions, wind generation, total generation, and interconnections.
- **System Changes**— Change in demand, generation, interconnection or fuel mixes will at large reflect changes in the power system that determine which generator is on the margin.
- **Timing**— The time of day may affect marginal emissions given how load distributions vary by the hour and daily seasonality patterns may exist

A total of 40 input features are identified as part of the empirical analysis & prior feature engineering (see table 8 for a full breakdown).

3.4.2 Pearson Correlation

The Pearson correlation measures the linear correlation between two datasets [26]. It ranges from -1 to 1, signalling the strength and direction of linear correlation between two variables.

$$r = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum(x_i - \overline{x})^2 \sum(y_i - \overline{y})^2}} \tag{3.6}$$

A value of 0 means no linear correlation. Conversely, an absolute correlation value $\geq \pm 0.7$ between two variables would indicate a significant positive or negative relationship.

While pearson coefficents can offer some insight into the strength of the relationship between the target and predictor varaibles for MEFs, the inherent non-linearity and complexity associated with thier calculation (refer to sections section 1 to section 2.1.2) retards the assessment. MEFs cannot be easily approximated to a linear relationship involving any single input variable. This is evident in the feature listings in table 8, as no single predictor has a strong positive or negeative correlation.

3.4.3 Weak Predictors

Given that perceptron models represent a linear combination of weighted predictor inputs, see eq. (2.1), features predominantly composed of near-zero values were evaluated for potential exclusion.

It was thought that features which have 75% or more of its values are near-zero likely serve as weak predictors. Their multiplication with the corresponding weights would consistently result in zero values, contributing no meaningful variance or information to the model.

Feature	mean	std	min	25%	50%	75%	max
ΔBattery Storage	-0.0	2.67	-86.95	0.0	0.0	0.0	106.11
ΔSolar	0.01	8.31	-107.38	0.0	0.0	0.0	136.76
ΔOil	-0.0	12.41	-141.37	0.0	0.0	0.0	198.9
ΔDistillate	0.0	9.61	-203.29	0.0	0.0	0.0	136.38
Distillate	6.86	29.47	0.0	0.0	0.0	0.0	323.11
Solar	14.34	46.29	0.0	0.0	0.0	0.0	349.37
ΔOther Non-Renewable	-0.0	0.22	-4.18	-0.0	0.0	0.0	4.36
Battery Storage	0.33	2.89	0.0	0.0	0.0	0.01	141.52
ΔSysFrequency	0.0	0.06	-0.26	-0.03	0.0	0.03	0.26
ΔOther Renewable	-0.0	0.66	-19.94	-0.2	-0.01	0.19	9.71
ΔWaste	-0.01	3.56	-78.38	-0.46	-0.01	0.41	61.92

Table 1: Summary statistics of the weak predictors that were excluded

The threshold value was set to 0.5–if the absolute value of the 75th percentile was less than this, the feature is considered a weak prdictor. The model was designed with and without their inclusion.

3.4.4 Grey's Relational Analysis

Given the complex and non-linear nature of MEFs, traditional linear correlation methods are inadequate, as they cannot aptly handle the dynamic interdependencies and variability inherent in the data. These methods are limited by their assumption of constant, proportional relationships between variables.

To circumvent this GRA is employed [7, 13]. A custom implementation of Grey Relational Analysis was built in Python based off of literature in Sallehuddin et al. [7], shown in listing 21.

Listing 11: Example usage of the Grey Relational Analysis functions from listing 21

```
from lib.common import gra

df = gra.dt_encode(df, column='EffectiveTime') # Encode date-time as sine/cosine waves
X = df.drop('MarginalEmissions')
Y = df['MarginalEmissions']
grg = gra.grg(X,Y,zeta=0.5,norm=True).mean() # Get the average grey relational coefficents
```

GRA excels in situations where data is incomplete or uncertain and relationships are not linear. section 2.4.3 offers a detailed outline of Grey Relational Analysis, and its application in ANN feature selec-

tion. The analysis shown in listing 11 was performed on the prepared data set containing the historical values for the 42 input features and marginal emissions at each time step. The results are shown in table 2.

Feature	GRG
ΔOil	0.98
ΔSolar	0.95
ΔBattery Storage	0.94
ΔCoal	0.91
ΔSysFrequency	0.83

(a) GRG of the 'weak' predictors where $\gamma \geq 0.8$

Feature	GRG
ΔSystemDemand	0.93
ΔCoal	0.91
ΔBiomass/Peat	0.9
ΔPumped Storage Hydro	0.87
ΔWindActual	0.85
ΔGenExp	0.84
ΔCo2Emissions	0.83
ΔCo2Intensity	0.82
GenExp	0.82
ΔInterEwic	0.8

(b) GRG of the 'strong' predictors where $\gamma \geq 0.8$

Table 2: Grey relational degrees associated with each input feature. Only features with $\gamma \geq 0.8$ are shown. The entire table is available in the appendix, see table 8

Generally, $\gamma < 0.6$ indicates a negligible influence, $\gamma \geq 0.8$ indicates a notable influence and $\gamma \geq 0.9$ indicates a marked influence [7]. Considering only input features with a notable influence (as shown in table 2) the dimensionality of the input features from 42 to 15:

1. Change in oil generation
2. Change in solar generation
3. Change in battery storage
4. Change in demand
5. Change in coal generation
6. Change in biomass or peat
7. Change in pumped storage/hydro
8. Change in other non renewables
9. Change in wind
10. Change in net generation
11. Change in frequency
12. Change in CO_2 emissions
13. Change in carbon intensity
14. Net generation
15. Change in ROI interconnection

Of these features, (1-3), (5-7) and 15 have been identified as weak predictors. Two models were built for comparison, one with all 15 input features as outlined, and the other with 10 that exclude the weak predictors.

3.5 LSTM Neural Network

The Long Short-Term Memory (LSTM) design frames the problem as a time-series prediction rather than a function approximation. The LSTM's output is a time series of the MEFs over a distinct time interval and window size (e.g., 30-minute intervals, 24-hour window). This simplifies conceptualising the network's outputs compared to a MLP model. Visualising a N-dimensional function directly is challenging, whereas a forecasted sequence can be easily represented in 2D-space.

LSTMs are typically preferred for such problems given their ability to 'remember' from previous time steps (refer to section 2.3.2 for further details). This makes them particularly adept for forecasting a marginal emissions profile for a given day, based on learning the underlying patterns in the historical dataset. This approach is well-supported in the literature with Maarif et al. [10], Bouziane et al. [11], Amarpuri et al. [12], and Mayes et al. [5] applying LSTM network designs for similar reasons.

3.5.1 Windowing

The LSTM network is trained on a set of sequences. The original time-series data set, where each row represensts a different interval, and each column is a feature, has to be transformed into a suitable data structure [14]. The time-series data must be structured such that each row is a distinct feature, and each

Original			
Time	Feature1	Feature2	Feature3
T_1	a	d	g
T_2	b	e	h
T_3	c	f	i

Transform →

Transformed		
T_1	T_2	T_3
a	b	c
d	e	f
g	h	i

Figure 5: Example of the sequence transformation applied with the data window

column aligns with a uniform time interval. This specific format presents each time step sequentially, forming distinct sequences that capture snapshots of the temporal progression within the data (see fig. 5). The network is trained on these sequences allowing it to discern and predict patterns over time.

The sequences are defined by a window size, signifying the number of time steps included in each. A window size of 24 hours is used for this project, translating to 48 time steps given the half-hourly interval. A `DataWindow` class was written to manage this transformation from raw time-series to structured sequences (see listing 23 for the full implementation)

Each window is corresponds to a sequence, including only the data points that occur within its specific temporal interval. For instance, if there be 48 data points on the 25/12/2023, the resulting sequence for that window would contain 48 points. Conversely, if 26/12/2023 has only 23 data points, the sequence for that day would consist of 23 points.

3.5.2 Standardisation

To improve training performance, both predictors and targets are standardised. Marsland [6] notes that keeping the values of predictors within a reasonable range prevents the model weights from becoming excessively small, thereby enhancing the stability and effectiveness of the learning process.

The z-score transformation adjusts the original dataset to have a mean of zero and a standard deviation of one. Given the disparate scales among the input variables it mitigates potential optimisation challenges stemming from scale imbalances.

The formula for the z-score transformation is,

$$X_{norm} = \frac{X - \mu}{\sigma} \tag{3.7}$$

To revert data back to its original scale (denormalisation), the inverse transformation of the z-score is,

$$X = \sigma X_{norm} + \mu \tag{3.8}$$

Implementation of this standardisation process has been developed in MATLAB. The StandardScaler is employed to fit the scaler to a dataset, setting the values of μ and σ. The normalisation and denormalisation are subsequently applied to new data points. The complete implementation of the class is provided in listing 22.

3.5.3 Data Preparation

The data sets were processed in Python and imported into MATLAB. The Deep Learning Toolbox makes it easier to build, train, and test artificial neural networks [23]. The toolbox was chosen for its comprehensive set of features and robustness, which surpass similar tools like Python's Keras. This made it the preferred choice to build and refine the ANN models.

Listing 12: Example usage of the DataWindow class for sequence formation

```
% Read in the data sets .csv files that were processed in Python
weakdata = readtable('../data/processed/final_data_set_weak.csv');
strongdata = readtable('../data/processed/final_data_set_no_weak.csv');

% Setup the data window including the "weak" predictors
windowSize = 24; % 24-hour data window
inputfeatures = weakdata.Properties.VariableNames;
inputfeatures = setdiff(inputfeatures, {'EffectiveTime', 'MarginalEmissions'});
weakdw = DataWindow(weakdata, windowSize, 'EffectiveTime');
weakdw = weakdw.gseq(inputfeatures,'MarginalEmissions');

% Setup the data window excluding the "weak" predictors
inputfeatures = strongdata.Properties.VariableNames;
```

```
inputfeatures = setdiff(inputfeatures, {'EffectiveTime', 'MarginalEmissions'});
strongdw = DataWindow(strongdata, windowSize, 'EffectiveTime');
strongdw = strongdw.gseq(inputfeatures,'MarginalEmissions');

%% Setup the training data sequences
Xw_Scaler = StandardScaler().fit(weakdw.Inputs);
Yw_Scaler = StandardScaler().fit(weakdw.Targets);
Xw = Xw_Scaler.normalize(weakdw.Inputs);
Yw = Yw_Scaler.normalize(weakdw.Targets);
tw = traintestval(Xw, Yw, [0.5, 0.3, 0.2]);

Xs_Scaler = StandardScaler().fit(strongdw.Inputs);
Ys_Scaler = StandardScaler().fit(strongdw.Targets);
Xs = Xs_Scaler.normalize(strongdw.Inputs);
Ys = Ys_Scaler.normalize(strongdw.Targets);
ts = traintestval(Xs, Ys, [0.5, 0.3, 0.2]);

%% Save to .mat file
save('lstm_15.mat', 'Xw_Scaler', 'Yw_Scaler', 'weakdw', 'weakdata', 'Xw', 'Yw', 'tw')
save('lstm_10.mat', 'Xs_Scaler', 'Ys_Scaler', 'strongdw', 'strongdata', 'Xs', 'Ys', 'ts')
```

The following preliminary steps were taken before designing the LSTM network,

1. The datasets were processed using Python and subsequently loaded into MATLAB. As detailed in section 3.4.4, two distinct datasets were prepared: one with 10 input features (.i.e. the 'strong' dataset) and another with 15 input features (.i.e. the 'weak' dataset).
2. Sequences were formed from the input data using the windowing technique outlined in section 3.5.1.
3. The sequences were normalised using the standardisation process described in section 3.5.2.
4. The sequences were divided into training, testing, and validation sets with a distribution of 50%, 30%, and 20% respectively. The testing dataset, scaler, and the original dataset were saved in a MATLAB .m file for subsequent use.

3.5.4 Network Design

The following steps were taken to arrive at the final network design

1. The model data was retrieved from the .m file after processing, as previously described.
2. Network configuration and hyperparameters were established, followed by training using a random selection of 50% of the total available dataset.
3. Early stopping was employed to determine the optimal number of training epochs.
4. The network's performance was evaluated using the Root Mean Squared Error (RMSE) on the test dataset, providing a quantifiable measure of accuracy in the same units as the MEF

Listing 13: MATLAB code used to prepare the Python datasets for LSTM network training

```
% Load in the model data
x = load('lstm_10.mat');

% Setup training variables
ts      = x.ts;
X       = x.Xs;
XTrain  = ts.XTrain;
XTest   = ts.XTest;
XVal    = ts.XVal;
YTrain. = ts.YTrain;
YTest   = ts.YTest;
YVal    = ts.YVal;
YScaler = model.Y_Scaler;

% Hyper parameters
numResponses = size(YTrain{1},1);
numFeatures  = size(X{1},1);
numLSTMUnits = 128;
numHiddenUnits = 50;
dropOutRate = 0.5;

% Design the network by specifying each individual layer
layers = [ ...
    sequenceInputLayer(numFeatures)
    fullyConnectedLayer(numLSTMUnits*2)
    lstmLayer(numLSTMUnits,'OutputMode','sequence')
    fullyConnectedLayer(numHiddenUnits)
    dropoutLayer(dropOutRate)
    fullyConnectedLayer(numResponses)
    regressionLayer];

% options = ....
% Train the network
net = trainNetwork(XTrain,YTrain,layers,options);
```

Here is a breakdown of each of the network layers shown in listing 13

- The `sequenceInputLayer` in the network is the input layer for the training sequences.
- The first `fullyConnectedLayer` expands the dimensionality of the input data. It's used to introduce additional parameters that can capture more complex relationships in the data before it enters the LSTM layer. It helps in learning non-linear transformations of the input data.
- The `lstmLayer` is the core of the network and introduces the LSTM cells. The sequence output mode means the layer outputs a complete sequence, rather than the last data point.
- The second `fullyConnectedLayer` helps refining the outputs from the LSTM layer, consolidating the learned information into a representation that can be effectively used to predict the final output.
- The `dropoutLayer` randomly sets a fraction of the input units to 0 at each update during training time, which helps prevent overfitting.

- The final `fullyConnectedLayer` transforms the learned features into the actual predictions. It ensures that the output has the appropriate dimensionality for the problem at hand.
- Last, the `regressionLayer` computes the loss for a regression problem using RMSE

3.5.5 Hyperparameters

No specific empiricial methods were used to determine the optimal hyperparamters. The standard configuration was taken as a baseline and tweaked slightly for performance.

Listing 14: MATLAB code used to configure the hyperparameters of the LSTM network design

```
options = trainingOptions('adam', ...
    'MaxEpochs',1000, ...
    'MiniBatchSize',20, ...
    'InitialLearnRate',0.020, ...
    'Shuffle','every-epoch', ...
    'Plots','training-progress',...
    'ValidationData',{XVal,YVal},...
    'ValidationFrequency',10, ...
    'ValidationPatience',20, ...
    'OutputNetwork', "best-validation", ...
    'Verbose',0);
```

- The default learning rate for the Adam optimiser is set at 0.01. To expedite the training process and help the network avoid converging to a sub-optimal minimum, this rate was increased to 0.02
- The batch size was decreased from 128 to 10. The smaller batch size introduces more noise during training. This noise can help the network escape local minima and find a more robust path toward the global minimum, leading to better generalisation on unseen data.
- The training sequences were shuffled after each epoch to prevent order bias. If the sequence order remains constant throughout training, the model might inadvertently learn patterns related to the sequence order rather than the underlying relationships in the data.
- As well, shuffling the training sequences challenges the network to maintain its ability to recognise temporal dependencies under varying input conditions.

