uc3m | Universidad Carlos III de Madrid

Master in Statistics for Data Science
2024-2025

Master Thesis

"Optimizing Wind Turbine Placement in Wind Parks via Mixed Integer Optimization using Neural Network based Constraint Learning."

Simon Schmetz

Carlos Ruiz Mora
2nd Tutor complete name
Madrid the 29. of January

Acknowledgments

TODO Acknowledgments

ABSTRACT

This thesis combines Linear Optimization with Constraint Learning to optimize wind turbine placement for maximum performance in a predefined area under randomly distributed wind. A Neural Network is trained on simulated data to model the impact of turbine positioning on power output. This model is integrated as a constraint in a linear optimization problem, and the problem evaluated for a current state-of-the-art wind farm configuration.

CONTENTS

1

INTRODUCTION

With the clean energy transition currently taking place in europe with ambitious targets for 2030 and beyond [11] , wind energy is playing a central role in that transition, with wind energy expected rise to 50 % in the EU energy mix. [5] With wind energy thus expected to become the main contributer to the EU's energy production and large potentials identified for both onshore and offshore parks [1] attempts to optimize all parameters of windparks with even minor power efficeny improvement can be expected to yield significant returns in absolute power due to the scale of future wind energy production.

TODO : REWRITE As discussed in the Introduction, one of the main goals in the optimization of layouts, is to reduce the negative impact wake effects inbetween windturbines have on overall power generation with yield reduction of up to 15% mainly due to reduced windspeeds in wake reagions. Optimizing the farm for overal minimal wake exposure inbetween windturbines can thus bring potentially significant power yield increases. [16] [19]

As a contribution to increasing power efficeny on future wind farms, this thesis is dedicated to a new approach for optimizing the placement of a fixed number of wind turbines in a predefined area (typically a square). To solve this optimization problem, a extension to the pyomo python library is used, that allows the introduction of Neural Networks to the optimization problem as constraints. [3] This extension allows for introducing a Neural Neural Network to model the effects of wind turbine placement relative to each other on power production for the respective windturbines. Introducing this model to the optimization problem defined in pyomo then allows for the optimization of overall power productions across all wind turbines in the wind park. The optimization problem in its simplest form can be defined as

$$\max_{\mathbf{x},\mathbf{y}} \sum_{i=1}^{n} f_{Power,\mathrm{NN}}(x_i, y_i; \mathbf{x}, \mathbf{y}) \tag{1}$$

$$\text{s.t.} \quad X_{\min} \leq x_i \leq X_{\max}, \quad \forall i \in \{1, \ldots, n\} \tag{2}$$

$$Y_{\min} \leq y_i \leq Y_{\max}, \quad \forall i \in \{1, \ldots, n\} \tag{3}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq d_{\min}, \quad \forall i \neq j \tag{4}$$

where:

- (x_i, y_i) are the coordinates of turbine i out of n total turbines,
- $f_{\mathrm{NN}}(x_i, y_i; \mathbf{x}, \mathbf{y})$ is a neural network approximating the power output for each turbine i ,
- $X_{\min}, X_{\max}, Y_{\min}, Y_{\max}$ define the boundaries of the rectangular placement

- d_{min} is the minimum distance between any two turbines.

To create a model optimally fit to the needs of the optimization problem, the model is trained on data specifically generated with the FLORIS wind farm simulation tool for optimal coverage of the parameter space of the optimization.

To simplify the problem, the surface below the turbines is assumed to be perfectly flat and equal wind speed is assumed along the entire hight of the turbines.

Data Generation and Neural Network:

(...)

Optimization Problem:

(...)

This thesis is structued according to the two main steps required to solve the optimization problem as presented above.

2

STATE OF THE ART

Since the arrival of large-scale wind turbine farm operations as part of energy infrastructure, optimizing the positioning of the individual wind turbines relative to each other to mitigate wake effects reducing the total power output is the subject of scientific investigation. As this Thesis is an attempt to apply a novel constraint learning method introduced by Alcantara and Ruiz in [3] to the optimization of wind farm layouts, the following state-of-the-art is split into two pieces, with the first investigating the current state of the art of wind farm optimization and the second a brief introduction into recent developments made in the field of constraint learning.

Optimization of Wind Farm Layouts

As discussed in the Introduction, one of the main goals in the optimization of layouts is to reduce the negative impact wake effects between wind turbines [19]. Historically, the initial approaches were to use rule of thumb approaches by setting up the layout as a grid and with the distance between wind turbines in the dominant wind direction between 8 and 12 times the turbine rotor diameter and spacing perpendicular to the dominant wind direction 4 to 6 times the turbine rotor diameter [6] [16].

These methods have evolved to with most current research pursuing the goal of maximizing the Anual Energy Production (AEP) of wind farms in the context of stochastic optimization as done in [26] [19].

The core of any of the most recent optimizations is a wake model which becomes part of the objective function by representing the wake effects on power output. These models can be categorized as [31]:

1. Experimental Methods
2. Numerical modeling
3. Analytical/semi-empirical modeling
4. Data-driven modeling

While experimental methods and numerical models might be the most precise models available for wake modeling, one of the challenges that come with the optimization is that the model has to be able to be introduced into the current state-of-the-art solvers as part of an objective function, leading to the prevalent use of analytical wake models like Gaussian wake model and the 3D wake model [31]. With advancements in machine learning the field of data-driven modeling is meanwhile expanding, with successful attempts of introducing Neural Networks and other Machine Learning frameworks into

optimizations of wind farm layouts. Generally, either experimental data or data from numerical modeling (more prevalent) is used to train a chosen model type. The resulting model is then introduced into the optimization problem, as done in [33] [7] [28] [29].

Constraint Learning

The term Constraint Learning, defined as "finding a set of constraints, a constraint theory, that satisfies a given dataset" by Raedt et al. is the intersection of machine learning and optimization or more in practical terms, the introduction of machine learning models into optimization problems as constraints. As the models learned (from a given data set), the constraints resulting from such a model are thus equally learned. [13]

For this thesis specifically, the novel constraint learning method of decomposing a neural network into a set of constraints using a big M approach introduced in [3] and [4] is the foundation. Similar approaches of embedding machine learning models into optimization problems have been taken for Decision Trees and Random Forests in [9] or again for Neural Networks but without integer variables required as done in [12]. A survey performed by Fajemisin et al.[14] shows how the field is currently emerging with an increase in publications in recent years and most publications revolving around the Embedding of Neural Networks and Decision Trees/Random Forests.