The optimal number of epochs are determined using early stopping, which is implemented in listing 14 by the following parameters [23]

- `ValidationFrequency` — specifies the number of iterations between evaluation of validation loss (.i.e. the delay between subsequent validation checks)
- `ValidationPatience` — specifies the number of times that the validation loss can be greater than or equal to the previous best value before the network stops training

The parameters were set at 10 and 100 respectively

- Each 10 iterations the network is validated by measuring the RMSE of the predicted output from data not used in training (.i.e. from the validation set)
- If after 100 validation checks the RMSE is at or above the previous lowest value the network has stopped learning and the training is haulted

3.5.6 Evaluation

The model will undergo testing using data that it has not encountered during the training phase. The dataset will be split into two sets: 70% for training and validation, and the remaining 30% for testing. This 70/30 split reduces the risk of developing a bias towards the data used in training and ensuring a more objective assessment of its performance.

Root Mean Squared Error (RMSE) is used as the performance metric during testing and training. RMSE is particularly suited for evaluating the performance of LSTM networks given its ability to maintain the error units consistent with the units of the data.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2} \tag{3.9}$$

$$\text{Overall RMSE} = \sqrt{\frac{1}{\sum_{k=1}^{N} n_k}\sum_{k=1}^{N}\sum_{i=1}^{n_k}(Y_{k,i} - \hat{Y}_{k,i})^2} \tag{3.10}$$

- eq. (3.9) is the equation for evaluating the RMSE of a given sequence.
- eq. (3.10) calculates the overall RMSE across all sequences combined.

The square root expresses the result in the data's original magnitude.

Listing 15: MATLAB code use to calculate the RMSE of the LSTM network

```
% Predict and denormalise the responses
yTest = predict(net, XTest);
yTest_norm = YScaler.denormalize(yTest);
YTest_norm = YScaler.denormalize(YTest);

% Calculate RMSE for each sequence
for i = 1:size(YTest,1)
    rmse(i) = sqrt(mean((yTest_norm{i} - YTest_norm{i}).^2,"all"));
end

% Calculate the overall RMSE
residuals = zeros(size(Y, 1), 1);
for i = 1:size(YTest,1)
   residual = yTest_norm{i} - YTest_norm{i};
   residuals = [residuals; residual(:)];
end
overallRMSE = sqrt(mean(residuals.^2));
```

4 Results

4.1 Historical Emissions Factors

The historical MEFs are derived from the dataset detailed in table 7, which encompasses data from as early as 2020, presented at quarter-hourly intervals. Python is used for this computation. The helper function in listing 1 incorporates the prescribed methodology in eq. (3.4) to calcualte the historical MEFs

Marginal Emissions Heat Map fig. 15 is a heat map illustrating the historical half-hourly marginal emissions for the years 2020, 2021, 2022, and 2023, based on the calculated dataset. While the visual patterns across these years appear relatively consistent, a distinct lack of clear seasonality or predictable trends is evident. This observation underscores the erratic nature of MEFs within Ireland's electricity sector, reflecting the inherent variability of the power grid.

Such visual analysis reinforces the complexity of the underlying data and the challenges in predicting MEFs using traditional methods. It emphasises the potential of an ANN to discern and learn the intricate rules governing this data. The intricacy and irregularity evident in the heat map substantiate the decision to employ an ANN for forecasting marginal emissions, as simpler statistical models are likely inadequate for capturing these complex relationships.

Fuel Generation vs Marginal Emissions fig. 16a presents a detailed analysis of fuel generation changes based on the CSO report data, set against a stem plot of marginal emissions for a specific day, October 15th, 2023. Notably, at 5am and 4pm, marginal emissions peak at over 2000 gCO_2/kWh. These spikes coincide with increased non-renewable (primarily gas) generation and a simultaneous decrease in renewable energy output. Conversely, at 12pm, there is a marked drop in marginal emissions, which follows a period of increased renewable generation and reduced non-renewable output.

This day's marginal emissions profile mirrors the changes in fuel composition at each 30-minute interval. While this pattern is evident here, it is important to recognise that it does not consistently apply to every day. This variability further reinforces that more factors beyond categorical fuel consumption influence marginal emissions.

Average vs Marginal Emissions fig. 16b depicts a 365-day rolling average comparison of marginal emissions against average emissions (i.e., carbon intensity) for the four-year period in question. The graph reveals a notable divergence in the trends between average and marginal emissions. Specifically, while both marginal and average emissions exhibited an upward trend in 2021, a distinct pattern emerges from

mid-2022 to early 2023: average emissions show a declining trend, whereas marginal emissions continue to rise.

This phenomenon, highligthed in Holland et al. [9], underscores the critical relevance of marginal emissions in the context of climate action policy. Relying solely on average emissions fails to provide a comprehensive perspective, potentially obscuring significant trends and insights crucial for informed policy-making.

4.2 Relational Analysis

The Grey Relational Analysis (GRA) applied to the 2020-2024 dataset indicates that the following set of features exerts a significant influence on the MEFs

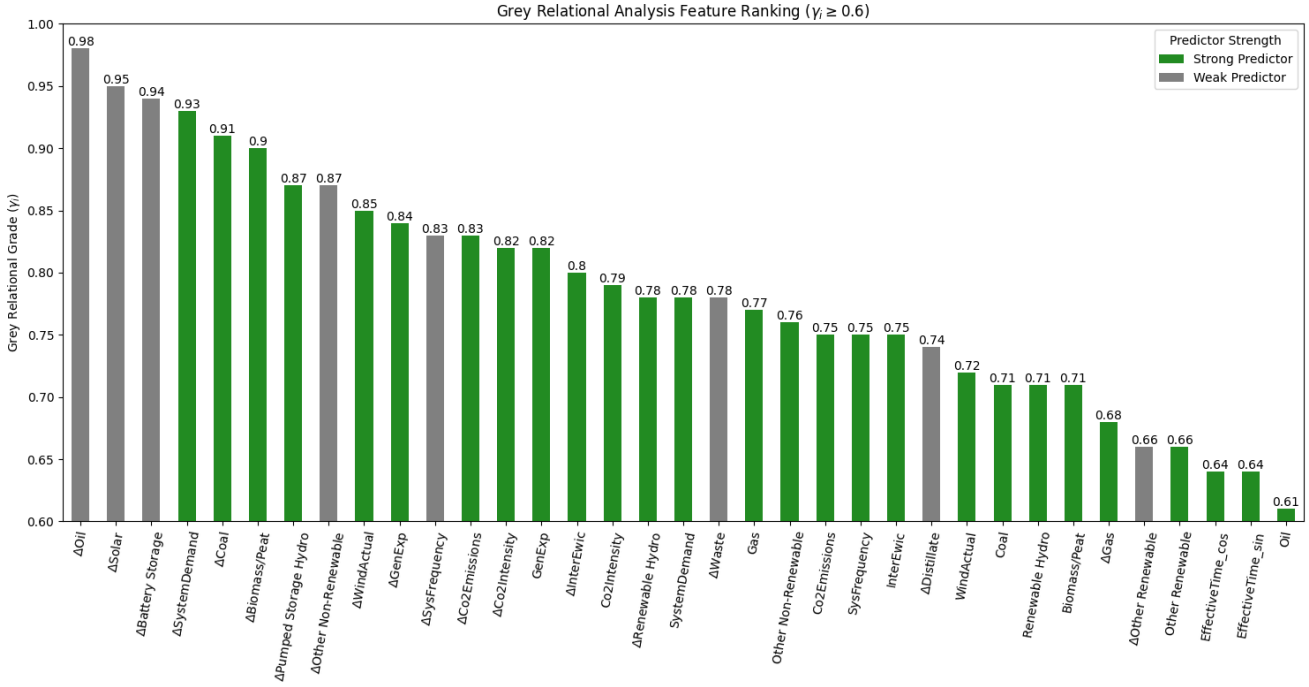


Figure 6: Grey Relational Grade (GRG) where $\gamma_i \geq 0.6$ based on a GRA on the entire 2020-24 dataset

- Variations in non-VRE generation sources such as oil, solar, coal, biomass, and other
- Changes in electricity demand
- Changes in VRE generation, including hydro and wind
- Overall shifts in total electricity generation
- Changes in grid frequency
- Changes in carbon intensities or emission level
- The net level of electricity generation
- Changes in net interconnection

Application of GRA using lag-1, lag-2, and lag-3 input features shows near-identical results in terms of the grey relational degrees for the input variables. For this reason lagged input features aren't considered.

As shown in fig. 6, time of day appears to have a relatively minor impact on marginal emissions compared to other variables. This suggests that factors influencing the choice of the marginal operator extend

beyond mere temporal patterns, evident from the initial observations on the historical MEFs.

Considering only features that are not 'weak' (see section 3.4.3), the most influential set of inputs are,

1. Δ Demand
2. Δ Coal
3. Δ Biomass
4. Δ Hydro
5. Δ Wind
6. Δ Generation
7. Δ CO_2 Emissions
8. Δ CO_2 Intensity
9. Net generation
10. Δ Net interconnection

The factors identified through the application of GRA align with the empirical analysis in section 3.4.1,

- **Changes in non renewable generation** — These sources are typically relied upon to compensate for any shortfall in VRE output. Notably, conventional generation seems to exert a stronger influence on marginal emissions than renewable sources.
- **Changes in demand** — Perturbations in demand directly affect marginal emissions. An increase in demand, especially when VRE output is low, often necessitates the ramping up of conventional generation.
- **Changes in VRE generation** — As mentioned above, fluctuations in VRE output tend to push conventional generators on the margin. Given that Wind and Hydro are Ireland's primary renewable energy sources, these results are as expected.
- **Changes in generation** — These exert a direct influence on MEFs as they ascertain which generators operation on the margin
- **System frequency** — Frequency response mechanisms include bringing fast-ramping, often fossil fuel-based, generators online to stabilise the frequency
- **Changes in CO_2 emissions or intensities** — The GRGs for these input features are roughly the same, since the carbon intensity is simply the total CO_2 emissions over generation, as per eq. (3.2). As well, changes in CO_2 intensity are directly linked to the MEF calculation method in eq. (3.4).
- **Net generation** — Power systems use different types of generation units: base load (often coal), which operate continuously to meet constant demand, and intermediate or peaking units (like natural gas turbines), which handle higher demands or supplement fluctuating renewable outputs. The level of net generation dictates the contribution of each type of unit.
- **Interconnection** — Provides access to cleaner energy sources from neighbouring grids or requiring the local grid to rely on higher-emission generation during high demand/low VRE output

4.3 LSTM Network

The best performing LSTM network design is presented. Two models were built

1. A 10-input network considering only the 'strong' predictors
2. A 15-input network considering both the 'strong' and 'weak' predictors

The input features were selected through the appliation of GRA as outlined in section 3.4.4 and section 4.2

Feature	Description	Unit	Source	GRG	
ΔOil	Change in Energy from Oil	MWH	CSO	0.98	weak
ΔSolar	Change in Solar	MWH	CSO	0.95	weak
ΔBattery Storage	Change in Energy from Battery Storage	MWH	CSO	0.94	weak
ΔSystemDemand	Change in Total Energy Demand	MWH	EirGrid	0.93	strong
ΔCoal	Change in Energy from Coal	MWH	CSO	0.91	strong
ΔBiomass/Peat	Change in Energy from Biomass	MWH	CSO	0.9	strong
ΔPumped Storage Hydro	Change in Pumped Hydro	MWH	CSO	0.87	strong
ΔOther Non-Renewable	Change in Energy from Non-Renewables	MWH	CSO	0.87	weak
ΔWindActual	Change in Wind	MWH	EirGrid	0.85	strong
ΔGenExp	Change in Generation (.i.e. deltaP)	MWH	EirGrid	0.84	strong
ΔSysFrequency	Change in System Frequency	Hz	EirGrid	0.83	strong
ΔCo2Emissions	Change in Total Carbon Emissions	tCO_2/hr	EirGrid	0.83	weak
ΔCo2Intensity	Change in Average Emissions	gCO_2/kWh	EirGrid	0.82	strong
GenExp	Total Energy Production	MWH	EirGrid	0.82	strong
ΔInterEwic	Change in Interconnection between ROI/Wales	MWH	EirGrid	0.8	strong

Table 3: Summary table of the 15 input features considered. This is a modified extract from table 8

Both networks were configured with the same design

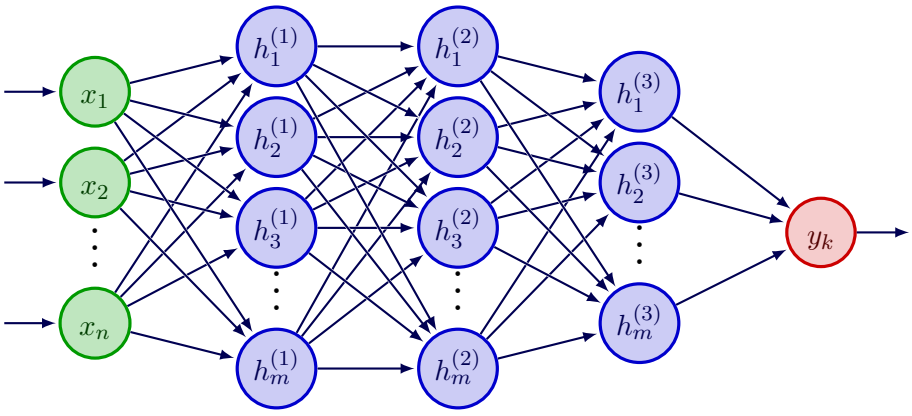


Figure 7: Representative topology of the final network design

There are five total layers in the network,

1. A sequence input layer with 10 neurons
2. A hidden layer with 128 neurons
3. A LSTM layer with 128 neurons

4. A hidden layer with 50 neurons
5. A single output layer

Between the layers (2-3), (3-4) are dropout layers, with a 50% dropout rate. These randomly set the input units to zero, 'dropping out' nodes temporarily to prevent overfitting. This is implemented in listing 13.

The best results were arrived at when the 10-input network was trained for 10,160 iterations, with a scaled training RMSE of 0.44 validation RMSE of 0.47. 59,014 samples were used to build the network, spanning four-years of historical data on system parameters and calculated marginal emissions.