3 FARM POWER MODEL

The first central component to the optimization of the wind farm layout is to generate a data drive surrogate Model that can be introduced into the optimization problem and be solved by a solver. As detailed in the introduction, more specifically the aim is to use the distCL extension to the pyomo python package, requiring a small Neural Network as a surrogate model. The following chapter documents the steps taken to generate such a model. To train such a Neural Network, data is required that covers the parameter space of the optimization to prevent extrapolation by the model. Therefore, the chapter starts by explaining how the open-source wind farm simulation tool FLORIS® was used to generate a dataset and what the simulations behind this dataset are. Then, the fundamentals of Neural Network architecture and training are briefly introduced, before the model of the interactions for two turbines is trained and evaluated.

3.1 DATA SOURCE

The power output of wind turbines and wind farms as a whole, is fundamentally connected to the aerodynamic conditions in the airflow every wind turbine experiences, with wind speed as the biggest factor relevant to how much power a wind turbine can generate. The main effect reducing the power generated by a wind turbine is to be positioned in the wake downwind of another turbine. This power reduction is primarily a result of the reduction in windspeed joined with the increased turbulence in the wake airflow (see 1) [20]. Moving further downstream of a wind turbine, the wake gradually mixes with the outer airflow and thus again increases windspeed until the entire airflow reaches a new homogeneous air speed [22]. Thus, even if a wind turbine is positioned in the wake of another wind turbine, the greater the distance between the two, the less the second wind turbine is affected. Ideally, wind turbines are not positioned in the wake of other wind turbines at all.

Figure 1: Visible wind turbine wakes due to contrail generation [32]

With the overall goal of creating a model that models how the interaction between wind turbines (induced by the wake) is related to the wind turbines relative positioning to each other and the wind conditions as input to an optimization model, a tailor-made dataset from simulation results is the easiest to handle to ensure that the data fits the use case well. Using self-generated simulation data allows to set the parameter space of the model as required for the different variants of the optimization problems explored in the second part of this thesis. After investigating two Python based open source wind farm simulation tools FLORIS (FLORIS official website) and PyWake (PyWake Documentation), FLORIS was chosen after a brief comparison between the two due to apparent ease of use. The FLORIS is a wind farm simulation tool developed by the National Renewable Energy Laboratory (NREL) as one of multiple tools containing models of varying complexity available for different use cases. FLORIS is a control-systems-oriented toolbox and therefore contains a lower complexity model yielding fast simulation results. This allows the automatic generation of large quantities of data in a relatively short time. As the data is coming from simulation results, the data is deterministic even though there is an underlying uncertainty attributed which can be quantified by FLORIS if required.

For this thesis, depending on the optimization problem, FLORIS is given a grid of the following parameters:

- Position of Wind Turbines
- Wind Direction
- Wind Speed
- Wind turbolence

defining the parameter space limits with evenly spaced data points to then generate the corresponding simulation data, with the simulation outputs being the generated power by the individual wind turbines as well as by the farm as a whole (e.g. the sum of individual power outputs). The configuration of all of these parameters as well as the number of wind turbines can be modified at will in this setup and the result is a dataset with the rows corresponding to a specific combination of parameters and the resulting power outputs of the turbines. As turbine type, an IEA 10-MW is used for data generation due to its repeated use as baseline turbine in other scientific works like [21] and [18], a more exact technical specification of the turbine can be found in [10]. The configurations of the specific dataset generated for the individual optimization problem formulations are documented in the corresponding Sections of Chapter 4.

3.2 INTRODUCTION TO NEURAL NETWORKS

To model the relationship between the attributes of an incoming airflow to a wind turbine and the output generated by the same wind

turbine many surrogate models could be chosen. As the model generated for this thesis is created to be then introduced into the pyomo extension referenced in Chapter 1, the model has to be compatible with said extension. That is why the model chosen is a simple neural network with a limited number of hidden layers and nodes, with the size of the Neural Network being limited due to its introduction into the optimization problem.

In the following section, a brief introduction is given to how Neural Networks work and are trained, including some regularization techniques.

Arcitecture of a Neural Network

Fundamentally, Neural Networks represent Graph Networks consisting of function blocks as the nodes, in the case of Neural Networks called perceptron (or Neurons as more general terms) and arcs which correspond to the inputs/outputs of a given perceptron. In simple words, the inputs to the Neurons are summed up and introduced as argument into a function $f()$ which yields an output y, representing the output of the given perceptron. These Neurons are organized in layers, which correspond to the row-type structure Neural Networks are usually represented in and each perceptron of one layer is connected to every other perceptron of the following layer. The following layer in this case means in the direction of flow, in visualizations usually from left to right. The arcs thus in simple terms correspond to a single number and the Neurons to the action of summing up all inputs and then applying the unspecified function $f()$ to generate an output. This process is repeated for all Neurons in the network, with the first layer of the network taking as input the values of the values of the given data features (the input values to the model) and the last layer producing outputs corresponding to the output value(s) of the model. The number of Neurons in this first and last layer corresponds to the task the Neural Network is supposed to perform. If the goal is to identify handwritten numbers 1-9 from 50x50 pixel images, the input layer might have 50x50 Neurons for the value of each pixel, and the output layer 9 layers with the output value of each perceptron representing how much the network "thinks" the given image shows the corresponding numbers 1-9. A schematic of this architecture is given in Figure 2.

As shown in Figure 3 the output of a Neuron is slightly more complicated as described before, as the output y is generated by summing up the inputs x multiplied by a corresponding weight w together with a bias b and introducing this summation as an argument into an activation function $f()$. In this process, the weights w represent a weight to give importance to the individual inputs and the bias b serves to set a minimum output value that will always be reached, regardless of the inputs.

The activation function $f()$ is called activation function, as in its simplest form it represents a step function that decides if a neuron activates or not, e.g. takes the binary values $0, 1$ for a given threshold.

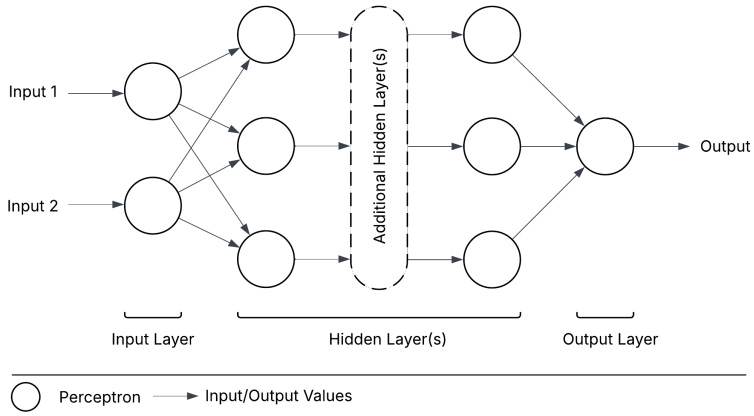

Figure 2: Schematic of Neural Network Architecture

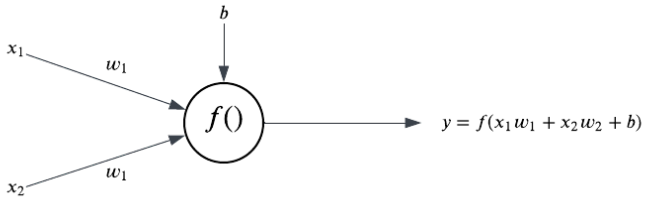

Figure 3: The output of a neuron is generated by applying the activation function to the sum as the weighted inputs $w_i x_i$ and the bias of the neuron b

Contrary to the human brain where neurons are indeed binary, most Neural Networks resort to an activation function whose outputs are not binary but deliver continuous values between 0 and 1 to avoid the boundary issues that occur with binary thresholds. The most common function used instead of a step function is the sigmoid function, which roughly corresponds to a continuous version of the step function, with $\sigma(x) \approx 1$ for $x \to \infty$ and $\sigma(x) \approx 0$ for $x \to -\infty$.

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

This relationship also becomes apparent when plotting both of those functions over each other, as shown in Figure 4. An alternative to the sigmoid as activation function is the Rectified Linear Unit (ReLu) function, or in simple words, the maximum function, defined as follows.

$$\mathrm{ReLU}(x) = \max(0, x) = \begin{cases} 0 & \text{if } x < 0, \\ x & \text{if } x \geq 0. \end{cases}$$

The main general applicable benefit of the ReLu function is that it has a constant, non-vanishing gradient for both directions (see Figure 4), leading to better gradient propagation, an attribute that is relevant in the context of the backpropagation algorithm briefly discussed in Section 3.2. [15] Beyond this, the ReLu function allows for embedding

the Neural Network into a conventional optimization problem as will be discussed in Section 4.2.

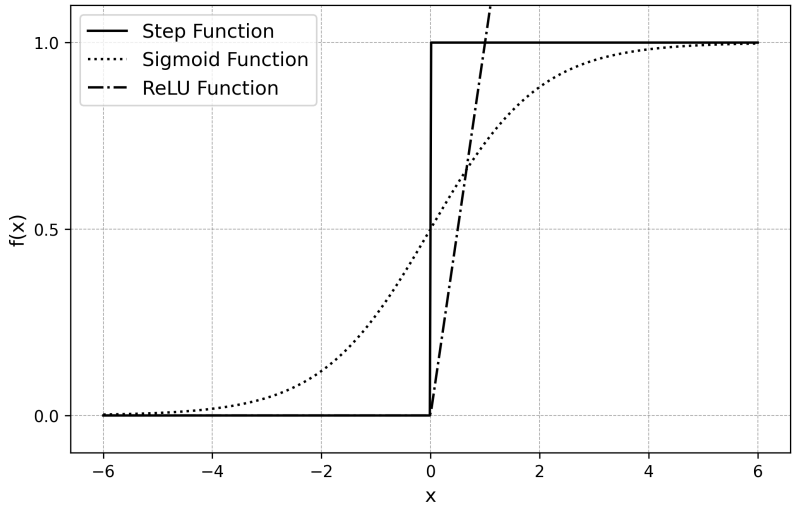

Figure 4: The Sigmoid Function being close to the Step Function without being as sensitive to slight changes in x due to its continuity. Alternatively, the Rectified Linear Unit function (ReLu) can be used as activation function.

[24]

Training a Neural Network

With the general structure set up, and assuming that a layout has been chosen for the neural network (e.g. number of layers and number of their corresponding neurons), the training of the neural network corresponds to adjusting the weights w and the biases b of each neuron in a way, that allows the model to perform the task it is given well. What it means for the model to perform well is defined by a *Loss Function*, which defines a relationship between the output of the model and the correct output (defined by training data) and gives a Loss as output, which is some sort of delta between model prediction and truth. What it means for a model to perform well heavily depends on the task (regression, binary classification, multiclass classification etc.), but regardless of the task, the goal is always to minimize the Loss Function. A well-known Loss Function in regression is the Mean Squared Error (MSE)

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

The training of the Neural Network thus corresponds to adjusting the weights and biases in a way that minimizes the chosen loss function. The algorithm most commonly used for this is called *Backpropagation*. [24]

Before diving into how the algorithms work exactly, it makes sense to first define specifically what parameters have to be optimized, e.g. all weights and biases of the network to be optimized. A common

notation of the individual weights is as w^l_{jk} with l being the layer of nodes into which w^l_{jk} are the weights of its inputs (weighing the values from the outputs of the $(l-1)^{th}$ layer) and j as the neuron from the l^{th} layer as well as k the neuron of the $(l-1)^{th}$. Similarly, the biases are written as b^l_j with l as the layer and j the neuron in that layer as as can be seen for both the weights and bias in an example in Figure 5.

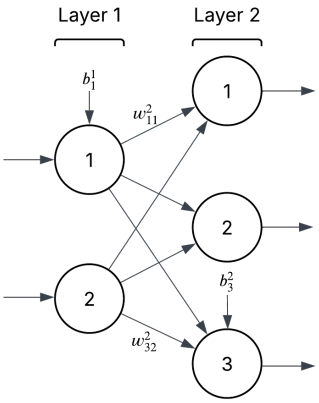

Figure 5: Notation of Neural Network weights and biases with weight w^l_{jk} and l being the input layer, j the neuron from that input layer and k the neuron of the $(l-1)^{th}$ output layer. Similarly, the bias as b^l_j with l as the layer and j the neuron.