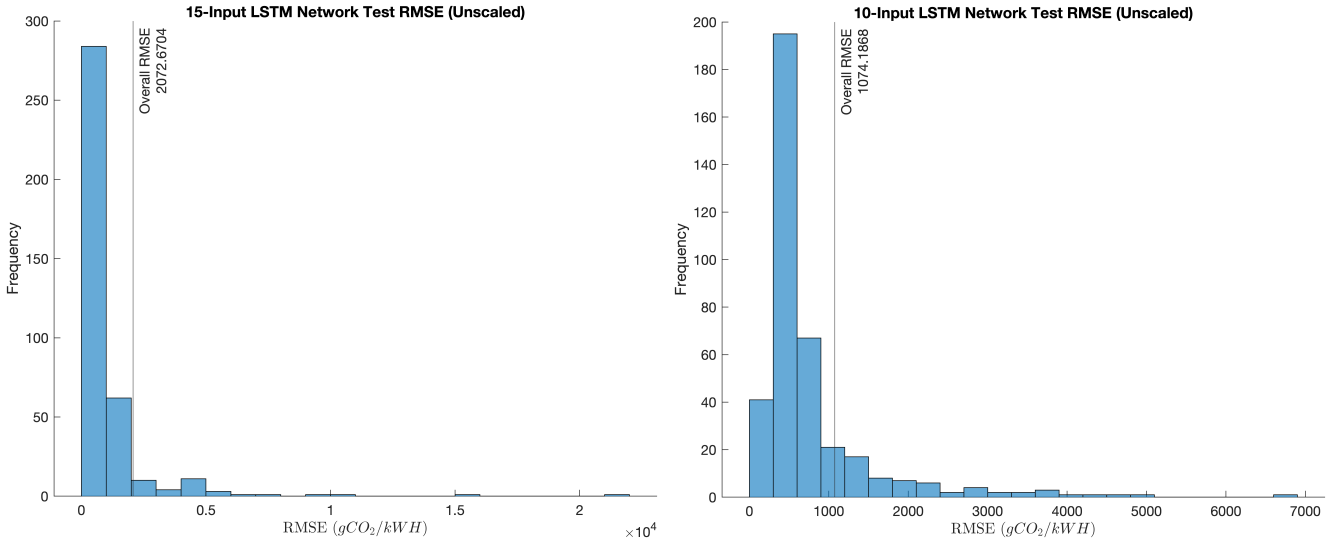
From this, 1,269 sequences were formed using a 24-hour data window. The sequences varied in length, but corresponded to a 48-time step interval, given each half-hourly sample. The sequences were split into,

- 634 training sequences (50% of the total)
- 380 test sequences (30% of the total)
- 255 validation sequences (20% of the total)

The sequences were split at random, and shuffled during training to prevent overfitting. The same set of sequences was used for all experimentation and iteration on both network designs.

4.3.1 Performance

Figure 8 illustrates the scaled Root Mean Squared Error (RMSE) distributions in gCO_2/kwH of the predicted sequences for both LSTM models. A lower RMSE indicates a higher prediction accuracy.



(a) 15-input network **(b)** 10-input network

Figure 8: RMSE in gCO_2/kWh for the 15-input and 10-input LSTM networks

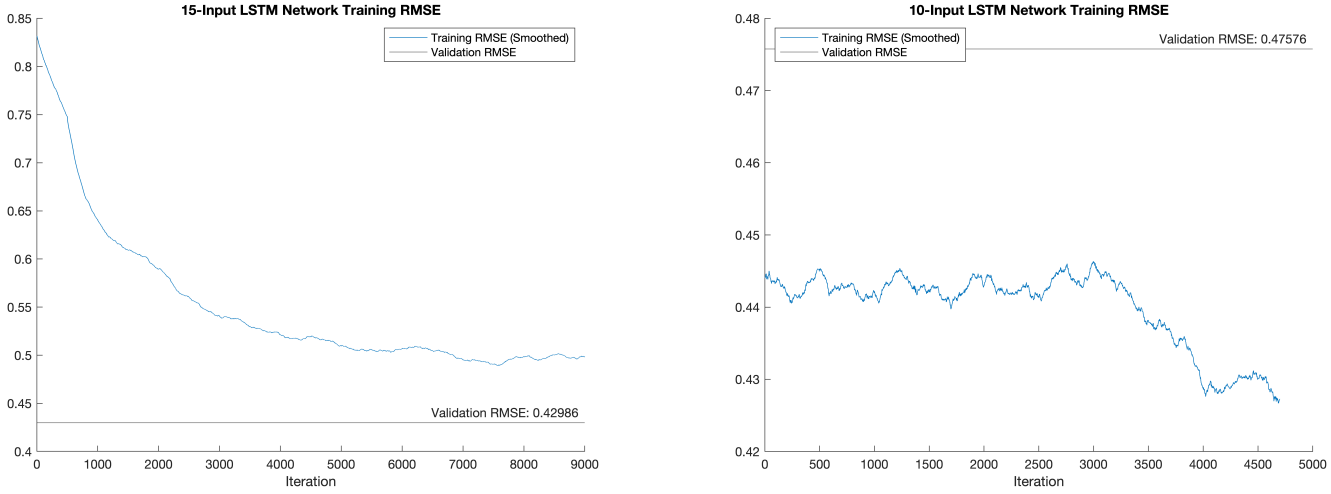
The RMSE distribution of the 15-input network, shown in fig. 8a, had an overall RMSE of 2072 gCO_2/kwH. This is roughly 92% higher than the 10-input network. The error distribution, primarily clustered near

the lower end, demonstrates some degree of predictive consistency.

In contrast, the 10-input network, built solely with 'strong' predictors, achieved an overall RMSE of 1074 gCO_2/kWh, showing a smaller average error across its predictions. The distribution of RMSE values for the 10-input network is much narrower, ranging between $0 \rightarrow 7000$, versus $0 \rightarrow 20000$, which implies a more consistent prediction performance with fewer outliers.

An explanation for the higher RMSE in the 15-input network could be the inclusion of the 'weak' predictors. Given they exist at a near-zero operating point, for most input sequences, they lack meaningful contributions, and manifest as noise. However, this does not imply that these 'weak' predictors lack influence on the Marginal Emissions Factors.

Neural networks aggregate input values and their corresponding weights. When inputs predominantly approach zero, they might seem unnecessary. However, in practical scenarios, non-zero inputs can exert considerable influence. Even a slight deviation from their operating point can have massive impacts. But the relative numerical insignificance cannot be easily captured in a perceptron model.



(a) 15-input network **(b)** 10-input network

Figure 9: Standardised training RMSE for the 15-input and 10-input LSTM network

Figure 9 shows the training and validation RMSE of the 15 and 11-input LSTM networks. These curves have been smoothed to reduce noise and give a better representation of the overall trends during training.

For the 15-input network, shown in fig. 9a,

- The training error starts noticeably high but exhibits a rapid decrease, suggesting significant learning early in the training process. This could show the initial weights not being optimal, hence the model learns quickly to correct its predictions.
- The validation RMSE lying below the training RMSE could suggest that the network generalises well, although it may also raise concerns about underfitting if the training RMSE does not decrease

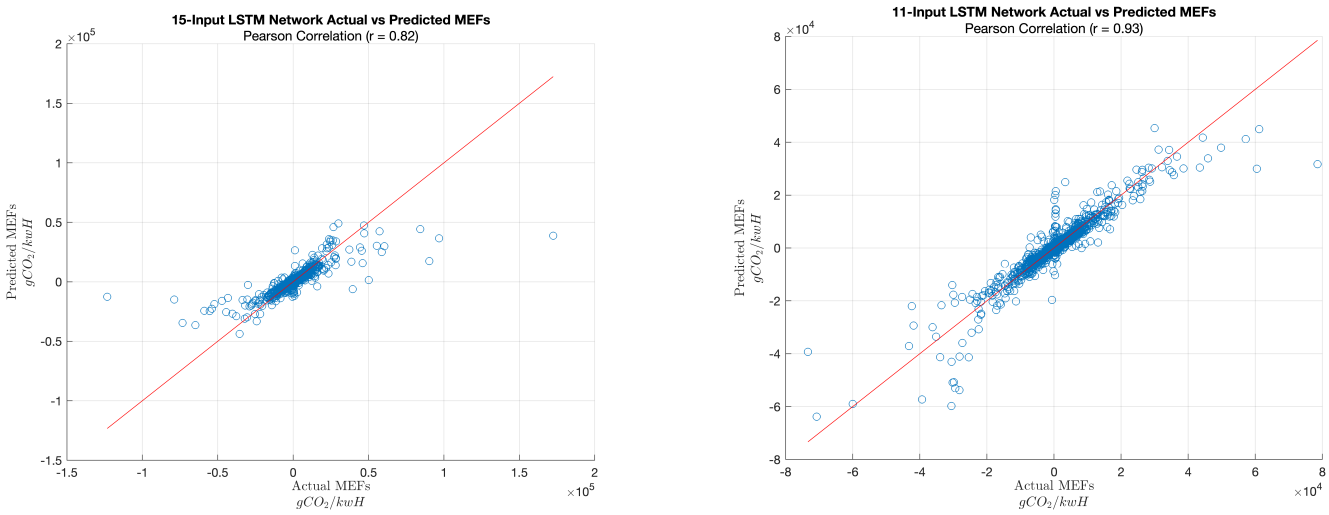
further or over-smoothing during the validation loss computation.

For the 10-input network, shown in fig. 9b

- The training error is much narrower in range. This suggests that the network's predictions are more consistent from the beginning. This could imply a more effective random initialisation of weights. The removal of less informative, 'weak' inputs appears to have streamlined the learning process.
- The validation RMSE is higher than the training RMSE. This signifies that the model is fitting the training data better than it is generalising to new, unseen data, which could be a sign of overfitting. It is possible that the `ValidationPatience` or `ValidationFrequency` parameters set in listing 14 were too high, hindering the effectiveness of the early stopping measures.

4.3.2 Predictions

The sequences were flattened from a 3D to 2D-shape, encompassing the predictions across all time steps. In fig. 10, the predicted values of both networks were plot against each other and a regression line was fit through to understand the general trend. The Pearson Correlation eq. (3.6) was calculated to assess the strength between the predicted and reference emission factors.



(a) 15-input network **(b)** 10-input network

Figure 10: Correlation between the inputs and outputs of the 11-input LSTM network

For the 15-input network, shown in fig. 9a,

- The correlation r value is 0.82, indicating a strong positive correlation between the actual and predicted values. However, the spread of the points show variability in the predictions.
- Some points deviate significantly from the regression line of the network. What this means is, while the network has captured the general trend in the data, it struggled with certain predictions.
- Given the erratic nature of marginal emissions factors, where at one time step there can be a

significant increase or decrease, this is expected. Neural networks typically rarely learn extrema, as during the generalisation process this is discarded as 'noise'. However, in this project these contain crucial information.

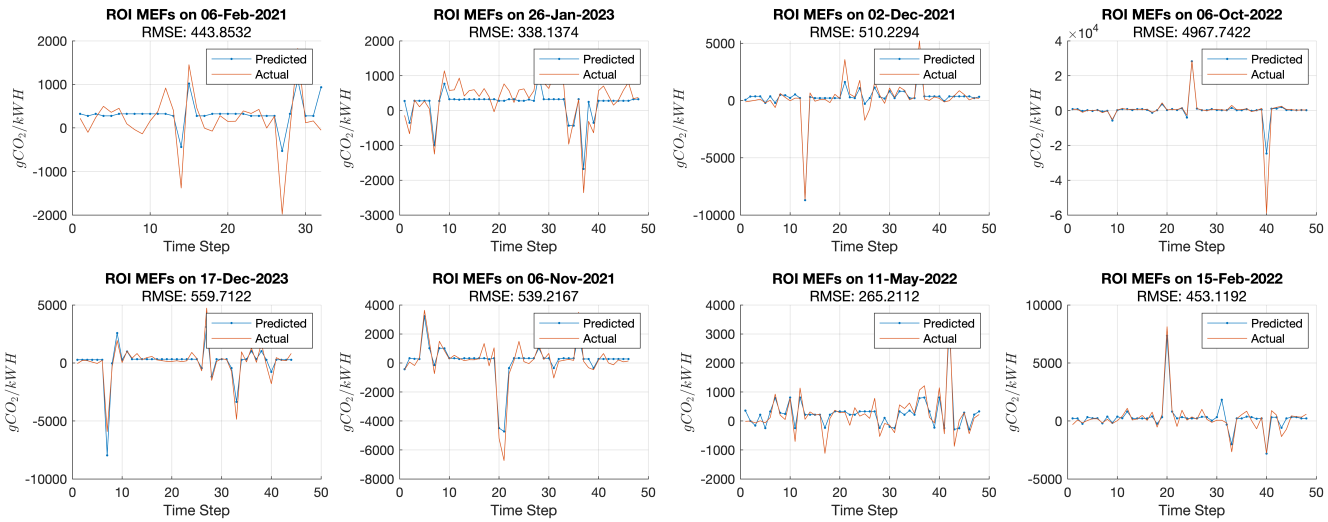
For the 10-input network, shown in fig. 9b

- Here, the correlation value increases significantly to 0.93, reflecting a very strong positive relationship. This shows that the predictions are more consistently aligned with their actual values.
- The closer clustering of points around the regression line implies that the network is more accurate and less variable in its predictions, when compared to the 15-input network. This could be attributed to the exclusion of the 'weak' predictors, which likely has made the model less prone to noise and more focused on the most informative features.

The removal of the 'weak' predictors in the 10-input network overall appears to have a positive impact on the prediction accuracy, as evidenced by the lower RMSE and increased positive correlation. In perceptron models, where the weighted sum of inputs is key to the output, removing inputs with negligible influence can streamline the network's learning process and improve the clarity of the signal it is modelling.

The 15-input network, while still performing well, shows that additional features do not equate to better predictions. It emphasises the importance of feature selection in model design and the potential benefits of simplifying the model to focus on the most impactful inputs.

However, as mentioned before, this does not necessarily mean those 'weak' features are redundant in a real world context. They Grey Relational Analysis still interprets these are features with a significant influence on MEFs, it could just be this influence cannot be properly captured in a neural model.