Using this notation, the output of a given neuron a^l_j can be calculated as

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right),$$

this expression can be further simplified,by writing weights and biases in matrix form, using for each layer, a weight matrix w^l for all weights input into the given layer l and a bias vector b^l. By doing so, the output vector a^l of the entire layer l can be calculated as.

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

With the notation for weights and biases established, the goal becomes tuning them in such a way, as to minimize a chosen Loss function as discussed above. This is done by using training data with a known correct output and changing weights and Biases in such a way, that moves the Neural Network towards outputting the true known value. The algorithm to do so essentially corresponds to:

1. Introduce Training Data into the Neural Network

2. Evaluate the Loss Function for the Training Points (with small random values for weights and biases for the first iteration) and evaluate the gradients of the Loss function for the gradients of

the individual weights and biases ($\frac{\Delta Loss}{\Delta w^l_{jk}}$, $\frac{\Delta Loss}{\Delta b^l_j}$) using backpropagation [1]

3. Update the weights and biases according to a learning rate ρ weighted by the specific weight or biases gradient like $w^l_{jk} = w^l_{jk} - \rho \frac{\delta Loss}{\Delta w^l_{jk}}$ and $b^l_j = b^l_j - \rho \frac{\Delta Loss}{\Delta b^l_j}$

4. Repeat process n Epochs, with one Epoch being the algorithm passing through all training points/batches of points once

This algorithm represents a gradient descent algorithm with learning rate ρ corresponding to the step length while evaluating the partial derivatives $\frac{\Delta Loss}{\Delta w^l_{jk}}$, $\frac{\Delta Loss}{\Delta b^l_j}$ is the critical step. Here the backpropagation algorithm gives an efficient method of finding these partial derivatives by evaluating the individual error contributions of each Neuron and how errors accumulate as the inputs move through the Network. A more extensive explanation of how backpropagation works can be found in [25]. As common for gradient descent, finding the global optimum for the weights and biases is not guaranteed and even improbable if many local minima exist as they do for the many parameters of a Neural Network. Nonetheless, using gradient descent it can be hoped to find a good local minimum. [17] [25]

Regularization

Like for most machine learning models, Neural Networks run at risk of overfitting. Neural Networks are especially prone to overfitting due to their many degrees of freedom represented by the many biases and weights, potentially exceeding the number of training samples.

To prevent overfitting, regularization techniques like Cross-Validation can be applied in training like for any other model. One very specific regularization method for Neural Networks is *Dropout Learning*, which corresponds to randomly removing Neurons from the network (by effectively setting their activation function and thus their output to 0) for each training observation. By other nodes having to "stand in" for the dropped-out nodes, nodes are prevented from developing overspecialization. [17]

Notes - many Degrees of Freedom, many local maxima - Adaptive Moment Estimation (ADAM) optimization

Estimating Distributions using Neural Networks

Neural Networks can also be used to estimate distributions of a random variable. Two possible approaches are to so-called Quantile Neural Networks and Distributional Neural Networks. While their Architecture is that of a normal Neural Network as discussed in the previous section, they differentiate in the output layer, where the Quantile Neural Network has outputs that correspond to the probability for a

1 The Gradient can either be evaluated for each training point individually or for batches of training points

specific quantile while the distributional Neural Network outputs the distribution, like in Figure 8 for a normal distribution.

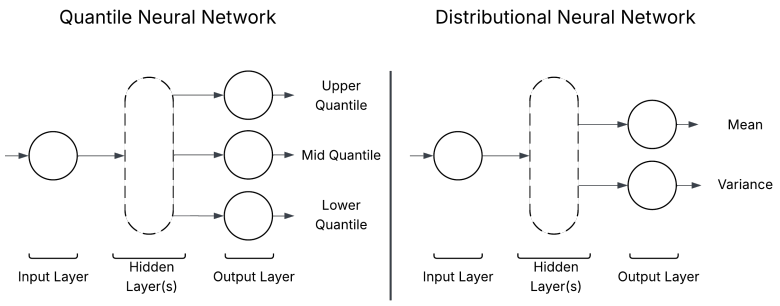

Figure 6: Neural Networks to estimate a distribution by estimating specific quantiles (left) or the parameters of a distribution (right)

The different outputs from both neural Networks require appropriate Loss functions. For Quantile Neural Networks, the Pinball Loss (Quantile loss, for more see [27]) is used while in the case of Distributional Neural Networks, the Likelihood (like in the form of the Negative Log Likelihood) can be used as commonly used in classical statistics for parameter estimation.

[2] [23]

3.3 MODELLING PIPELINE

With the theoretical foundations layed out, this section is going to treat the actual training process of the Neural Networks to be introduced into the optimization. Part of the challenge of training Neural Networks for this specific purpose is, that the objective in optimizing the Neural Network is to not only minimize the cost function, but also to minimize the number of model parameters, e.g. to minimize the total number of neurons the network consists of. This is due to the fact, that each neuron corresponds to an additional three linear constraints in the optimization problem, leading the number of constraints to grow as $\Delta n_{constraints} = 3 \times \Delta n_{neurons}$. In general, individual models are trained for each optimization problem seen in Chapter 4 tailored to their individual needs and parameter space. This is why this chapter will detail the main components of the modeling process and parameters that will be defined according to the optimization problem later on. The exact configuration of the models will then be defined together with the optimization problem.

The modelling pipeline contains three main components, with the elements consisting of the data generation, hyperparameter tuning (training) and the final model training.as seen in in Figure **??**, where data generation corresponds to running the required simulations via FLORIS for the given parameter space, hyperparameter tuning corresponds to the training and evaluation of different model configurations and final model training as the step in which the final model that is to be introduced into the optimization problem is trained.

TODO : Consits of two turbines
TODO: Validate Model ?
TODO: FLOW

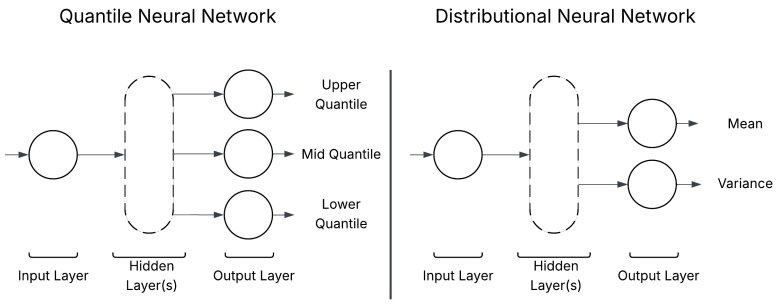

Figure 7: Neural Networks to estimate a distribution by estimating specific quantiles (left) or the parameters of a distribution (right)

Data Generation

The Data Generation corresponds to the creating of a dataset corresponding to a grid that is set up within the parameter space of the optimization problem. The objective is to generate a dataset that covers the parameter space with sufficient density to train a model which is able to accurately interpolate between the data given points. Each data point corresponds to a individual simulation performed using the tool FLORIS, allowing for an array of inputs and devlivering an array of outputs. As inputs, the following variables can be defined as part of the parameter grid:

- x_range
- y_range
- wind_speeds
- wind_directions
- turbulence_intensities

Where x_range and y_range correspond to the relative distances of the second wind turbine to the first, wind speed corresponds to wind velocity in m/s, direction to the incoming wind angle in Degrees (with 0° corresponding to norht, 180° to south) and turbolence intensities as a measure of turbolence for the incoming airflow. Figure **??** shows a simulation configuration for (PARAMETERS).

TODO: Insert plot from FLORIS

The method implemented in [LINK TO GITHUB], takes a range of values together with a specified number of chosen number of steps for the specific parameter and performs simulation for all combinations for the given ranges to fill the entire grid. As the number of steps or limits may be different for the different parameters, the grid

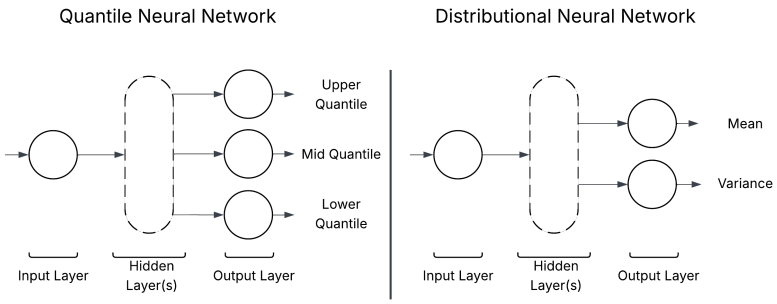

Figure 8: Neural Networks to estimate a distribution by estimating specific quantiles (left) or the parameters of a distribution (right)

is not necessarily equally spaced for all parameters, but for each parameter individually is constant, e.g. the distance between values is the same for the entire range. Depending on the configuration of the problem, some parameters may be constant. If for example a variable turbolence intensity is not taken into account for a given optimization, a constant value is set and the data is generated for that constant value. The model trained on this dataset is thus constrained in use to the assumption of that specific constant, a limitation that is important to keep in mind further down the road when the initial dataset generation might not be as concious any more.

The ouputs of the simulation are:

- Total Power Generation
- Power generated by Turbine 1
- Power generated by Turbine 2

where the total power generation corresponds to the summation of the power generated by the individual turbines. With the objective of maximizing total farm output, the toal power generation will now be used as target for the model in the following steps.

Neural Network Configuration

The next step in the process corresponds to the hyperparameter tuning of the Neural Network. According to the chosen type of Neural Network, (distributional Neural Network or single output Neural Network) a apropriate loss function has to be chosen first. The data is then split into a training and a evaluation set and the model is trained on on the training partition using the algorithms detailed in 3.2, using backpropagation/gradient descent as well as dropout learning for regularization for a chosen number of times. TODO ADD CONVERGENCE ?

Once the training process is ended, the model is used to predict for the test partition of the data and the results evaluated by scoring the predictions using the Mean Squared Error (MSE).

This process is repeated for a grid of the hyperparameters:

- Number of Hidden Layers
- Number of Nodes in each hidden layer

and a plot like shown in Figure 9 generated to decide what the best configuration of the Neural Network is.

Figure 9: Exemplary Output of Hyperparameter tuning

The final decision on what is the ideal model configuration is not only based on the best performance, but via the trade off between complexity and output, with the goal of generating a model that is as small as possible. The kind of optimization problem the model will be used to solve is further taken into account, with complex optimization problems having less tolerances for a complex model.

Final Model Training

After having chosen a specific model configuration, a model is then trained using that configuration. This is the model that eventually is introduced into the optimization problems of detailed in Chapter 4.

3.3.1 *Potential future Improvements in the Modelling Process*

- Data points not evenly spaced - Different number of nodes in each hidden layer

4

OPTIMIZATION

With the model of farm/turbine power established, the following section treats the embedding of the trained model(s) into a stochastic optimization problem. The underlying theory of optimization under uncertainty and constraint learning is introduced before the actual optimization problems are defined, required models trained and results discussed.

4.1 OPTIMIZATION UNDER UNCERTAINTY

While some things in life are certain, most are not and this is as true for optimization as for anything else. That means, that while optimizing anything from next year's crop to tomorrow's energy pricing might be done by assuming fixed values for the parameters of an optimization problem, in reality, most of these parameters contain uncertainty. The field of Stochastic Programming is occupied with finding methods that allow for introducing these uncertainties into optimization problems.