(a) 15-input network **(b)** 10-input network

Figure 11: Sample predictions of the 15-input and 10-input LSTM network

fig. 11 produces sample predictions for both networks. These sequences were chosen at random from

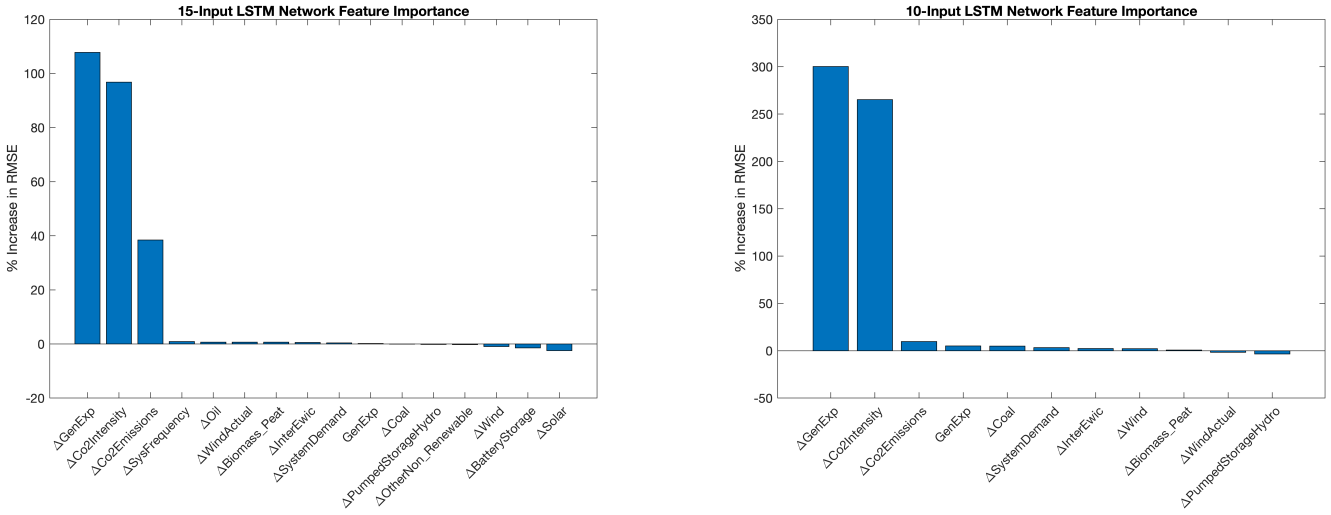
the testing data set, which means the network was not exposed to this during the training process. The models seem to capture the general trend of the MEFs reasonably well on most days, but struggles with the sudden spikes in the data.

There is a noticeable variation in the RMSE across different dates, which suggests that the model's performance is not uniform over time. For example, in fig. 11b, on February 15th, 2022, there is a tight convergence between the predicted and actual lines, with a relatively low RMSE of 453 gCO_2/kWH. But on October 6, 2022, the RMSE is high, at nearly 5000. This could be influenced by the underlying data distribution on different days or extraneous factors that are not considered.

Given that some predictions are quite close to the actual values while others are not, there might be a risk of overfitting to certain patterns in the data. This overfitting might not generalise well to all days, resulting in variable performance.

4.3.3 Feature Importance

fig. 12 demonstrates the feature importance determined by the permutation method. The importance is measured by the percentage increase in RMSE when each feature's values are shuffled across time steps.



(a) 15-input network **(b)** 10-input network

Figure 12: Calculated feature importance of the 15-input and 10-input LSTM network

The analysis can be understood as identifying which feature, when perturbed, causes the most significant increase in prediction error. This is not equivalent to pinpointing the most influential factor directly, yet it serves as a useful tool to better understand the model's behaviour. Notably, changes in carbon intensity and generation are prevalent in both networks. This suggests that the model has learned the underlying pattern in the data, given that both values are used in the calculation method in eq. (3.4). So, as expected, when these change they cause the largest increase in error. listing 24 outlines this computation in MATLAB.

5 Discussion

fig. 17 shows the 10-input LSTM predicted vs actual Marginal Emissions Factors as well as the input sequences. There are several notable peaks and throughs in the graph,

- There is a significant spike in marginal emissions at time step 2, .i.e. 12:30 AM, to 3000 $\frac{\text{g CO}_2}{\text{kWH}}$. The residual is roughly 1000 $\frac{\text{g CO}_2}{\text{kWH}}$, a relatively high degree of inaccuracy. At this point,

  - ΔGen = 11 MW h
  - ΔDemand = 103 MW h
  - ΔCO_2 Intesnsity = 8 $\frac{\text{g CO}_2}{\text{kWH}}$
  - ΔCO_2 Emissions = 29 $\frac{\text{t CO}_2}{\text{kWH}}$
  - ΔWind = -14 MW h
  - ΔBiomass = 1.06 MW h
  - ΔCoal = -2.03 MW h

  There is an increase in demand, met with a fall in VRE output. Here, the wind generator is on the margin and a conventional biomass/peat generator is used to bridge the shortfall. Other fuel sources may contribute to this that are not considered in the model. The result is a spike in emissions.

- Another notable spike occurs at time step 17, to roughly 3000 $\frac{\text{g CO}_2}{\text{kWH}}$. At this point,

  - ΔGen = -8 MW h
  - ΔDemand = -56 MW h
  - ΔCO_2 Intesnsity = -6 $\frac{\text{g CO}_2}{\text{kWH}}$
  - ΔCO_2 Emissions = -24 $\frac{\text{t CO}_2}{\text{kWH}}$
  - ΔWind = -46 MW h
  - ΔBiomass = -1.62 MW h
  - ΔCoal = -1.24 MW h

  There is a fall in demand which is commensurate to the fall VRE output. Average emissions levels have fallen, as have conventional generation, but there is still a significant spike. This suggests that there is an extraneous factor contributing. As well, the residual value remains high.

- The last notable spike is at time step 35. Here, marginal emissions fall to -1900 $\frac{\text{g CO}_2}{\text{kWH}}$ and the residual value remain high, with a predicted fall of -900 $\frac{\text{g CO}_2}{\text{kWH}}$

  - ΔGen = 15 MW h
  - ΔDemand = -93 MW h
  - ΔCO_2 Intesnsity = -9 $\frac{\text{g CO}_2}{\text{kWH}}$
  - ΔCO_2 Emissions = -29 $\frac{\text{t CO}_2}{\text{kWH}}$
  - ΔWind = 73 MW h
  - ΔHydro = -84.52 MW h
  - ΔBiomass = 0.42 MW h
  - ΔCoal = 0.92 MW h

  Again, demand levels fall. Generation rises, which appears to be attributed to the increase in wind output. Average emission has also fallen, while conventional generation has increased marginally. There is a significant decrease in hydropower, which appears to cover for the decrease in demand.

5.1 Challenges & Limitations

The results depicted in fig. 17 illustrate the LSTM model's ability to detect broad patterns in MEF, as evidenced by notable peaks and troughs aligned with input fluctuations. These outcomes confirm the model's sensitivity to dynamic changes in input variables such as wind generation and demand shifts. However, significant residuals, such as a roughly 1000 $\frac{\text{g CO}_2}{\text{kWH}}$ discrepancy at 12:30 AM, suggest potential overfitting, or underfitting, showing limitations in the model's ability to generalise.

Peaks in predicted marginal emission that do not align with clear changes in input features hint at the model's inability to capture certain extraneous factors or complex nonlinear dynamics, like at time step 35. This might require the incorporation of additional variables or a change of model parameters to better account for external influences like market conditions or weather patterns

5.1.1 Dimensionality

However, including additional variables to account for extraneous factors introduces significant challenges related to the dimensionality of the data. High-dimensional datasets cause the feature space becomes so large that the data are sparse and insufficient to train a model effectively. This sparsity increases the risk of overfitting, as the model may learn noise instead of underlying patterns.

High-dimensional data sets inherently complicate the training process. As the number of dimensions increases, the volume of data needed to fill the feature space exponentially grows. High dimensionality can drastically increase the computational load, leading to longer training times. As well, they make the model less practical to use. Visualising a high-dimensional dataset is difficult, if not impossible, and working with a model that requires 20 inputs is much harder than one that requires 2 or 4. The most user-friendly models will have as few inputs as possible.

Dimensionality can be addressed by ensuring the feature selection process only picks features that are strongly correlated to the target, and uncorrelated to each other. As well, compression methods like Principal Component Analysis (PCA) can create linear combinations of the input features that compress them to lower dimensions, while retaining the variance and important characteristics of the data.

Issues with high-dimensional networks are more pronounced for MLP networks than LSTM networks, which process sequences through recurrent structures. This effectively manages high-dimensional inputs by utilising the same weights across time steps, thus reducing detrimental impact.

5.1.2 Underfitting

Underfitting in neural networks occurs when the model cannot capture the underlying pattern of the data, leading to inadequate performance on both training and unseen data. This often results from a model that

is too simple relative to the complexity of the data or from overly conservative learning parameters that prevent the model from learning significant trends, including outliers, which with MEF, are a substantial part of the environmental data dynamics.

Despite these challenges, the general aim is to ensure that the network learns the general shape or trend of the emissions data without being swayed by noise, which can sometimes lead to overlooking crucial variations, like spikes at specific times.

5.1.3 Overfitting

Overfitting presents a significant challenge, especially evident when a model learns the details in the training data to an extent that it negatively impacts the performance on new data. This often manifests through an excellent fit to the training data but poor validation performance, as detailed in section 5.2.1.

Overfitting not only reduces the model's ability to generalise but also can lead to misleading conclusions about the factors influencing marginal emissions, thus complicating the decision-making process in policy formulation and energy management.

5.1.4 Adressing Limitations

To address these limitations, the following steps could be taken to refine the LSTM network:

- Implement regularisation techniques or change the network architecture to reduce overfitting.
- Increase the robustness of the model to fluctuations in input data through enhanced preprocessing, feature engineering and feature selection. Run more experiments using different combinations of input features to determine the most optimal set.
- Expand the dataset to include additional variables that capture extraneous factors affecting emissions, such as the market pricing data from SEMO, or even weather data that could relate to VRE output. However, this is quite tricky as there are infinite avenues to explore, and including too many inputs will lead to the challenges associated with dimensionality
- Explore advanced neural network models that better handle complex interactions, such as hybrid models combining LSTM with other architectures like Convolutional Neural Network (CNN), as done by Amarpuri et al. [12]

5.2 Alternative Network Designs

Alternative network designs were explored in this project. While they were not completed, they are potential avenues for future enhancements and research.

5.2.1 Multi-Layer Perceptron

The MLP model was initially evaluated as a baseline for network design. The `feedforwardnet` function can build a MLP neural network in MATLAB. The hidden layer configuration and training optimisier are specified through input arguments.

- Hidden layers are defined using a MATLAB vector. For example, a network with two hidden layers containing 128 and 64 neurons, respectively, is specified as `hiddenLayers = [128 64];`.
- The training algorithm is designated by the `trainFcn` parameter. Various options are available, but `trainlm`, which uses Levenberg-Marquardt Backpropagation is default option [23].

Listing 16: Initialising and training a MLP in MATLAB

```
net = feedforwardnet([512 256], 'trainlm');
[net,tr] = train(net, inputs, targets);
```

5.2.1.1 Benefits compared to Long Short-Term Memory

The MLP network design frames the problem as a function approximation. The aim is to determine the underlying function that, given a set of N input features, produces the marginal emissions. The MLP model can be regarded as an N-dimensional function corresponding to the N input features. Because of this, it can be seen as an alternative to the LSTM model, which considers a time series prediction. Some benefits are,

- **Simplicity** – The MLP model is simpler in terms of architecture compared to LSTM networks, which make them easier to understand, implement and maintain.
- **Practicality** – If the problem involves only a few inputs, such as (1) Δ Demand, (2) Δ Coal, (3) Δ Generation, (4) Δ Wind — you could use toggles or other controls to manipulate the MEFs directly. This might make the model more transparent and user-friendly for practical applications where only key factors like changes in demand or specific fuel types

5.2.1.2 Results

15 different iterations of MLP design were explored with varying outcomes. Each network was trained on a subset of the entire dataset. Experiments (11-15) were built using data from 2022 and after (total of 33,274 samples), and all other experiments (1-4) were only considered the 2023 dataset (total of 15,843 samples). This was done to accelerate training time, and possibly produce more accurate results since the network has a smaller range of data to capture the underlying patterns.

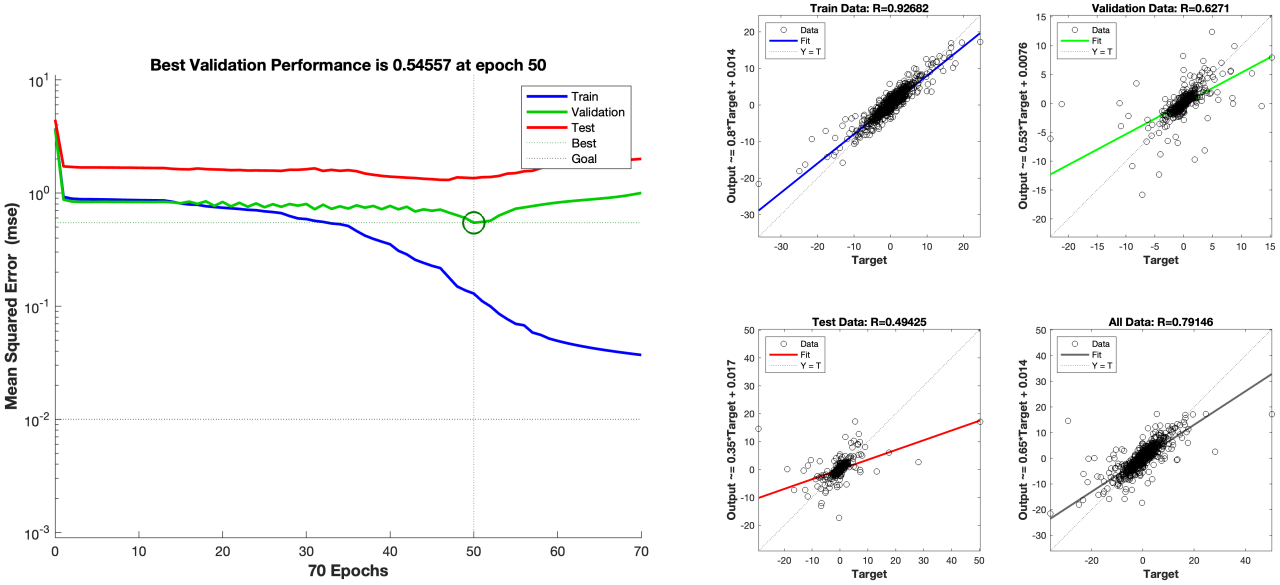
	Features	N	Epochs	MSE
1	CO_2 Emissions, Wind, Demand, Coal, Gas, Biomass, Oil, Distillate, Δ Gen	9	12	0.970
2	CO_2 Emissions, Wind, Demand, Coal, Gas, Biomass, Oil, Distillate, Δ Gen, ΔCO_2 Emissions, ΔOil, ΔDemand, InterEwic, ΔInterEwic	14	4	0.978
3	ΔCO_2 Emissions, Coal, Wind, ΔDistillate, ΔDemand, Solar, ΔGas, OtherRenewable	8	3	1.045
4	Δ Gen, ΔWind, ΔCO_2 Emissions, ΔCoal, ΔGas, ΔBiomass, ΔOil	7	8	0.975
5	Δ Gen, ΔWind, ΔCO_2 Emissions, ΔCoal, ΔGas, ΔBiomass, ΔOil, ΔOtherRenewable, ΔBiomass, InterEwic	10	214	1.036
6	Δ Gen, ΔWind, ΔCO_2 Emissions, ΔCoal, ΔGas, ΔBiomass, ΔOil, ΔOtherRenewable, ΔBiomass, InterEwic	10	51	0.401
7	CO_2 Emissions, Demand, Coal, Gas, Δ Gen, ΔWind, ΔCO_2 Emissions, ΔCoal, ΔGas, ΔBiomass, ΔOil, ΔOtherRenewable, ΔBiomass, InterEwic	14	102	0.969
8	Δ Gen, ΔWind, ΔCO_2 Emissions, ΔCoal, ΔGas, ΔBiomass, ΔOil, ΔOtherRenewable, ΔBiomass, InterEwic	10	5	1.071
9	Δ Gen, ΔWind, ΔCO_2 Emissions, ΔCoal, ΔGas, ΔBiomass, ΔOil, ΔOtherRenewable, ΔBiomass, InterEwic	10	21	0.809
10	ΔDemand, ΔCoal, Δ Gen, ΔCO_2 Emissions, GenExp	5	69	0.500
11	ΔDemand, ΔCoal, Δ Gen, ΔCO_2 Emissions, GenExp, Demand	6	13	0.934
12	ΔOil, ΔSolar, ΔBattery, ΔDemand, ΔCoal, ΔHydro, ΔBiomass, Δ Gen, ΔCO_2 Emissions, GenExp, ΔOther Non-Renewable	11	137	0.096
13	ΔOil, ΔSolar, ΔBattery, ΔDemand, ΔCoal, ΔHydro, ΔBiomass, Δ Gen, ΔCO_2 Emissions, GenExp, ΔOther Non-Renewable	11	71	0.202
14	ΔOil, ΔSolar, ΔBattery, ΔDemand, ΔCoal, ΔHydro, ΔBiomass, Δ Gen, ΔCO_2 Emissions, GenExp, ΔOther Non-Renewable	11	50	0.129
15	ΔOil, ΔSolar, ΔBattery, ΔDemand, ΔCoal, ΔHydro, ΔBiomass, Δ Gen, ΔCO_2 Emissions, GenExp, ΔOther Non-Renewable	11	174	0.839

Table 4: Overview of various MLP network configurations tested. The table details the experiment number, N, the dimensionality, training epochs and best performance for each MLP experiment

Similar to the LSTM model, the models were trained using a subset of the data. 70% was used for training, and 15% for validation and 15% for testing. However, this randomisation process occurred during each experiment – making it harder to compare the performance between the network designs as they were not trained on the same data. This was rectified when building the LSTM network.