One way to approach these uncertainties is to split the problem into scenarios. A bakery for example has to decide on how many Baguettes to bake for the next day to maximize its profit. Baking too few Baguettes will lead to missing out on potential sales while baking too many baguettes will mean that the demand is fulfilled but the money invested in the excess number of baguettes is lost. Assuming the mean number of baguettes bought every day is 1000, the price to buy a baguette is 1 and the cost to produce a baguette is $0,2$ the classical approach to solving such a problem would be by the following formulation [1]:

$$\max_x \quad (1.00 \cdot \min(x, 1000) - 0.20 \cdot x)$$

To introduce the two scenarios additional scenarios of the demand being 10% lower (900 Baguettes) and 10% higher (1100 Baguettes) can be added. Assuming that the mean demand has a 50% probability of occurring and the 10% demand increases a 20% probability and the decrease a 10% probability, the problem can be modified to maximize the expected profit across these three scenarios by the formulation

$$\begin{aligned}\max_x \quad & 0.5 \cdot (1.00 \cdot \min(x, 1000) - 0.20 \cdot x) \\ & + 0.3 \cdot (1.00 \cdot \min(x, 900) - 0.20 \cdot x) \\ & + 0.2 \cdot (1.00 \cdot \min(x, 1100) - 0.20 \cdot x)\end{aligned}$$

1 x of course non-negative

The result from this optimization problem would be the Expected profit and how many Baguettes are the optimal number of Baguettes to yield the maximum Expected profit across all scenarios. This would be optimal assuming there exists no more information about the next day's demand, meaning that by using this approach the total profit over a long time would be maximal. In case there is more information regarding the next day's demand the probabilities that give the weights in this optimization might shift, with one scenario potentially reaching probability 100% if there were to be absolute certainty that the next day's demand would be for example 1100 baguettes. As having such exact information is however very rare, the best solution will be in most cases to maximize the profit Expectation.

The obvious connection to conventional statistical analysis is that the demand is a random variable that can take multiple values, in this case, we assumed it to be a discrete random variable Y with support $900, 1000, 1100$ even though in real-world applications the demand of baguettes will move somewhere between $[0, \infty]$. Finding the Expectation for such a discrete random variable can be done as

$$\mathbb{E}[X] = \sum_i x_i \cdot \mathbb{P}(X = x_i) = \sum_i x_i p_i$$

or for continuous random variables

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x \cdot f_X(x)\, dx$$

Using these two expressions, optimization problems can thus be formulated to optimize the Expectation of objective functions containing random variables. [8]

4.2 CONSTRAINT LEARNING

Constraint learning refers to introducing a model that has learned relationships between certain variables from data into an optimization problem. In the case of constraint learning, the model gets more specifically introduced into an optimization problem as part of a constraint. As many real-life relationships struggle to be represented by explicit function to be defined as objective function or constraint, introducing machine learning models to optimization problems opens up many new possibilities [14].

In the case of Neural Networks, one way of introducing a Network as a constraint into an optimization problem is by recognizing that when using the Rectifier Linear Unit (ReLu) Function is used as an activation function with the (linear) sum of neuron bias and weighted inputs $\tilde{v}_i^\ell$ being the function argument

$$v_i^\ell = \max(0, \tilde{v}_i^\ell) = \max(0, b_i^\ell + \sum_j w_{ij}^\ell v_j^{\ell-1}) \tag{5}$$

the function can be rewritten as the following constraints

$$v_i^\ell \geq \tilde{v}_i^\ell \tag{6}$$
$$v_i^\ell \leq \tilde{v}_i^\ell - M^{\text{low}}(1 - j_i) \tag{7}$$
$$v_i^\ell \leq M^{\text{up}} j_i \tag{8}$$

with $j_i \in \{0, 1\}$ a integer variable such that

$$j_i = \begin{cases} 0 & \text{if } \tilde{v}_i^\ell < 0 \\ 1 & \text{if } \tilde{v}_i^\ell > 0 \end{cases} \tag{9}$$

This decomposition allows for introducing a Neural Network of limited size into an optimization problem by decomposing it into a set of linear constraints of the form shown above. [3]

4.3 THE TWO TURBINE PROBLEM

Optimizing the positioning of two wind turbines can be expressed as optimizing the relative position of a second wind turbine T_2 to a fixed first turbine T_2, defined by the relative distances Δx and Δy. Both Δx and Δy are constrained by minimum distance Δ_{min} to T_2 and maximum distances $\Delta x_{max}/\Delta y_{max}$ to make the problem bounded. This problem can be visualized as shown in Figure 10.

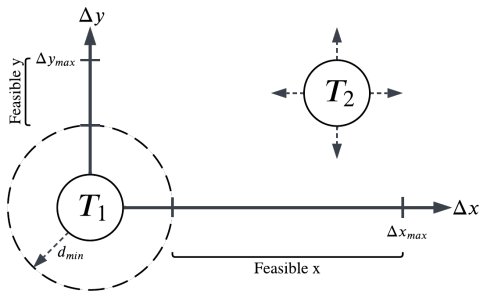

Figure 10: Optimizing the relative position $\Delta x/\Delta y$ of a second wind turbine T_2 to a fixed first turbine T_2, constrained by minimum distance d_{min} to T_1 and maximum distances $\Delta x_{max}/\Delta y_{max}$

The objective function to be optimized is the total power generation, e.g. the sum of power generated by both turbines. This objective is a function both of the position of the wind turbine as well as of the wind conditions like wind direction and wind speed.

$$f_{totalPower}(x, y, \text{windspeed}, \text{wind direction}, (...))$$

Differently from the geographic coordinates, th wind condition parameters like windspeed are inherently not deterministic and follow distributions like the normal distribution as shown in Figure 11.

The two main challenges of the optimization of the shown two turbine problem are thus:

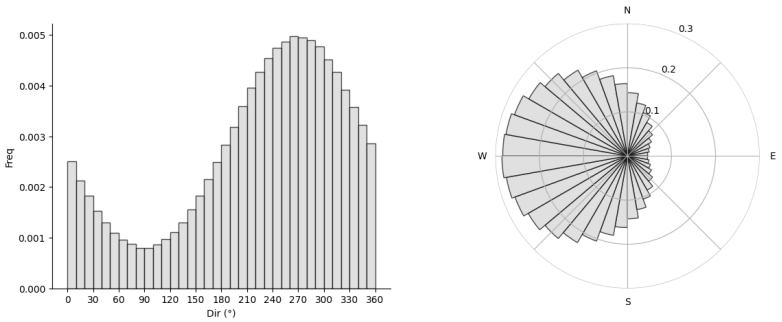

Figure 11: Histogram and Polar Plot across radiants for a normally distributed wind direction probability density function with mean West

1. Introduce the complex relationship between turbine position and wind conditions and power output into the optimization problem
2. Introduce the non-deterministic nature of wind conditions into the optimization problem

The first of these challenges is tackled by applying the Neural Network models discussed in Section 3.2 and introducing them into the optimization problem via constraint learning as described in Section 4.2. For the second problem, multiple approaches are now explored in the following subsection.