As well, both the input and the target variables were scaled using the same `StandardScaler` (see listing 22). The network's performance was evaluted using Mean Squared Error (MSE). Early stopping was implemented if the validation MSE increased over 20 epochs. Unlike RMSE, it does not keep the same unit as the predictions. Because of this, the network's error cannot be directly interpreted since the output scale has been changed.

All networks were trained using Levenberg-Marquardt Backpropagation, with two hidden layers of 50 and 20, respectively. Experiment 6 and 15, which were trained using gradient descent with adaptive learning rate (`traingdx` in MATLAB). The best performance was experiment 13 with a MSE of 0.129.



(a) MSE curves for training, validation and testing **(b)** Regression graphs between the inputs and targets

Figure 13: The performance metrics from experiment 13 of the alternative MLP network design

fig. 13a displays the performance graphs for testing, training, and validation. At epoch 50, the training error decreases sharply while the validation error increases, showing that the network has ceased generalising and is now overfitting to the dataset. The significantly higher best validation performance compared to the test MSE of 0.129 corroborates this observation.

fig. 13b plots a regression curve against the training, testing, and validation datasets, revealing their correlation coefficients. The training data exhibits a strong, positive correlation between predictors and targets, while the test data shows only a weak positive correlation, further showing overfitting, which diminishes the model's practical utility.

5.2.1.3 Interpretation Though the relatively low test MSE of 0.129 might seem promising, suggesting that the MLP network is generating accurate predictions, this is deceptive for several reasons:

- The error metric is standardised and not reported in the same units as the predictions. Although 0.129 may appear numerically trivial, it could represent a significant error on the scale the data was trained on, a detail obscured by this standardisation.
- The markedly lower error compared to the validation error suggests overfitting. While superficially positive, it likely shows that the network cannot generalise effectively, merely learning the outputs from the training data.

5.2.1.4 Challenges The primary challenge with a MLP in this context would concern dimensionality, especially if the complexity of the input data increases or if the relationships between inputs and outputs are highly non-linear, which are more adeptly handled by LSTM networks.

From experimentation, a smaller number of inputs often yielded better results. However, with fewer inputs, there are fewer data patterns available to decipher the underlying functions behind them and the MEFs. Unlike LSTM networks, which benefit from their ability to learn from historical data to identify patterns, MLP networks do not have this capability. Therefore, careful feature selection becomes crucial when designing an MLP network.

The selected input features must be uncorrelated and encapsulate as much diverse information as possible within the smallest workable dimensionality. This is challenging when predicting MEFs because there are many potential input features that could influence determining which generator is at the margin. Considering only the empirical data set of grid parameters and fuel mixes, if we were to construct a 4-layer MLP from the total data set, that results in $\binom{40}{4} = 91,390$ combinations.

This does not consider potential extraneous factors, such as weather patterns or market data, which could also significantly impact marginal emissions.

5.2.2 Linear Composite Model

The final alternative network design explored is a linear composite model, akin to the one used in Mayes et al. [5] for projecting the MEFs in California. An ANN predicts the carbon intensity, and a linear function then uses this prediction along with the change in generation as secondary inputs to calculate the marginal emissions, according to eq. (3.4).

This approach is considered to evaluate whether performance improvements could be achieved, considering that predicting carbon intensity is relatively straightforward compared to marginal emissions. The carbon intensity model can be treated as a typical regression problem, where the network is tasked with learning the general pattern and ignoring the noise or outliers. In contrast, the MEF model requires the network to accurately predict spikes and outliers, as these carry crucial information.

5.2.2.1 Feature Selection Carbon intensity represents the average emissions, calculated as the total carbon emissions divided by generation. As discussed in section 2.4, this results in a relatively small amount of grey information in the dataset for predicting carbon intensity compared to that for marginal emissions. The Pearson correlation can determine the optimal feature set, selecting features that exhibit a relatively strong correlation with the target variable, specifically where $|r| > 0.19$.

Predictor	**Description**	**Unit**	**Source**	r
Co2Emissions	Total Carbon Emissions	tCO_2/hr	EirGrid	0.83
GenExp	Total Energy Production	MWH	EirGrid	-0.21
InterEwic	Flow of energy between ROI and Wales	MWH	EirGrid	0.24
WindActual	Actual Wind Generation	MWH	EirGrid	-0.80
Coal	Non-Renewable Energy Source	MWH	CSO	0.64
Distillate	Non-Renewable Energy Source	MWH	CSO	0.28
Gas	Non-Renewable Energy Source	MWH	CSO	0.63
Oil	Non-Renewable Energy Source	MWH	CSO	0.47
Other Renewable	Renewable Energy Source	MWH	CSO	0.25

Table 5: The identified input feature set based on the statistical analysis

fig. 19 visualises the table below and fig. 18 shows regression plots of the input features against CO_2 intensity to further understand the relationship between the features. The table above has idenitfied 10 input features that can be used as a baseline for building the neural model. The weakly correlated inputs were removed to reduce the dimensionality as needed.

5.2.2.2 Neural Model The neural model can be designed as a Multi Layer Perceptron similar to the baseline model designed for predicting the MEFs as outlined in section 5.2.1. Rather than a LSTM for its simplicity. Here is a proposed outline of the neural model design,

- The model has two outputs
    1. the carbon intensity at the current time step CI_t
    2. the carbon intensity at the next time step $CI_{t+\Delta t}$.
- The model will have two hidden layers with 50 and 20 neurons, respectively
- The model will be trained using the Levenberg-Marquardt backpropogation

5.2.2.3 Linear Model The linear model takes the two outputs of the neural model and applies eq. (3.4) to determine the MEFs. The additional inputs to the linear model are,

1. The generation at the current time step PG_t
2. The generation at the next time step $PG_{t+\Delta t}$

Listing 17: Example demonstration of linear composite model transformation in MATLAB

```
index = find(strcmp(inputfeatures, 'GenExp'));
GenExp = X(index,:);
GenExp_deltaT = circshift(GenExp, -deltaT);
epsilon = GenExp .* y(1,:);
epsilon_deltaT = GenExp_deltaT .* y(2,:);
deltaP = GenExp_deltaT - GenExp;
mef = (epsilon_deltaT - epsilon) ./ deltaP;
```

The transformation is as follows,

- The output of the neural model is denormalised back to the original scale
- $CI_t \times PG_{i,t}$ is calculated for the both time steps based on the two outputs of the neural model
- The difference in these values is divided by the difference to the linear model inputs

6 Conclusion

This project set a clear objective: to develop a model for predicting marginal CO_2 emissions in Ireland's electricity sector, addressing the challenges posed by VRE integration. Through processing historical grid data and applying established machine learning techniques, this goal has been achieved.

Our analysis revealed the highly erratic nature of marginal emissions, highlighting the stark contrast to static average emissions figures. In line with our aim, a LSTM model was designed and fine-tuned, showing promise in forecasting MEFs. Despite a relatively low RMSE, the model's difficulty in precisely predicting emission spikes suggested the occurrence of underfitting, while its close adherence to certain data segments shows potential overfitting.

The findings from the GRA provided valuable insights into feature relevance, guiding the feature selection for the LSTM network and reinforcing the importance of input variables such as non-VRE and demand changes. The aim of acquiring insight into the factors that influence marginal emissions has been accomplished by employing techniques gained from the literature review and validated through experiments.

Compared to existing literature, the developed model stands out given in its focus on Ireland. The results show an advanced capability in handling the complexity of time-series data that influence marginal emissions. The work undertaken here builds a new method from Rabiee et al. [4] and eq. (3.4). table 6 offers an in-depth comparison between our work and what was studied during the literature review.

6.1 Basis for Future Work

The list of input features outlined in table 3 isn't entirely conclusive, as there are biases introduced by only considering a finite number of data sources. A thorough analysis would consider multiple experimentations using random permutations of the dataset, or segmenting the dataset into temporal sections for individual analysis and comparison. However, this extends well beyond the scope of this project. The method applied in section 4.2 is sufficient, and it leaves scope for future analysis.

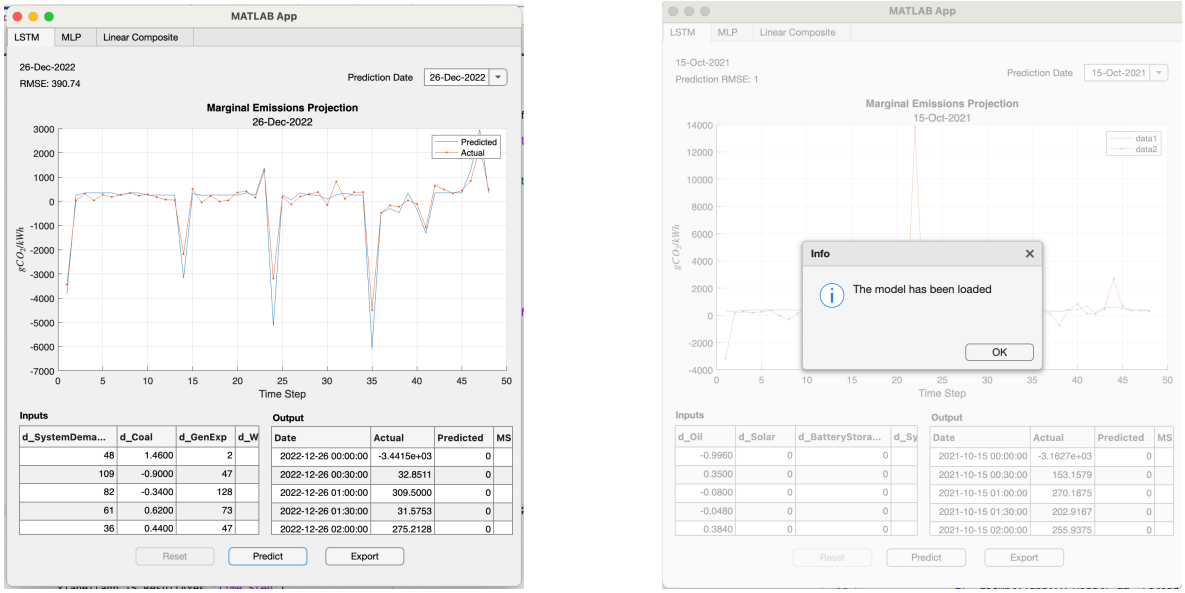
Limitations in this project arose from the high-dimensionality of datasets and the inherent difficulties in capturing the erratic behaviour of marginal emissions. Projecting the MEF based on the approach outlined in Rabiee et al. [4] is atypical to standard regression problems, where noise and outliers are discarded as containing irrelevant information. Here, these extrema offer invaluable insights into the changes in CO_2 emission levels, which have real-world applications in demand curtailment and policy analysis. Because of this, the training of the network is complicated, since it is attempting to find the right balance between generalisation and accuracy. Hence, there is a high risk of overfitting if the input features do not fully encapsulate the underlying patterns in the MEF profiles. This is the biggest challenge in this project that

would need to be addressed in future research.

Addressing this challenge would likely involve incorporating additional extraneous factors that impact MEF, alongside those identified in table 8. With that said, this is a very difficult problem that theoretically may have no end, since there are a multitude of factors that have indirect influences on marginal emissions. Drawing from Mayes et al. [5], a more targeted approach might concentrate on changes in VRE and non-VRE outputs, as well as on temporal and demand data. Future work on this project can focus on ensuring the data collected for these input features are high quality and conclusive, and experiment with feature engineering to form combinations of these inputs that offer different information.

6.1.1 Visualisation

A Graphical User Interface (GUI) for interacting with the different predictive models was explored using MATLAB's App Designer to demonstrate the potential practical applications of the models. A prototype was built during the project, but a final version has not been produced.



(a) after a prediction has been made **(b)** after the model has been loaded

Figure 14: UI of the visualisation while it is active

The Graphical User Interface (GUI) shown in fig. 14 has three tabs, corresponding to each network designs, and three primary actions,

1. **Predict** – Starts the prediction process for the selected prediction date:
    - The network model is loaded into the workspace, if not already. The system profile for the specified day is retrieved from the complete system profile dataset
    - Relevant input features are selected according to the network design, and necessary pre-processing steps such as sequence formation and standardisation are applied.

- The processed inputs are fed into the network to generate the outputs. Post-processing steps, including denormalisation, are applied to the outputs, which are then displayed on the graph and the results tables are updated accordingly.

2. **Export** – Saves the resulting tables and graphs as individual files.
3. **Reset** – Resets the state of the visualisation to its initial configuration.

The practical application of this is to make interacting with the model easier for the end-user. It also gives an example of how a policymaker could interact with the model. Finishing the visualisation, and including the alternative network designs, is a basis for future work and research.

6.1.2 Policy Analysis

The models developed in this project holds substantial potential for application in policy analysis. Building off the visualisation, a tool can be developed for policymakers to evaluate the implications of climate policies. These models could discern the effectiveness of a new policy by simulating its impact on marginal emissions, offering a granular assessment of its environmental footprint. Take, for example, a future iteration of the model as a MLP that accepts four inputs:

1. Wind
2. ΔDemand
3. ΔCoal
4. ΔGeneration

This could easily assess the impact of low-carbon technology integration. Existing unit commitment models can provide inputs on forecasted changes in conventional generation, given they adhere to strict protocols. By passing these inputs into the model, a policymaker could assess the resultant marginal emissions profile. The visualisation could be adjusted to offer toggles that allow interactive tweaking of these inputs and live predictions. If the final model was a LSTM, the same design could be adjusted to include an additional input for obtaining a time series prediction of the MEF profile for a specific sample date. This change, when applied, would aid Ireland in advancing targets 7 and 13 of SGD.

6.2 Ethics

The deployment of AI and predictive tools in shaping policy decisions is laden with ethical implications. It is imperative to critically evaluate and understand the limits and potential biases inherent in these technologies. The predictions made by the designed model must be transparent, explainable and thoroughly peer-vetted to avoid misguiding policy decisions.

In tandem, leveraging open source data and technology in projects like this raise questions about the balance between serving the public interest and personal gain. Since many of these resources are meant

for communal benefit—a distinction must be made between using them for public good versus private gain. Even more so given the nature of the project dealing with outcomes that could influence societal and public policy matters.

The development of an ANN requires extensive data collection and mining. Ethical practises around such processes are paramount—upholding privacy, securing consent, and respecting data ownership. Being transparent about the origins, methods of collection, and intended use of the data are important. As well as ensuring that the application of the data aligns with public interest.

Lastly, it must be noted that inaccuracies in predictions can misdirect policies adversely affecting environmental objectives. Striving for precision and openly discussing the models limitations and uncertainties is ethically essential. Misrepresentations, overshoot or neglect can undermine the environmental protection and sustainable development efforts put forth in this project.

6.3 Sustainability

The development of a method to predict Ireland's marginal emissions factors has the potential to significantly influence the sustainability of the national electricity sector. By providing an analytical tool to inform policy decisions, the project can aid in the integration of renewable energy sources and technologies leading to a sustainable, cleaner energy mix. The advantage this project offers over existing methods is the ability to assess how demand response programs affect marginal emissions at a given point in time. This gives the granular ability to understand the direct environmental consequence of a given demand-based climate action policy (.e.g. EV integration).

The nature of the project aligns with global trends in sustainable energy development with similar projects being undertaken in other parts of the world, Mayes et al. [5] in California and Tran et al. [15].

6.3.1 Sustainable Development Goals

The project's focus on marginal emissions projection is directly relevant to Sustainable Development Goals 7 and 13. The project contributes to SDG 7 by identifying paths towards more efficient and renewable energy usage by understanding the nuances of marginal carbon emissions. Furthermore, it aligns with SDG 13 by offering data driven insights for informed climate action, including strategies that can reduce carbon emissions and mitigate climate impacts.

Bibliography

[1] United Nations, "THE 17 GOALS | Sustainable Development." [Online]. Available: https://sdgs.un.org/goals

[2] Department of Environment, Climate and Communication (DECC),, "Climate Action Plan 2023," Tech. Rep. [Online]. Available: https://www.gov.ie/pdf/?file=https://assets.gov.ie/270956/94a5673c-163c-476a-921f-7399cdf3c8f5.pdf#page=null

[3] A. Pepiciello, F. De Caro, and A. Vaccaro, "Real-time Pricing Demand Response Scheme based on Marginal Emission Factors," in *2022 IEEE International Conference on Environment and Electrical Engineering and 2022 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*, Jun. 2022, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9854687

[4] A. Rabiee, S. A. Alavi, A. Keane, and J. McCann, "An Approach to Calculate Marginal CO $_2$ Emissions Factor Based on Historical Emissions," in *2023 58th International Universities Power Engineering Conference (UPEC)*. Dublin, Ireland: IEEE, Aug. 2023, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/10294836/

[5] S. Mayes, N. Klein, and K. T. Sanders, "Using neural networks to forecast marginal emissions factors: A CAISO case study," *Journal of Cleaner Production*, vol. 434, p. 139895, Jan. 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652623040532

[6] S. Marsland, *Machine Learning: An Algorithmic Perspective*, 2nd ed. Chapman and Hall/CRC. [Online]. Available: https://www.taylorfrancis.com/books/9781466583337

[7] R. Sallehuddin, S. M. H. Shamsuddin, and S. Z. M. Hashim, "Application of Grey Relational Analysis for Multivariate Time Series," in *2008 Eighth International Conference on Intelligent Systems Design and Applications*, vol. 2, Nov. 2008, pp. 432–437, iSSN: 2164-7151. [Online]. Available: https://ieeexplore.ieee.org/document/4696371

[8] "Renewables." [Online]. Available: https://www.seai.ie/data-and-insights/seai-statistics/key-statistics/renewables/

[9] S. P. Holland, M. J. Kotchen, E. T. Mansur, and A. J. Yates, "Why marginal CO2 emissions are not decreasing for US electricity: Estimates and implications for climate policy," *Proceedings of the National Academy of Sciences*, vol. 119, no. 8, p. e2116632119, Feb. 2022, publisher: Proceedings of the National Academy of Sciences. [Online]. Available: https://www.pnas.org/doi/full/10.1073/pnas.2116632119

[10] M. Maarif, A. Rahman Saleh, M. Habibi, N. L. Fitriyani, and M. Syafrudin, "Energy Usage Forecasting Model Based on Long Short-Term Memory (LSTM) and eXplainable Artificial Intelligence (XAI)," *Information*, vol. 14, p. 265, Apr. 2023.

[11] S. E. Bouziane, J. Dugdale, and M. T. Khadir, "Modeling renewable energy production and CO2 emissions in the region of Adrar in Algeria using LSTM neural networks," 2022. [Online]. Available: http://hdl.handle.net/10125/79641

[12] L. Amarpuri, N. Yadav, G. Kumar, and S. Agrawal, "Prediction of CO2 emissions using deep learning hybrid approach: A Case Study in Indian Context," in *2019 Twelfth International Conference on Contemporary Computing (IC3)*, Aug. 2019, pp. 1–6, iSSN: 2572-6129. [Online]. Available: https://ieeexplore.ieee.org/document/8844902

[13] Y. Huang, L. Shen, and H. Liu, "Grey relational analysis, principal component analysis and forecasting of carbon emissions based on long short-term memory in China," *Journal of Cleaner Production*, vol. 209, pp. 415–423, Feb. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652618331445

[14] M. Peixeiro, *Time Series Forecasting in Python*.

[15] J. Tran, J. Gilles, R. Mann, and V. Murthy, "Automated Demand Response Refrigerator Project," 2015.

[16] A. G. N. Elenes, E. Williams, E. Hittinger, and N. S. Goteti, "How Well Do Emission Factors Approximate Emission Changes from Electricity System Models?" *Environmental Science & Technology*, vol. 56, no. 20, pp. 14 701–14 712, Oct. 2022, publisher: American Chemical Society. [Online]. Available: https://doi.org/10.1021/acs.est.2c02344

[17] D. Singhal and K. S. Swarup, "Electricity price forecasting using artificial neural networks," *International Journal of Electrical Power & Energy Systems*, vol. 33, no. 3, pp. 550–555, Mar. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0142061510002231

[18] M. Singh and R. K. Dubey, "Deep Learning Model Based CO2 Emissions Prediction Using Vehicle Telematics Sensors Data," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 768–777, Jan. 2023, conference Name: IEEE Transactions on Intelligent Vehicles. [Online]. Available: https://ieeexplore.ieee.org/document/9508812

[19] C. Wang, Y. Wang, C. J. Miller, and J. Lin, "Estimating hourly marginal emission in real time for PJM market area using a machine learning approach," in *2016 IEEE Power and Energy Society General Meeting (PESGM)*, Jul. 2016, pp. 1–5, iSSN: 1944-9933. [Online]. Available: https://ieeexplore.ieee.org/document/7741759

[20] A. Mohammadazadeh, M. H. Sabzalian, O. Castillo, R. Sakthivel, F. F. M. El-Sousy, and S. Mobayen, *Neural Networks and Learning Algorithms in MATLAB*, ser. Synthesis Lectures on Intelligent Technologies. Cham: Springer International Publishing, 2022. [Online]. Available: https://link.springer.com/10.1007/978-3-031-14571-1

[21] X. Yu, M. O. Efe, and O. Kaynak, "A general backpropagation algorithm for feedforward neural networks learning," *IEEE transactions on neural networks*, vol. 13, no. 1, pp. 251–254, 2002.

[22] J. Christensen and C. Bastien, "Chapter | three - introduction to general optimization principles and methods," in *Nonlinear Optimization of Vehicle Safety Structures*, J. Christensen and C. Bastien, Eds. Butterworth-Heinemann, pp. 107–168. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124172975000031

[23] Mark Hudson Beale, Martin T. Hagan, and Howard B. Demuth, "Deep learning toolbox user's guide." [Online]. Available: https://ge0mlib.com/papers/Books/04_Deep_Learning_Toolbox_Users_Guide.pdf

[24] Yi Lin and Sifeng Liu, "A historical introduction to grey systems theory," in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, vol. 3. IEEE, pp. 2403–2408. [Online]. Available: http://ieeexplore.ieee.org/document/1400689/

[25] S. Liu, Y. Yang, and J. Y.-L. Forrest, *Grey Systems Analysis: Methods, Models and Applications*, ser. Series on Grey System. Springer Nature Singapore. [Online]. Available: https://link.springer.com/10.1007/978-981-19-6160-1

[26] D. Nettleton, "Chapter 6 - selection of variables and factor derivation," in *Commercial Data Mining*, D. Nettleton, Ed. Morgan Kaufmann, pp. 79–104. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124166028000066

A.1 Comparison to Established Work

Source	Description
Own Work	MEF projection in Ireland from EirGrid data based on 10-input LSTM network **Inputs:** (1) GenExp (2) ΔBiomass (3) Δ Emissions (7) ΔInterEwic (8) ΔHydro (9) ΔDemand (10) ΔWind **Design:** 128 hidden 128 LSTM, 50 hidden, **Hyperparameters:** 50% dropout, tanh activation, 2,700 epochs, $\eta = 0.02$, Adam optimiser, 70/30/20 split
Mayes et al. [5]	MEF projection California from CAISO data with a MLP-linear composite model **Neural inputs:** (1) Demand (2) VRE (3) Hour of Day (4) Day of year (5) Time Since 2018 **Linear inputs:** (1) ΔDemand, (2) ΔVRE **Design:** 512 hidden, 256 hidden, normalisation layer **Hyperparameters:** 50% dropout, ReLu activations, 40,000 epochs, $\eta = 0.01$, Adam optimiser, 60/20/20 split
Maarif et al. [10]	Energy usage forceasting using a LSTM network **Inputs:** 9 features, time information, load demand and historical price information **Design:** Single-layer, double-layer and bi-directional LSTM networks with varying window sizes **Hyperparameters:** Optimal number of LSTM units found using GridSearch to be 64, 10% dropout
Singhal and Swarup [17]	Electricity price forecasting using MLP network **Inputs:** 5 features, Current hour energy usage and previous 4, 8, 12 and 16 hours **Design:** 10-hidden and 5-hidden
Bouziane et al. [11]	Renewable energy and CO_2 emission foreacsting in Algeria using LSTM network **Inputs:** Used exogenous inputs such as temperature, wind speed, irradiance **Design:** Separate models were created for PV and Wind energy. Wind model had 50 and 30 hidden neurons **Hyperparameters:** Adam optimiser, $\eta = 0.01$, varied droput from 10-30%
Amarpuri et al. [12]	Predicting CO_2 emisisons in India using a hybrid LSTM network **Design:** Hybrid model of CNN and LSTM — input layer, convolutional layer, pooling layer, LSTM layer **Hyperparameters:** Adam, 1D-convolutional layer with 138 filters, ReLu activations, 100 LSTM units
Singh and Dubey [18]	Predicting CO_2 emissions based on vehicle telematic sensor data with a LSTM network **Inputs:** Engine load, speed, temperature, RPM, mileage, throttle, accelleration, fuel flow **Hyperparameters:** Moving 64-time step window with 50% overlap

Table 6: Comparing the LSTM network to established work in literaure

B.2 Dataset Tables

EffectiveTime	SysFrequency	CO_2 Emissions	Co2Intensity	SystemDemand	GenExp	InterEwic	WindActual
2024-04-16 23:30:00	50.03	814.0	233.0	3500.0	3261.0	3.0	1366.0
2024-04-16 23:15:00	49.99	844.0	236.0	3577.0	3371.0	3.0	1379.0
2024-04-16 23:00:00	50.01	819.0	225.0	3645.0	3303.0	3.0	1420.0
2024-04-16 22:45:00	50.0	815.0	217.0	3755.0	3313.0	3.0	1447.0
2024-04-16 22:30:00	49.98	823.0	213.0	3873.0	3431.0	3.0	1527.0
2024-04-16 22:15:00	49.98	838.0	209.0	4000.0	3535.0	3.0	1591.0
2024-04-16 22:00:00	50.04	879.0	217.0	4058.0	3684.0	3.0	1558.0

Table 7: Extract of the Eirgrid System Dataset

Feature	Description	Unit	Source	GRG	r
ΔOil	Change in Energy from Oil	MWH	CSO	0.98	0.01
ΔSolar	Change in Solar	MWH	CSO	0.95	-0.0
ΔBattery Storage	Change in Energy from Battery Storage	MWH	CSO	0.94	0.0
ΔSystemDemand	Change in Total Energy Demand	MWH	EirGrid	0.93	0.01
ΔCoal	Change in Energy from Coal	MWH	CSO	0.91	0.01
ΔBiomass/Peat	Change in Energy from Biomass	MWH	CSO	0.9	-0.0
ΔPumped Storage Hydro	Change in Pumped Hydro	MWH	CSO	0.87	-0.0
ΔOther Non-Renewable	Change in Energy from Non-Renewables	MWH	CSO	0.87	0.01
ΔWindActual	Change in Wind	MWH	EirGrid	0.85	-0.01
ΔGenExp	Change in Generation (.i.e. deltaP)	MWH	EirGrid	0.84	0.0
ΔSysFrequency	Change in System Frequency	Hz	EirGrid	0.83	0.01
Δ CO_2Emissions	Change in Total Carbon Emissions	tCO_2/hr	EirGrid	0.83	0.02
Δ CO_2Intensity	Change in Average Emissions	gCO_2/kWh	EirGrid	0.82	0.01
GenExp	Total Energy Production	MWH	EirGrid	0.82	0.0
ΔInterEwic	Change in Interconnection between ROI/Wales	MWH	EirGrid	0.8	0.0
Co2Intensity	Average Emissions Factors	gCO_2/kWh	EirGrid	0.79	0.02
ΔWaste	Change in Wind	MWH	CSO	0.78	-0.0
SystemDemand	Total Energy Demand	MWH	EirGrid	0.78	0.01
ΔRenewable Hydro	Change in Hydro	MWH	CSO	0.78	-0.0
Gas	Non-Renewable Energy Source	MWH	CSO	0.77	0.02
Other Non-Renewable	Non-Renewable Energy Source	MWH	CSO	0.76	-0.0
SysFrequency	System grid frequency	Hz	EirGrid	0.75	0.01
CO_2 Emissions	Total Carbon Emissions	tCO_2/hr	EirGrid	0.75	0.03
InterEwic	Flow of energy between ROI and Wales	MWH	EirGrid	0.75	0.0
ΔDistillate	Change in Energy from Distillate	gCO_2/MWh	CSO	0.74	-0.0
WindActual	Actual Wind Generation	MWH	EirGrid	0.72	-0.02
Coal	Non-Renewable Energy Source	MWH	CSO	0.71	0.01
Renewable Hydro	Renewable Energy Source	MWH	CSO	0.71	-0.01
Biomass/Peat	Non-Renewable Energy Source	MWH	CSO	0.71	0.01
ΔGas	Change in Energy from Gas	gCO_2/MWh	CSO	0.68	0.0
ΔOther Renewable	Change in Energy from Renewables	MWH	CSO	0.66	0.0
Other Renewable	Renewable Energy Source	MWH	CSO	0.66	0.01
EffectiveTime_sin	Sine encoding of effective time	N/A	EirGrid	0.64	-0.0
EffectiveTime_cos	Cosine encoding of effective time	N/A	EirGrid	0.64	-0.0
Oil	Non-Renewable Energy Source	MWH	CSO	0.61	0.01
Pumped Storage Hydro	Renewable Energy Source	MWH	CSO	0.6	-0.01
Waste	Mixed Energy Source	MWH	CSO	0.58	0.01
Solar	Renewable Energy Source	MWH	CSO	0.57	0.01
Distillate	Non-Renewable Energy Source	MWH	CSO	0.56	0.01
Battery Storage	Renewable Energy Source	MWH	CSO	0.55	-0.0