Deterministic Optimization: Optimization for main Wind direction and Speed

To begin solving the problem, the simplest approach is taken by assuming the wind conditions to be discrete. In application, this might be analogous to getting the Expectation of the joined probability distribution of all wind condition parameters. Taking these parameters as constant and homogeneous across the entire parameter space, the result is an objective function that is effectively only dependent on the relative positions of the turbine with the previously discussed geometrical constraints.

$$\max_{\mathbf{x},\mathbf{y}} f_{Power,\mathrm{NN}}(\Delta x, \Delta y) \tag{10}$$

$$\text{s.t.} \quad \Delta x \leq X_{\max} \tag{11}$$

$$\Delta y \leq Y_{\max} \tag{12}$$

$$\sqrt{(\Delta x)^2 + (\Delta y)^2} \geq d_{\min} \tag{13}$$

where:

- $(\Delta x, \Delta y)$ are the relative distances of the two turbines,
- $f_{Power,\mathrm{NN}}(\Delta x, \Delta y)$ is a neural network (deterministic) approximating the total power output ,

- $X_{\max}, Y_{\max}$ define the maximal distance the two turbines can be placed apart
- $d_{\min}$ is the minimum distance between the two turbines.

When solving this problem as defined using Gurobi, a major challenge becomes apparent as the result shows that there is an infinite amount of equally optimal solutions (e.g. degeneracy of the problem, see [30] for an exact definition of degeneracy) everywhere outside the wake of turbine 1. This phenomenon is very intuitive when generating a heatmap of the power generation for all possible positions of turbine 2, as shown in Figure 13

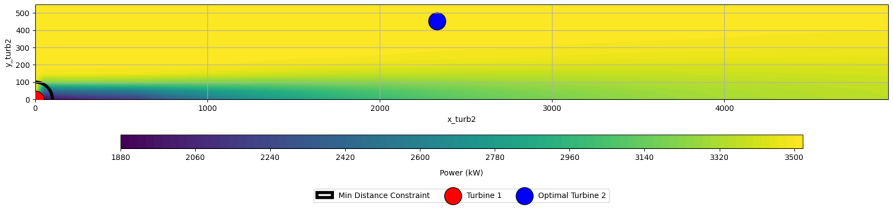

Figure 12: Deterministic optimization of the relative position of two windturbines relative to each other with winddirection 270° and constant windspeed

This phenomenon will have to be kept in mind going forward while introducing a distribution for wind condition parameters might reduce the number of global optima.

Stochastic Optimization: Power weighted by probability

As the deterministic optimization does not take into account the distribution of wind condition parameters and having found that the deterministic approach leaves a range of infinite global optima outside of the wake of turbine 1, the next step is proceeding with the introduction of uncertainty in the form of probability distributions of wind condition parameters into the problem. This step primarily changes the objective function, which is set up such that the neural network takes in both position and wind condition parameters. the output of this neural network is then weighed by the probability of the specific wind condition occurring and the Expected power is calculated by integrating (discretized as a sum) across all combinations of wind condition parameters.

$$\max_{\mathbf{x},\mathbf{y}} \sum_{i=1}^{n} f_{Power,\mathrm{NN}}(\Delta x, \Delta y, \text{wind condition}) \cdot p_{n,\text{wind condtion combination}} \tag{14}$$

$$\text{s.t.} \quad \Delta x \leq X_{\max} \tag{15}$$

$$\Delta y \leq Y_{\max} \tag{16}$$

$$\sqrt{(\Delta x)^2 + (\Delta y)^2} \geq d_{\min} \tag{17}$$

where:

- $(\Delta x, \Delta y)$ are the relative distances of the two turbines,
- $f_{Power,\mathrm{NN}}(\Delta x, \Delta y)$ is a deterministic neural network approximating the total power output for turbine positions and wind conditions
- $X_{\max}, Y_{\max}$ define the maximal distance the two turbines can be placed apart
- $d_{\min}$ is the minimum distance between the two turbines
- n is the index of the discretized possible combinations of wind conditions

In practice, this approach comes with challenges as it requires the constraints representing the neural network to be realized n times for each wind condition combination. With one instance of the neural network already adding a significant amount of constraints to the optimization model, realizing those constraints n times may lead to an overflow of constraints, making the model too difficult to solve.

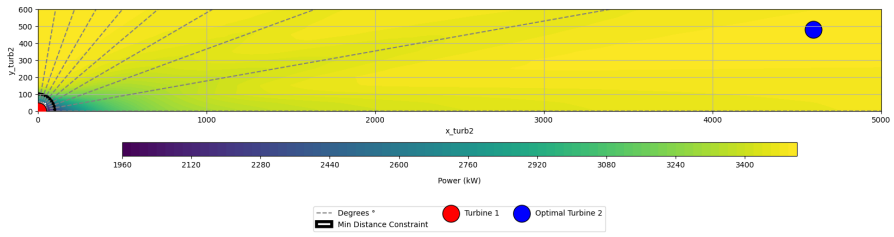

Figure 13: Stochastic optimization of the relative position of two windturbines relative to each other with wind normally distributed with mean μ =? and standard deviation σ =?, windspeed constant

Stochastic Optimization: Power function as Quantile Neural Network

An alternative approach to the one shown in the previous section is to implant the uncertainties of the wind condition into the power model, with the output as quantiles from a quantile neural network.

https://pubs.aip.org/aip/aml/article/2/1/016111/3265451/Predicting-wind-farm-wake-losses-with-deep

4.4 THE THREE TURBINE PROBLEM

4.5 THE N TURBINE PROBLEM

Thoughts:

IDEA: generalize inputs in a way that allow for power calculation of each turbine individually instead of farm power

IDEA : derive basic rules from 2 turbine problem and set up new optimization problem without NN

IDEA : analogue to forward/backward selection with 2 turbine neural network

FINIDINGS:

problem degenerate if not one wind turbine fixed

5 CONCLUSION

Once a proof of concept for the approach taken in this thesis has ben successfully performed, the data source may be eventually switched for experimental or real world data.

6

ADDENDA

7 ABBREVIATIONS

AEP	Anual Energy Production
MSE	Mean Squared Error
ReLu	Rectifier Linear Unit
IEA	International Energy Agency
MW	Mega Watt

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[1] European Environment Agency. Europe's onshore and offshore wind energy potential. Technical Report Technical report 6/2009, European Environment Agency, 2009. Accessed: 2025-04-03.