Table 8: Complete breakdown of the 40 input features, their units and data sources. Alongside the r value which represents the pearson linear correlation coefficent, and the associated grey relational grades

C.3 Figures

C.3.1 Historical Marginal Emissions

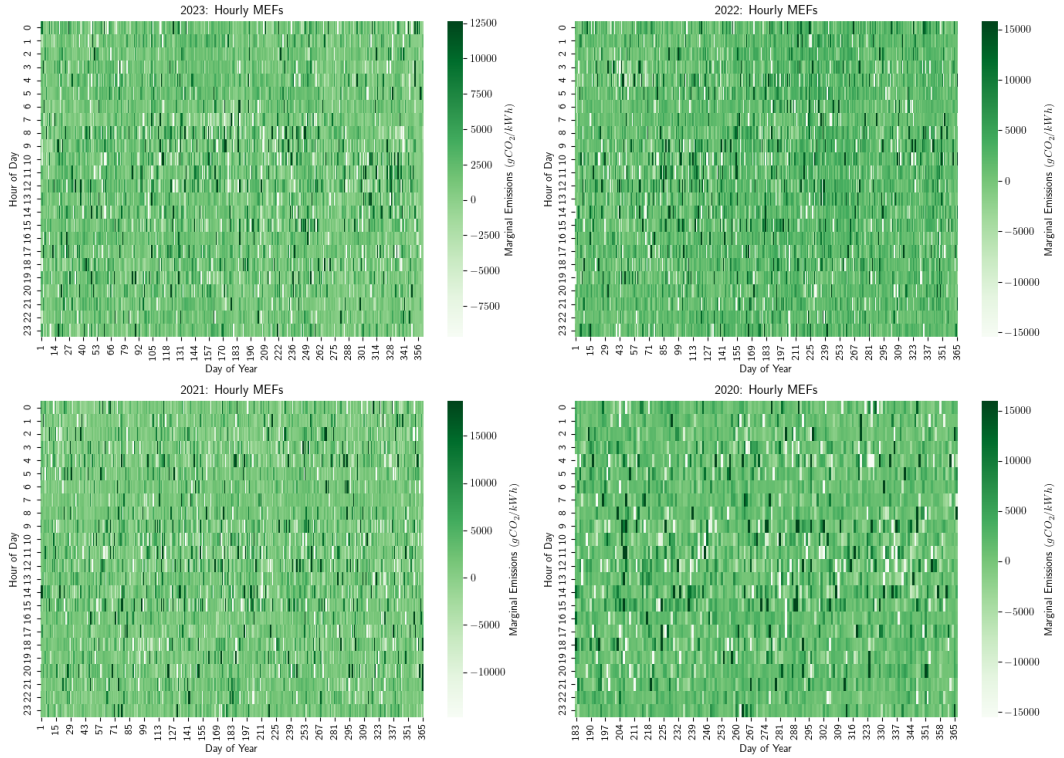
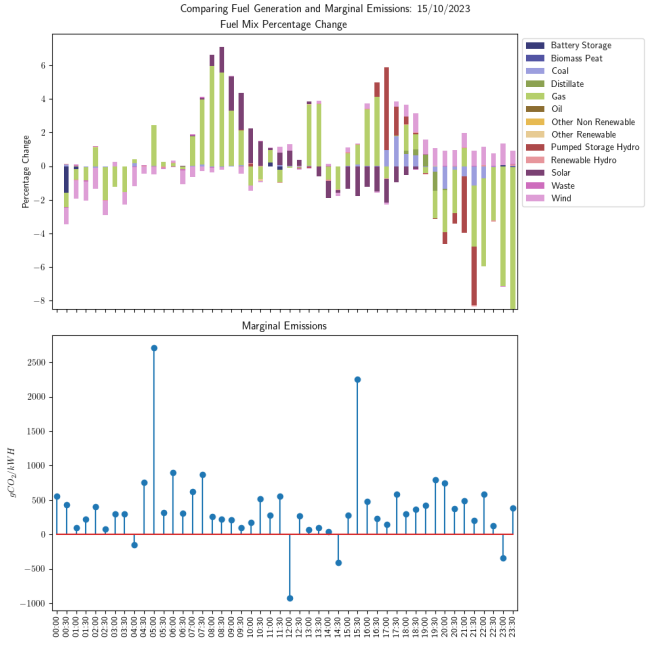
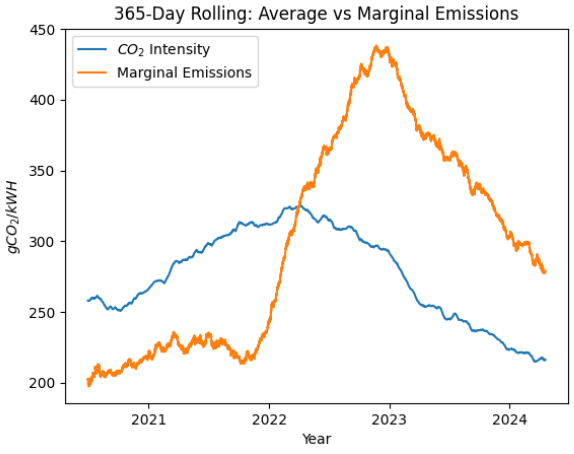


Figure 15: Heat map of the Half-hourly MEFs calculated for 2020, 2021, 2022 & 2023 created using the publicly available data sets from EirGrid and Python



(a) Dual plot comparing the half hourly changes in metered fuel generation (from CSO reports) versus half-hourly marginal emissions calculated using the prescribed methodology on October 15^{th} 2023



(b) 365-day rolling average of MEF vs AEFs

Figure 16: Marginal emissions compared against fuel generation, and 365-rolling average of marginal emissions

C.3.1.1 LSTM Network & Visualisation

Figure 17: Representative plot of the forecasted vs actual MEF on 25/12/2021 from the 10-input LSTM network. The bottom graph shows the input sequences used in the prediction, with several notable peaks and throughs in the output graph

C.3.1.2 Linear Combination Model

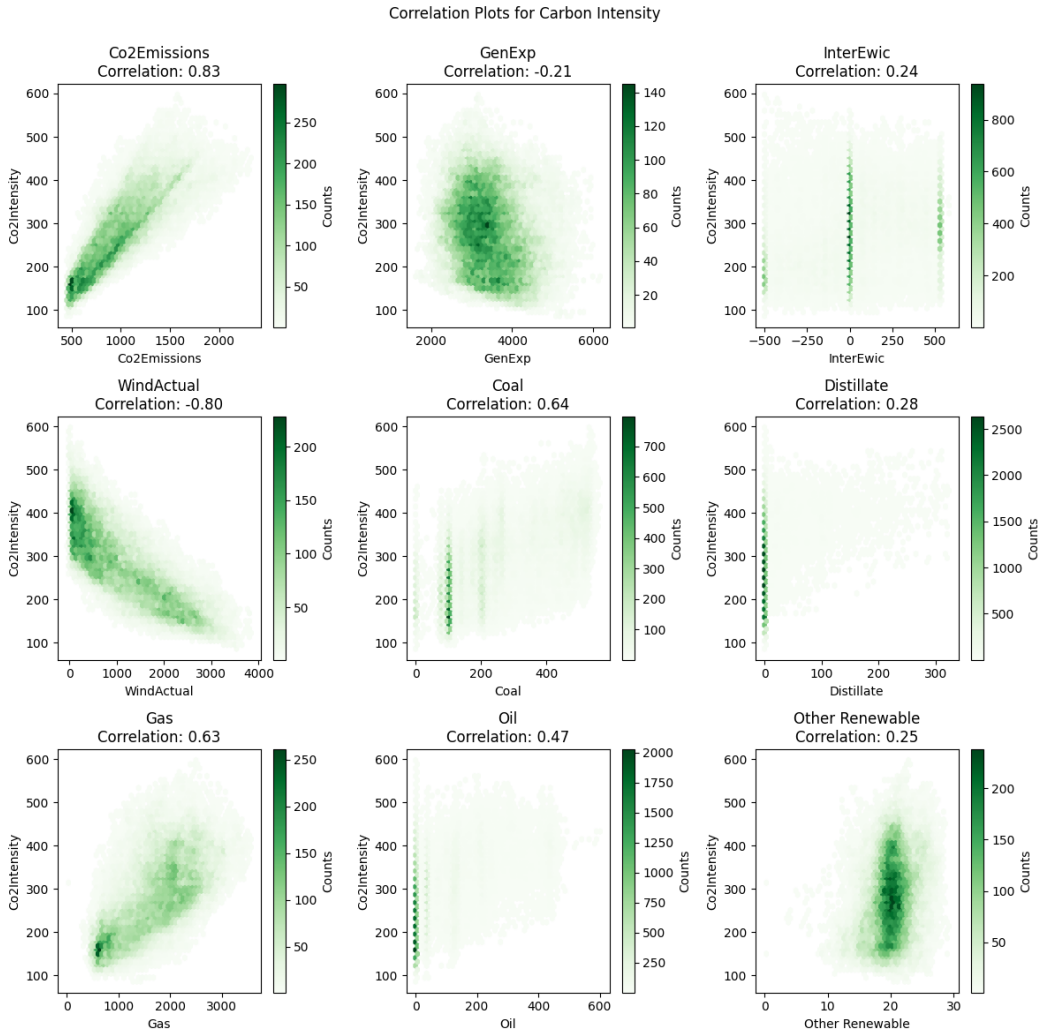


Figure 18: Scatter plots of the various non-neglible predictors against CO_2 intensity

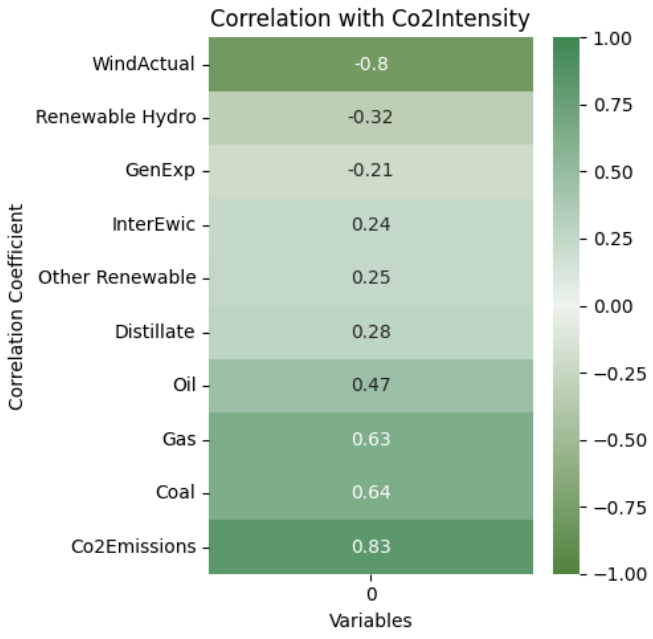


Figure 19: CO_2 intensity pearson correlation co-efficents that are non-neglible (.i.e. $|r| \geq 0.19$)

D.4 Code Listings

Listing 18: Python CLI `eirgrid_data_collection.py` used to collect the historical EirGrid dataset

```python
@cli_collect.command()
@click.option('-s', '--start', required=True, help='The start date in DD/MM/YYYY')
@click.option('-e', '--end', help='The end date in DD/MM/YYYY')
@click.option('-r', '--region', default='ROI')
@click.argument('areas', nargs=-1, required=True)
def collect(start: str, end: str, areas: tuple[str], region: str, usedates: bool):
    """
    Collects the historic data from the eirgrid smart grid dashbiard
    """
    if len(areas) == 1 and areas[0] == 'all':
        areas = list(map(lambda x: x.name, [
            Area.co2emission,
            Area.co2intensity,
            Area.demandactual,
            Area.interconnection,
            Area.generationactual,
            Area.frequency,
            Area.windactual
        ]))
    for area in areas:
        if not area in Area._member_names_:
            raise Exception(f'Area "{area}" is not valid')

    if end:
        end_date = datetime.datetime.strptime(end, '%d/%m/%Y')
    else:
        end_date = datetime.datetime.now()
    start_date = datetime.datetime.strptime(start, '%d/%m/%Y')

    click.clear()
    for area in areas:
        filename = f'{area}_{region}.csv'

        df = collect_smartgrid(
            area=area,
            start_date=start_date,
            end_date=end_date,
            region=region
        )

        filename = OUT_DIR / filename
        print(f'\nWriting output for area: {area} to file: {filename}\n')
        df.to_csv(filename)

@cli_sync.command()
@click.option('-r', '--region', required=True)
@click.argument('areas', nargs=-1, required=True)
def sync(areas: tuple[str], region: str):
    """
    Fetches the most up-to-date data and syncs it
    """
    for area in areas:
        filename = OUT_DIR / f'{area}_{region}.csv'
        df = pd.read_csv(filename, index_col=0, parse_dates=True).dropna()
        df['EffectiveTime'] = pd.to_datetime(df['EffectiveTime'])
        last_updated = df.tail(1)['EffectiveTime'].item()
        print(last_updated)

        response = collect_smartgrid(
            area=area,
            start_date=last_updated,
            end_date=datetime.datetime.now(),
            region=region
        )
        df = pd.concat([df, response], ignore_index=True)
        df = df.drop_duplicates(keep='first', subset=['EffectiveTime'])
        df.to_csv(OUT_DIR / f'{area}_{region}.csv')

cli = click.CommandCollection(sources=[cli_collect, cli_sync])

if __name__ == '__main__':
    OUT_DIR.mkdir(parents=False,exist_ok=True)
    cli()
```