[2] Inimfon Akpabio and Serap Savari. On the construction of distribution-free prediction intervals for an image regression problem in semiconductor manufacturing. 03 2022.

[3] Antonio Alcántara and Carlos Ruiz. A neural network-based distributional constraint learning methodology for mixed-integer stochastic optimization. *Expert Systems with Applications*, 232:120895, 2023.

[4] Antonio Alcántara, Carlos Ruiz, and Calvin Tsay. A quantile neural network framework for two-stage stochastic optimization. *Expert Systems with Applications*, page 127876, 2025.

[5] Analysis and General Secretariat of the Council of the European Union Research Team. Harnessing wind power: Navigating the eu energy transition and its challenges. https://www.consilium.europa.eu/media/1kyk0wjm/2024_685_art_windpower_web.pdf, September 2024. Accessed: 2025-04-03.

[6] F. Azlan, J.C. Kurnia, B.T. Tan, and M.-Z. Ismadi. Review on optimisation methods of wind farm array under three classical wind condition problems. *Renewable and Sustainable Energy Reviews*, 135:110047, 2021.

[7] N. Bempedelis, F. Gori, A. Wynn, S. Laizet, and L. Magri. Data-driven optimisation of wind farm layout and wake steering with large-eddy simulations. *Wind Energy Science*, 9(4):869–882, 2024.

[8] John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer Series in Operations Research and Financial Engineering. Springer, July 1997.

[9] Alessio Bonfietti, Michele Lombardi, and Michela Milano. Embedding decision trees and random forests in constraint programming. 05 2015.

[10] Pietro Bortolotti, Helena Canet Tarrés, Katherine Dykes, Karl Merz, Latha Sethuraman, David Verelst, and Frederik Zahle. Iea wind task 37 on systems engineering in wind energy – wp2.1 reference wind turbines. Technical report, National Renewable Energy Laboratory, 2019.

[11] European Commission. Renewable energy targets. https://energy.ec.europa.eu/topics/renewable-energy/

renewable-energy-directive-targets-and-rules/renewable-energy-targets_en, 2023. Accessed: 2025-04-03.

[12] Héctor G. de Alba, Andres Tellez, Cipriano Santos, and Emmanuel Gómez. A reformulation to embedding a neural network in a linear program without integer variables, 2024.

[13] Luc De Raedt, Andrea Passerini, and Stefano Teso. Learning constraints from examples. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[14] Adejuyigbe O. Fajemisin, Donato Maragno, and Dick den Hertog. Optimization with constraint learning: A framework and survey. *European Journal of Operational Research*, 314(1):1–14, 2024.

[15] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. volume 15, 01 2010.

[16] Peng Hou, Jiangsheng Zhu, Kuichao MA, Guangya YANG, Weihao HU, and Zhe CHEN. A review of offshore wind farm layout optimization and electrical system design methods. *Journal of Modern Power Systems and Clean Energy*, 7(5):975–986, September 2019.

[17] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. *An Introduction to Statistical Learning: with Applications in Python*. Springer Texts in Statistics. Springer, 2023.

[18] Samuel Kainz, Julian Quick, Mauricio Souza de Alencar, Sebastian Sanchez Perez Moreno, Katherine Dykes, Christopher Bay, Michiel Zaaijer, and Pietro Bortolotti. Iea wind tcp task 55: The iea wind 740-10-mw reference offshore wind plants. Technical Report NREL/TP-5000-87923, National Renewable Energy Laboratory, 2024. Technical Report.

[19] Taewan Kim, Jeonghwan Song, and Donghyun You. Optimization of a wind farm layout to mitigate the wind power intermittency. *Applied Energy*, 367:123383, 2024.

[20] C.T. Kiranoudis and Z.B. Maroulis. Effective short-cut modelling of wind park efficiency. *Renewable Energy*, 11(4):439–457, 1997.

[21] Mads Madsen, Frederik Zahle, Sergio Gonzalez Horcas, Thanasis Barlas, and Niels Sørensen. Cfd-based curved tip shape design for wind turbine blades. *Wind Energy Science*, 7:1471–1501, 07 2022.

[22] M. Magnusson and A.-S. Smedman. Air flow behind wind turbines. *Journal of Wind Engineering and Industrial Aerodynamics*, 80(1):169–189, 1999.

[23] Grzegorz Marcjasz, Michał Narajewski, Rafał Weron, and Florian Ziel. Distributional neural networks for electricity price forecasting. *Energy Economics*, 125:106843, September 2023.

[24] Michael Nielsen. Neural networks and deep learning - chapter 1, 2015. Accessed: 2025-04-10.

[25] Michael Nielsen. Neural networks and deep learning - chapter 2: How the backpropagation algorithm works, 2015. Accessed: 2025-05-01.

[26] Michael Sinner and Paul Fleming. Robust wind farm layout optimization. *Journal of Physics: Conference Series*, 2767(3):032036, jun 2024.

[27] Ingo Steinwart and Andreas Christmann. Estimating conditional quantiles with the help of the pinball loss. *Bernoulli*, 17(1), February 2011.

[28] Zilong Ti, Xiao Wei Deng, and Hongxing Yang. Wake modeling of wind turbines using machine learning. *Applied Energy*, 257:114025, 2020.

[29] Zilong Ti, Xiao Wei Deng, and Mingming Zhang. Artificial neural networks based wake model for power prediction of wind farm. *Renewable Energy*, 172:618–631, 2021.

[30] Robert J. Vanderbei. *Linear Programming*, chapter 3. Springer, New York, NY, 5 edition, 2020.

[31] Li Wang, Mi Dong, Jian Yang, Lei Wang, Sifan Chen, Neven Duić, Young Hoon Joo, and Dongran Song. Wind turbine wakes modeling and applications: Past, present, and future. *Ocean Engineering*, 309:118508, 2024.

[32] Windpower Monthly. Offshore wind clusters to lower energy production – study. *Windpower Monthly*, 2019. Accessed: 2025-04-29.

[33] Kun Yang, Xiaowei Deng, Zilong Ti, Shanghui Yang, Senbin Huang, and Yuhang Wang. A data-driven layout optimization framework of large-scale wind farms based on machine learning. *Renewable Energy*, 218:119240, 2023.