Listing 19: Python script `eirgrid_data_preprocessing.py` used to clean the EirGrid dataset

```python
def clean_interconnection():
```

```
    df = pd.read_csv(OUT_DIR / 'interconnection_roi.csv', index_col=0)
    df = df.pivot_table(index='EffectiveTime', columns='FieldName', values='Value')
    df = df.reset_index()
    df = df.melt(id_vars='EffectiveTime', var_name='FieldName', value_name='Value')
    df = df.loc[df['FieldName'] == 'INTER_EWIC']
    df.to_csv(OUT_DIR / 'interconnection_roi.csv')

def clean_frequency():
    df = pd.read_csv(OUT_DIR / 'frequency_all.csv', index_col=0)
    df['EffectiveTime'] = pd.to_datetime(df['EffectiveTime'])
    df = df.set_index('EffectiveTime')
    df = df.resample('15T').asfreq()
    df = df.reset_index()
    df.dropna(inplace=True)
    df.to_csv(OUT_DIR / 'frequency_all_resampled.csv')

def process_chunk(chunk: pd.DataFrame) -> pd.DataFrame:
    chunk['EffectiveTime'] = pd.to_datetime(chunk['EffectiveTime'])
    if 'Region' in chunk:
        unique_regions = len(chunk['Region'].unique())
        if unique_regions > 1:
            raise Exception('There are more than one regions in this file.')
        chunk = chunk.drop(columns='Region')

    df = chunk.pivot_table(
        index='EffectiveTime',
        columns='FieldName',
        values='Value',
        aggfunc='first'
    )
    df.columns = df.columns.to_series().apply(lambda x: humps.pascalize(x.lower()))
    return df

def process_file(file: Path) -> pd.DataFrame:
    df = pd.DataFrame()

    bar = tqdm(pd.read_csv(file, index_col=0, parse_dates=True, chunksize=600_000))
    bar.set_description(f'Processing {file.stem}')

    with ThreadPoolExecutor(max_workers=100) as executor:
        for chunk_pivot in executor.map(process_chunk, bar):
            df = pd.concat([df, chunk_pivot])
    return df

def clean():
    """
    Cleans the raw data files downlaoded using the  eirgrid_data_collection  script
    """

    files = list(map(
        lambda x: OUT_DIR / f'{x}.csv',
        [
            'frequency_all_resampled',
            'co2emission_roi',
            'co2intensity_roi',
            'demandactual_roi',
            'generationactual_roi',
            'interconnection_roi',
            'windactual_roi'
        ]
    ))

    click.clear()
    print(f'Reading {len(files)} files')

    df = pd.DataFrame()

    # Process files in parallel
    with ProcessPoolExecutor() as executor:
        for i, result in enumerate(list(tqdm(executor.map(process_file, files), total=len(files)))):
            if i == 0:
                df = result
            else:
                print(f'Applying join for {files[i].stem}')
                df = df.merge(result, on='EffectiveTime', how='outer')
    print(f'Processed all frames')

    # Group by year extracted from 'EffectiveTime'
    df = df.reset_index()
    df = df.sort_values(by='EffectiveTime', ascending=False)
    print(df)
    print('Writing complete output...')
```

```
    df.to_csv(k.CLEANED_DATA_DIR / f'eirgrid.csv')
    print(f'Wrote complete output\n')

    df['Year'] = df['EffectiveTime'].dt.year
    year_groups = df.groupby('Year')

    # Write each year's data to a separate file
    for year, group in year_groups:
        group.drop(columns='Year', inplace=True)
        outdir = k.CLEANED_DATA_DIR / f'eirgrid_{year}.csv'

        ixs = np.array_split(group.index, 1000)
        bar = tqdm(enumerate(ixs), total=len(ixs))
        for ix, subset in bar:
            bar.set_description(f'Writing output for {year} to {outdir.stem}')
            if ix == 0:
                group.loc[subset].to_csv(outdir, mode='w', index=False)
            else:
                group.loc[subset].to_csv(outdir, header=None, mode='a', index=False)

if __name__ == '__main__':
    clean()
```

Listing 20: Python script `cso_data_processing.py` used to process the CSO dataset

```
def process_fuel_mix():
    cso_df = pd.read_csv(k.RAW_DATA_DIR / 'cso_2020_23.csv')
    cso_df['VALUE'].fillna(0, inplace=True)
    cso_df.drop(cso_df[cso_df['Time Bands'] == 'All time periods'].index, inplace=True)

    cso_df['Day'] = pd.to_datetime(cso_df['Day'])
    cso_df['Hour'] = cso_df['Time Bands'].apply(lambda x: x.split(' < ')[1]).str.strip().replace('00:00
        ', '24:00').astype(str)
    cso_df['TimeDelta'] = pd.to_timedelta(cso_df['Hour'] + ':00')
    cso_df['EffectiveTime'] = pd.to_datetime(cso_df['Day'] + cso_df['TimeDelta'])
    cso_df.drop(columns=['Statistic Label', 'Day', 'Time Bands', 'UNIT', 'Hour', 'TimeDelta'], inplace=
        True)

    pivot_df = cso_df.pivot_table(
        values='VALUE',
        index='EffectiveTime',
        columns='Primary Fuel Output',
        aggfunc='first'
    )
    pivot_df.reset_index(inplace=True)
    pivot_df.set_index('EffectiveTime', inplace=True)
    pivot_df = pivot_df.drop(columns=['Net Generation']).multiply(2)
    pivot_df.to_csv(k.PROCESSED_DATA_DIR / 'fuel_mix_2020_23.csv')

if __name__ == 'main':
    process_fuel_mix()
```

Listing 21: Code implementation of Grey's Relational Analysis in Python based on Sallehuddin et al. [7]

```
class GreyRelationalGrade():
        def __init__(self, grc_target, columns):
        self.grc_target = grc_target
        self.columns = columns

    def df(self) -> pd.DataFrame:
        return pd.DataFrame(self.grc_target, columns=self.columns)

    def mean(self):
        return self.df().mean().sort_values(ascending=False)

def grg(features: pd.DataFrame, target: pd.Series, zeta=0.5, norm=False) -> pd.DataFrame:
    feature_data = features.values
    target_data  = target.values.reshape(-1, 1)
    data = np.concatenate([target_data, feature_data], axis=1)

    if norm:
        scaler = MinMaxScaler()
        data = scaler.fit_transform(data)
    abs_diff = np.abs(data - data[:, [0]])

    min_diff = np.nanmin(abs_diff)
    max_diff = np.nanmax(abs_diff)
    grc = (min_diff + (zeta * max_diff)) / (abs_diff + (zeta * max_diff))
    grc_target = grc[:, 1:]

    return GreyRelationalGrade(grc_target, features.columns)
```

Listing 22: MATLAB implementation of a z-score based normaliser

```
classdef StandardScaler
    properties
        Mean
        Std
    end

    methods
        function obj = StandardScaler()
            % StandardScaler Construct an instance of this class
            % This constructor takes no arguments
        end

        function obj = fit(obj, data)
            % fit Fits the scaler to the dataset
            % Computes the mean and standard deviation for each feature

            % Check if it's a cell
            if iscell(data)
                obj.Mean  = mean([data{:}],2);
                obj.Std = std([data{:}],0,2);
            else
                obj.Mean = mean(data, 2);
                obj.Std = std(data, 0, 2);
            end

            % To avoid division by zero, set stds of zero to one
            obj.Std(obj.Std == 0) = 1;
        end

        function Xnorm = normalize(obj, X)
            % normalize Applies the normalization to dataset X
            % Uses the Mean and Std from the fit dataset
            if isempty(obj.Mean) || isempty(obj.Std)
                error('StandardScaler/normalize: You must call .fit() beforehand.');
            end
            if iscell(X)
                Xnorm = cellfun(@(x) (x-obj.Mean)./obj.Std,X,UniformOutput=false);
            else
                Xnorm = (X - obj.Mean) ./ obj.Std;
            end
        end

        function Xdenorm = denormalize(obj, Xnorm)
            % denormalize Reverses the normalization from dataset Xnorm
            % Uses the Mean and Std from the fit dataset
            if isempty(obj.Mean) || isempty(obj.Std)
                error('StandardScaler/denormalize: You must call .fit() beforehand.');
            end
            if iscell(Xnorm)
                % If Xnorm is a cell array, apply the operation to each cell element
                Xdenorm = cellfun(@(x) x .* obj.Std + obj.Mean, Xnorm, 'UniformOutput', false);
            else
                % Apply denormalization to the whole array
                Xdenorm = Xnorm .* obj.Std + obj.Mean;
            end
        end
    end
end
```

Listing 23: MATLAB implementation of DataWindow class used to form the sequences for the LSTM network

```
classdef DataWindow
   properties
        WindowSize % Duration of each window in hours
        DateKey % Name of the date column in the data
        Data % The data table
        Sequences % The generated sequences
        Inputs % Input data for each sequence
        Targets % Target data for each sequence
        SequenceStartDates % Start dates for each sequence
    end

    methods
        function obj = DataWindow(data, windowSize, dateKey)
            if ~ismember(dateKey, data.Properties.VariableNames)
                error('DataWindow:InvalidDateKey', 'The dateKey does not exist in the data table.');
            end
```

```
17
18              obj.Data = sortrows(data, dateKey);
19              obj.WindowSize = windowSize;
20              obj.DateKey = dateKey;
21          end
22
23          function obj = gseq(obj, inputVars, targetVar)
24              % Generates sequences based on the input and target variables.
25              % Constructs input and target sequences separately for each window.
26
27              % Validate that the input variables and targets exist
28              mustBeMember(inputVars, obj.Data.Properties.VariableNames);
29              mustBeMember(targetVar, obj.Data.Properties.VariableNames);
30
31              % Get the start date
32              startDate = dateshift(obj.Data.(obj.DateKey)(1), 'start', 'day');
33
34              % Calculate the end date to ensure the last data point is included
35              lastDate = obj.Data.(obj.DateKey)(end);
36              % Adjust the end date to cover the last data point
37              if dateshift(lastDate, 'start', 'day') == lastDate
38                  % If last date is exactly at the start of a day, it starts a new interval
39                  endDate = lastDate;
40              else
41                  % Otherwise, extend the last interval to include the last date
42                  endDate = dateshift(lastDate, 'start', 'day') + hours(obj.WindowSize);
43              end
44              allDates = (startDate:hours(obj.WindowSize):endDate)';
45
46               % Initialize cell arrays for inputs and targets
47              obj.Inputs = cell(length(allDates)-1, 1);
48              obj.Targets = cell(length(allDates)-1, 1);
49              obj.SequenceStartDates = allDates(1:end-1);
50
51              % Assign data to sequences
52              for i = 1:length(allDates)-1
53                  sequenceMask = obj.Data.(obj.DateKey) >= allDates(i) & obj.Data.(obj.DateKey) <
                        allDates(i+1);
54                  sequenceData = obj.Data(sequenceMask, [inputVars, targetVar]);
55
56                  % Check for rows where all data points are zero and remove them
57                  nonZeroMask = ~all(sequenceData{:,:} == 0, 2);
58                  filteredData = sequenceData(nonZeroMask, :);
59
60                  if ~isempty(filteredData)
61                      obj.Inputs{i} = table2array(filteredData(:, inputVars))';
62                      obj.Targets{i} = table2array(filteredData(:, targetVar))';
63                  end
64                  %
65                  % obj.Inputs{i} = table2array(obj.Data(sequenceMask, inputVars))';
66                  % obj.Targets{i} = table2array(obj.Data(sequenceMask, targetVar))';
67              end
68              obj.Inputs(cellfun('isempty', obj.Inputs)) = [];
69              obj.Targets(cellfun('isempty', obj.Targets)) = [];
70              obj.Sequences = [obj.Inputs, obj.Targets];
71              obj.SequenceStartDates(cellfun('isempty', obj.Inputs)) = [];  % Update start dates
                    accordingly
72          end
73
74          function [date] = vlook(obj, idx)
75              % Returns the start date for a given sequence index.
76              % Ensure that sequences have been generated.
77              if isempty(obj.SequenceStartDates)
78                  error('DataWindow:EmptySequences', 'No sequences have been generated.');
79              end
80
81              % Check if the sequence index is within the valid range.
82              if idx < 1 || idx > length(obj.SequenceStartDates)
83                  error('DataWindow:IndexOutOfRange', 'The provided sequence index is out of range.');
84              end
85
86              % Retrieve the start date for the provided sequence index.
87              date = obj.SequenceStartDates(idx);
88          end
89
90          function [idx, seq, X, Y] = rlook(obj, date)
91              % Reverse lookup to find the sequence that includes a given date.
92              % Ensure the date is a datetime object.
93              if ~isa(date, 'datetime')
94                  error('DataWindow:invalidDateTime', 'Input must be a datetime object.');
95              end
96
```

```
            % Ensure that sequences have been generated.
            if isempty(obj.SequenceStartDates)
                error('DataWindow:EmptySequence', 'Cannot perform a reverse lookup with empty sequences.');
            end

            % Find the interval that the date falls into.
            % The index corresponds to the last sequence whose start date is less than or equal to the given date.
            idx = find(date >= obj.SequenceStartDates, 1, 'last');

            % If no such sequence is found, or the date is beyond the last sequence start date,
            % then the date does not correspond to any sequence.
            if isempty(idx) || date < obj.SequenceStartDates(idx) || date >= obj.SequenceStartDates(idx) + hours(obj.WindowSize)
                error('DataWindow:DateOutOfRange', 'The specified date is not within the range of any sequence.');
            end

            % Retrieve the input and target data for the identified sequence.
            seq = {obj.Inputs{idx}, obj.Targets{idx}};
            X = obj.Inputs{idx};
            Y = obj.Targets{idx};
        end
    end
end
```

Listing 24: MATLAB code to calculate the feature importance of the LSTM network

```
featureImportance = zeros(numFeatures, 1);
baselineRMSE = calculateOverallRMSE(YTest_norm, YPred_norm);

for i = 1:numFeatures
    XTest_permuted = XTest;

    % Loop over each sample and shuffle the feature values across all time steps
    for j = 1:length(XTest_permuted)
        numTimeSteps = size(XTest_permuted{j}, 2); % Number of time steps
        permutedIndices = randperm(numTimeSteps);  % Permute indices
        XTest_permuted{j}(i, :) = XTest_permuted{j}(i, permutedIndices);  % Shuffle the features values
    end

    % Make predictions with the permuted data
    YPred_permuted = predict(net, XTest_permuted);
    YPred_permuted_norm = YScaler.denormalize(YPred_permuted);

    % Calculate the performance metric with the permuted data
    permutedRMSE = calculateOverallRMSE(YTest_norm, YPred_permuted_norm);

    % Calculate the importance as the degradation in performance
    featureImportance(i) = (permutedRMSE - baselineRMSE) / baselineRMSE * 100;
end
```